

Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ
Факультет Информационных Технологий и Управления
Кафедра ИТАС
Аппаратно-программное обеспечение ЭВМ и сетей

Отчет по лабораторной работе №2
Технология администрирования корпоративных сетей на основе *Windows
PowerShell*

Выполнил
студент группы
820601 Шведов А.Р

Проверил
Ярмолик В.И.

Минск, 2021

1 ЦЕЛЬ РАБОТЫ

Освоение технологии администрирования корпоративных сетей *Microsoft Windows* средствами *PowerShell*.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1 Общая характеристика *PowerShell*

Недостатки традиционных средств автоматизации управления и администрирования *Microsoft Windows* (*cmd.exe*, *WMIC*, *WSH*), усугубляемые усложнением сетевых корпоративных технологий, привели к созданию *Windows PowerShell (PS)*. *Windows PowerShell* как технология достаточно проста и вместе с этим предоставляет широкие возможности системным инженерам и администраторам в деле автоматизации управления корпоративными сетями. *Windows PowerShell* как технология предоставляет практически неограниченные возможности по настройке серверов, виртуальных машин и сбору информации об их состоянии.

Истоки создания *PowerShell* (первоначальное название *Monad*) связаны с доработкой *WMIC* (*Windows Management Instrumentation Command*) для обеспечения доступа из командной строки к любым классам платформы *.NET Framework*. *Windows PowerShell* является и оболочкой командной строки и средой выполнения сценариев. Сценарии пишутся на новом языке. Язык сценариев *PowerShell* поддерживает управление объектами *.NET*, обладает совместимостью с языками, используемыми при программировании для *.NET* и имеет синтаксическое сходство с *C* (язык *PowerShell* по существу представляет собой упрощённый *C*); это обусловлено тем, что оболочка *PowerShell* основана на *.NET Framework*.

Windows PowerShell принципиально отличается от других оболочек тем, что PS обрабатывает не текст, а объекты платформы *.NET*. *Windows PowerShell* содержит встроенные команды (командлеты; *cmdlets*), которые имеют унифицированный интерфейс и обрабатываются одним синтаксическим анализатором. Командлет представляет собой команду, выполняющую одну единственную функцию.

2.2 Командная оболочка

Существует *Windows PowerShell* в двух ипостасях: помимо эмулятора консоли с командной оболочкой есть интегрированная среда сценариев

(*Integrated Scripting Environment — ISE*). Чтобы получить доступ к интерфейсу командной строки достаточно выбрать соответствующий ярлык в меню *Windows* или запустить *powershell.exe* из меню «Выполнить». На экране появится синее окошко, заметно отличающееся по возможностям от допотопного *cmd.exe*. Там есть автодополнение и другие фишки, привычные пользователям командных оболочек для *Unix*-систем.

