ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Кафедра вычислительных систем

ПРАКТИЧЕСКАЯ РАБОТА
«Изучение командной оболочки PowerShell»

Автор: С.Н. Мамойленко

Новосибирск - 2019

Оглавление

1. Введение....3
2. История создания командной оболочки PowerShell....3
3. Основные принципы работы с PowerShell....3
3.1. Командлет (cmdlet)....6
2.2. Псевдоним команды (alias)....7
2.3. Функция (function)....7
2.4. Числовые и символьные константы в командной строке....8
2.5. Переменные, объекты и подстановка значений в командной строке....9
2.6 Приоритезация выполнения команд....11
2.7. Операторы командной строки....11
2.8. Конвейеризация команд....13
4. Сценарии....15
5. Источники информации....15

1. Введение

Каждому разработчику программного обеспечения часто приходится сталкиваться с задачей автоматизации рутинных рабочих процессов, в которых необходимо выполнять множество однотипных взаимодействий с операционной системой. Примером таких рутинных операций могут служить задачи переименования файлов, расположенных в нескольких директориях, используя определённый шаблон или установка (развертывание) приложения. Очевидно, что эти операции человек вполне может сделать самостоятельно, если таких файлов не очень много или операция установки приложения делается один раз. А если файлов сотни или приложение необходимо установить на сотню рабочих мест? Тогда невозможно обойтись без автоматизации.

Способ автоматизации, конечно же, зависит от механизмов, которые использует разработчик для взаимодействия с операционной системой. Очевидный способ – разработать собственное специальное программное обеспечение, которое бы справилось с задачей. По сути, в дальнейшем речь пойдет так же о разработке программного обеспечения, но с использованием существующих средств. Речь пойдет о командных оболочках – специальном системном программном обеспечении, используемом пользователями для взаимодействия с операционной системой ЭВМ или вычислительной системы.

2. История создания командной оболочки PowerShell

Популярная операционная система Microsoft Windows изначально предлагала пользователю для взаимодействия с ней использовать графический визуальный интерфейс. Такой интерфейс имеет минимальный набор средств автоматизации (по сути, это только привязка каких-то действий к нажатию определённого сочетания клавиш). Для работы администраторов и разработчиков операционная система продолжала поддерживать «старый» командный интерфейс (command.com и затем cmd.exe) и активно разрабатывала иные средства автоматизации. «Старый» командный интерфейс поддерживал ограниченное количество встроенных средств, которые никак не адаптировались под изменения самой операционной системы, и поэтому разработка новых механизмов набирала популярность. Альтернативой командной оболочки в операционной системе Windows служили: Windows Script Host, netsh, Windows Management Instrumentation Console и т.д. Все эти средства никак не были связаны с командной оболочкой (для неё это были внешние утилиты) и зачастую плохо документированы.

В 2003 компания Microsoft начала разработку новой оболочки, называемой Monad (также известной как Microsoft Shell или MSH). Эта оболочка должна была стать новой расширяемой оболочкой командой 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. В 2008 году в свет вышла вторая версия командной оболочки PowerShell, и она стала доступной для систем Windows 7 и Windows Server 2008 R2 одновременно с их выпуском. В дальнейших версиях операционной системы Windows командная оболочка PowerShell входит в дистрибутив и предлагается как альтернатива «старой» cmd.exe.

Оболочка PowerShell базируется на платформе CLR .NET. Поэтому основной принцип работы с оболочкой – это манипулирование различными объектами (данными, элементами операционной системы, приложениями, пользователями и т.п.). Для расширения оболочки новым функционалом достаточно разработать свой класс объектов, базирующийся на .NET, сформировать динамическую библиотеку и подключить её к PowerShell[1].

3. Основные принципы работы с PowerShell

Командная оболочка PowerShell запускается как обычное приложение. По умолчанию это приложение располагается в директории `C:\Windows\system32\Windows Power Shell`[2]. Запустить PowerShell возможно в двух режимах (см. рисунок 1): простой командный режим (файл powershell.exe) и режим интегрированной среды разработки сценариев (powershell-ise.exe). Отличаются эти два режима только тем, что во втором случае имеется средства, упрощающие разработку сценариев (имеются

[1] Разрабатываются командлеты с помощью пакета PowerShell Software Developers Kit (SDK), который можно загрузить с официального сайта Microsoft.

[2] Для версий Windows старше Windows Vista оболочку PowerShell необходимо устанавливать отдельно, скачивая её с официального сайта - `http://microsoft.com/powershell`.

дополнительные области для редактирования скрипта, выбора команд). В остальном принципы работы с оболочкой абсолютно одинаковые.

Запуск командной строки возможен традиционными для Windows способами:

- прямой запуск приложения или запуск с использованием ярлыка – мышкой выбирается файл powershell.exe, расположенный в директории C:\Windows\system32\Windows Power Shell\v1.0 (см. рисунок 2);
- через меню «Пуск» (поиском в самом меню или через контекстное меню, см. рисунок 3).