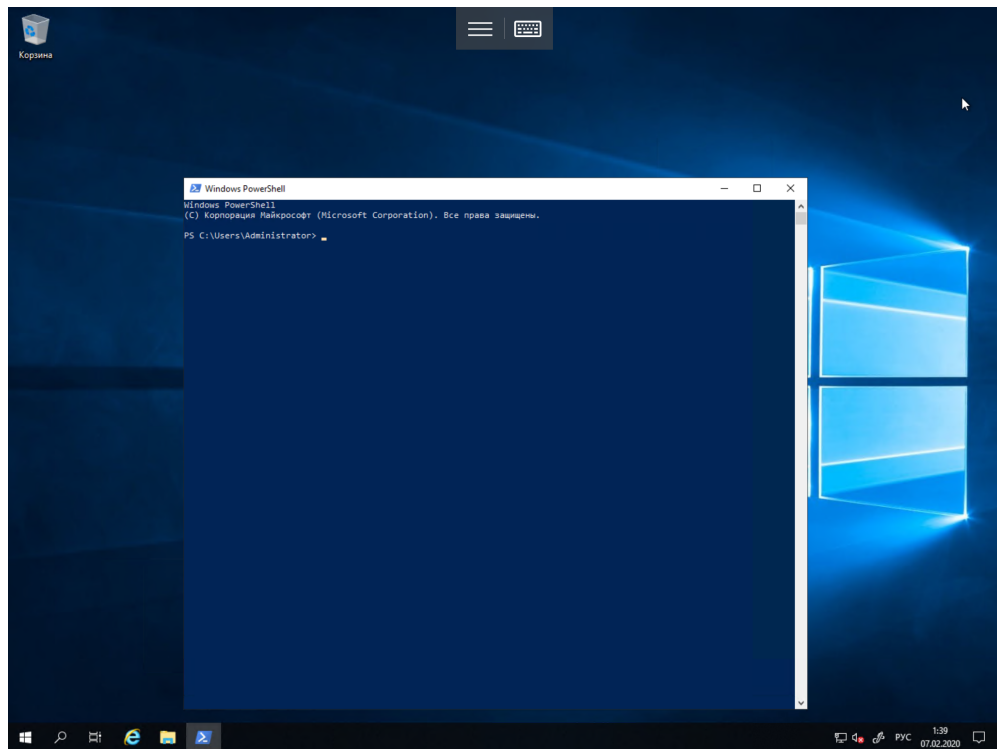


Рисунок 2.1 – Командная оболочка *Windows PowerShell*

2.3 Командлеты

В *Windows PowerShell* появились т.н. командлеты (*cmdlets*). Это специализированные классы *.NET*, в которые заложена разнообразная функциональность. Именуются они по принципу «Действие-Объект» (или «Глагол-Существительное, если вам так больше нравится), а разделенная дефисом связка напоминает сказуемое и подлежащее в предложениях естественных языков. Например, *Get-Help* буквально означает «Получить-Помощь» или в контексте *PowerShell*: «Показать-Справку». По сути это аналог команды *man* в *Unix*-системах и мануалы в *PowerShell* нужно запрашивать именно так, а не вызывая командлеты с ключом *-help* или */?..*

Не стоит забывать и об онлайн-документации по *PowerShell*: у *Microsoft* она достаточно подробная.

Помимо *Get* в командлетах для обозначения действий используются и другие глаголы (и не только глаголы, строго говоря). В списке ниже мы приведем несколько примеров:

- 1 *Add* — добавить;
- 2 *Clear* — очистить;
- 3 *Enable* — включить;
- 4 *Disable* — выключить;
- 5 *New* — создать;
- 6 *Remove* — удалить;
- 7 *Set* — задать;
- 8 *Start* — запустить;
- 9 *Stop* — остановить;
- 10 *Export* — экспортировать;
- 11 *Import* — импортировать.

Есть системные, пользовательские и опциональные командлеты: в результате выполнения все они возвращают объект или массив объектов. К регистру они не чувствительны, т.е. с точки зрения интерпретатора команд нет разницы между *Get-Help* и *get-help*. Для разделения используется символ ‘;’, но ставить его обязательно только если в одной строке выполняется несколько командлетов.

Командлеты *Windows PowerShell* группируются в модули (*NetTCPIP*, *Hyper-V* и т.д.), а для поиска по объекту и действию существует командлет *Get-Command*. Показать справку по нему можно так: *Get-Help Get-Command*.

По умолчанию команда отображает краткую справку, но в командлеты при необходимости передаются параметры (аргументы). С их помощью можно, например, получить детальную (параметр *-Detailed*) или полную (параметр *-Full*) справку, а также вывести на экран примеры (параметр *-Examples*):

2.4 Типы команд

Оболочка PS поддерживает следующие типы команд:

1. **Внешние исполняемые файлы** — это обычные, выполняемые операционной системой файлы. Для ознакомления с этими командами вве-

дите в окне *CMD* команду *help*. Начинать практическое освоение *PS* следует именно с этих команд.

2. **Командлет** используется внутри *PowerShell*. Командлеты являются классами *.NET*, порожденными базовым классом *CMDLET*, который гарантирует их совместимый синтаксис. Командлеты компилируются в динамическую библиотеку *DLL* и подгружаются к процессу *PS* во время запуска оболочки *PS*. Каждый из командлетов (очень простой или сложный) выполняет узкую задачу. Посредством вертикальной черты (*|*) командлеты могут быть организованы в конвейер, в котором объекты предаются от одного командлета к другому.

3. **Функция** в *PowerShell* представляет собой блок кода на языке *PS*, имеющий название и хранящийся в памяти до завершения текущего сеанса командной строки. Как и в других языках программирования в *PS* при описании функции можно задавать список формальных параметров, значения которых при выполнении функции будут заменены значениями переданных аргументов. Сценарии. Сценарий представляет собой код на языке *PS*, хранящийся во внешнем файле с расширением *ps1*. Важной особенностью в плане сетевой безопасности является то обстоятельство, что скрипт с расширением *ps1* невозможно запустить на исполнение иначе как, запустив оболочку *PS*; так что для запуска сценария с расширением *ps1* необходимо запустив оболочку *PS*, ввести имя файла и нажать клавишу *Enter*.

4. **Сценарии** позволяют работать с *PowerShell* в пакетном режиме с заранее созданными командами на основе управляющих инструкций языка *PowerShell*. Сценарии в *PowerShell* в отличие от сценариев *WSH* и командных файлов *cmd.exe* можно писать непосредственно в самой оболочке, перенося затем готовый отлаженный код во внешний файл.

2.5 Среда разработки

Windows PowerShell ISE является полноценной средой разработки с поддерживающим вкладки и подсветку синтаксиса редактором кода, конструктором команд, встроенным отладчиком и другими программистскими радостями. Если в редакторе среды разработки после имени команды написать знак дефис, вы получите в выпадающем списке все доступные параметры с указанием типа. Запустить *PowerShell ISE* можно либо через ярлык из системного меню, либо с помощью исполняемого файла *powershell_ise.exe*.

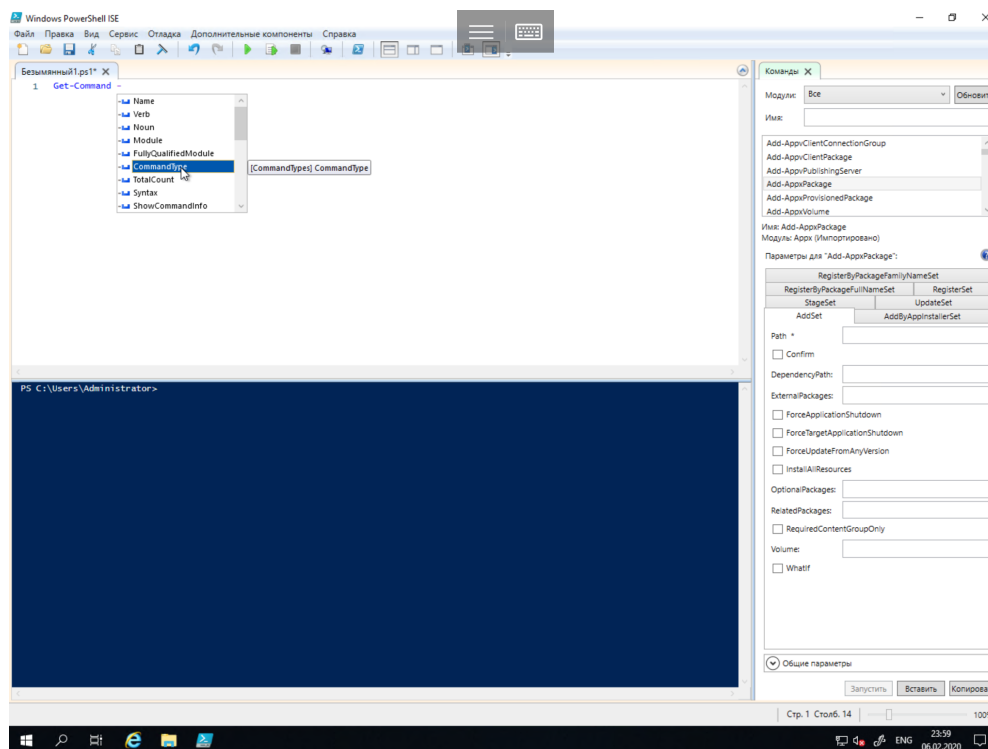


Рисунок 2.2 – *Windows PowerShell ISE*

3 ХОД РАБОТЫ

Так как выполнять базовые операции и вызовы командлетов не очень интересно, рассмотрим следующие задачи:

3.1 Преобразование данных о процессе в *html*, сохранение в файле и просмотр результатов

Для преобразования данных в формат *html* служит командлет *Convertto-html*. Параметр *Property* определяет свойства объектов, включаемые в выходной документ. Например, для получения списка выполняемых процессов в формате *html*, включающего имя процесса и затраченное время *CPU* и записи результата в файл *processes.html* можно использовать команду *Get-Process*. Для просмотра содержимого файла можно использовать командлет *Invoke-Item*

```
1 Get-Process | Convertto-html -Property Name, CPU > Processes.html
2 Invoke-Item Processes.html
```

Рисунок 3.1 – Листинг кода

3.2 Инвентаризация и диагностика *Windows*-компьютеров

Получим сведения о ПК:

```
1 Get-Wmiobject -Class Win32_Processor | Format-List *.
```

Рисунок 3.2 – Листинг кода

3.3 Получение информации о процессах

Выведем список названий и занятую виртуальную память (в *Mb*) каждого процесса, разделённые знаком тире, при этом если процесс занимает более *100Mb* – выводить информацию красным цветом, иначе зелёным.

```
1 Get-Process | %{  
2     if ($_.VM -gt 100mb){  
3         Write-Host ($_.ProcessName, $_.VM) -separator "-" -BackgroundColor Red  
4     } else {  
5         write-Host ($_.ProcessName, $_.vm) -separator "-" -BackgroundColor green  
6     }  
7 }
```

Рисунок 3.3 – Листинг кода



```
csrss-4294967295  
csrss-4294967295  
dashost-4294967295  
dihost-4294967295  
dm-4294967295  
esif_assist_64-47521792  
esif_uf-40828928  
ETDctrl-146907136  
ETDctrlHelper-103321600  
ETDService-31973376  
ETDTouch-90001408  
explorer-4294967295  
firefox-4294967295  
firefox-4294967295  
firefox-4294967295  
firefox-4294967295  
firefox-4294967295  
fontdrvhost-4294967295  
fontdrvhost-4294967295  
googlecrashhandler-85835008  
googlecrashhandler64-67518464  
googledrivesync-49786880  
googledrivesync-422621184  
jmsvc-1500864
```

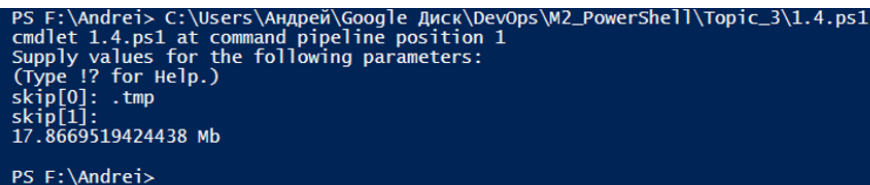
Рисунок 3.4 – Результат работы программы

3.4 Получение информации о файлах в папке

Подсчитаем размер занимаемый файлами в папке (например *C:*) за исключением файлов с заданным расширением (например *.tmp*) с помощью передачи параметра.

```
1 [CmdletBinding()]
2 Param(
3     [parameter(Mandatory = $true, HelpMessage = "Enter extension")]
4     [string[]]$skip
5 )
6
7 $sum = 0
8
9 Get-ChildItem "C:\Windows" -force | %{ if($_.GetType().Name -eq "FileInfo" -and !$skip
10     .Contains($_.Extension)) {$sum += $_.Length/1Mb}}
11 "$sum_Mb"
```

Рисунок 3.5 – Листинг кода



```
PS F:\Andrei> C:\Users\Андрей\Google Диск\DevOps\M2_PowerShell\Topic_3\1.4.ps1
cmdlet 1.4.ps1 at command pipeline position 1
Supply values for the following parameters:
(Type !? for Help.)
skip[0]: .tmp
skip[1]:
17.8669519424438 Mb

PS F:\Andrei>
```

Рисунок 3.6 – Результат работы программы

4 ЗАКЛЮЧЕНИЕ

Мы изучили основы оболочки *PowerShell* для администрирования корпоративных сетей *Microsoft Windows*, ее основные отличия от командной оболочки *CMD*, а также принципиальные различия сценариев *WSH* и *PS*.