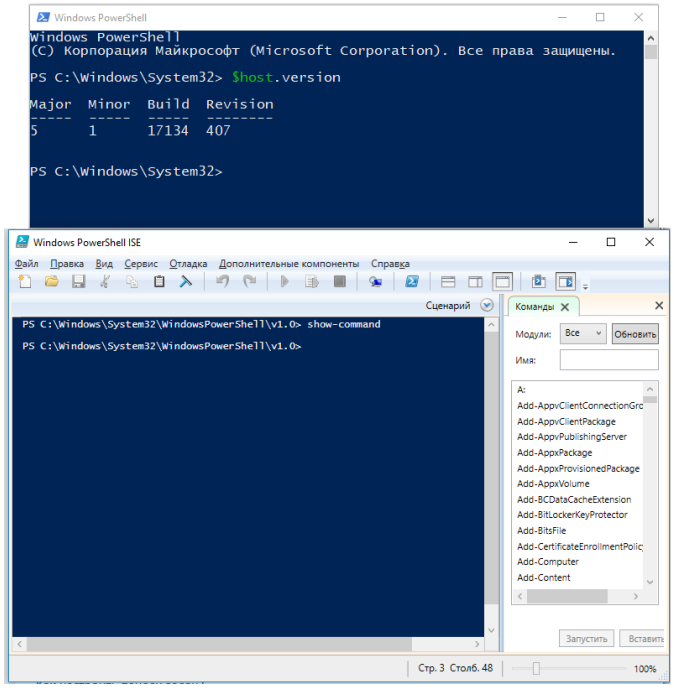


Рисунок 1 – виды запуска командной оболочки PowerShell (сверху – простой командный режим, снижу – режим интегрированной среды разработки сценариев)

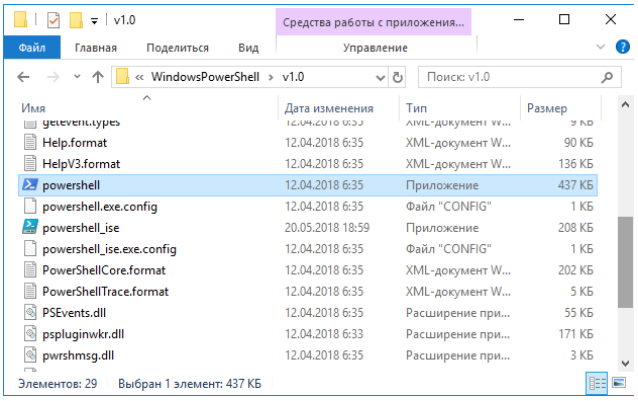
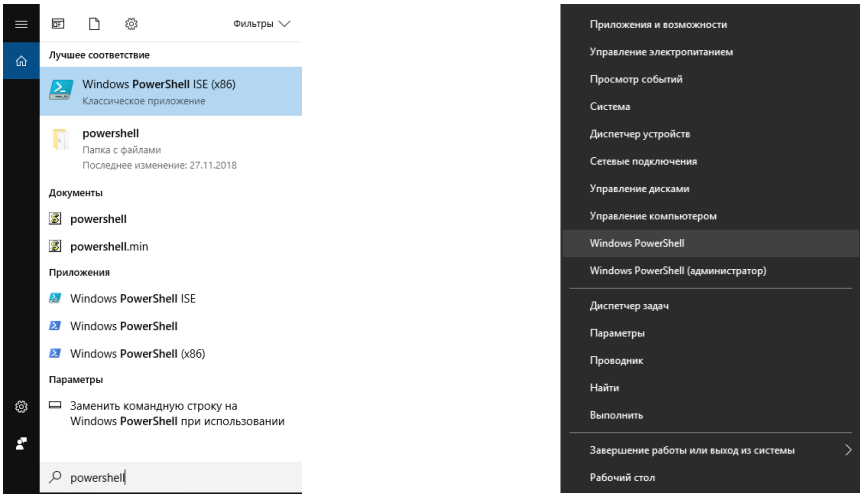


Рисунок 2 – Запуск PowerShell традиционным способом



а) б)

Рисунок 3 – Запуск PowerShell с использованием интерфейса Windows (а – через меню «Пуск» с использованием поиска, б – через контекстное меню, вызываемое нажатием комбинации клавиш Win+X[3])

Командная оболочка PowerShell является интерпретатором, который «получает на вход» командную строку, анализирует её и выполняет все то, что в ней написано.

[3] Чтобы в контекстном меню, вызываемом нажатием комбинации клавиш Win+X, появилась оболочка PowerShell необходимо в настройках системы (Параметры->Персонализация->Панель задач) выбрать пункт «Заменить командную строку оболочкой Windows PowerShell в меню, которое появляется при щелчке правой кнопкой мыши на меню «Пуск» или при нажатии клавиш Windows+X). В противном случае в этом меню будет доступна классическая «старая» командная оболочка.

PowerShell может работать в двух режимах: интерактивной и пакетном. В интерактивном режиме интерпретация командной строки начинается после того, как пользователь нажмет на клавиатуре клавишу Enter. В пакетном режиме, когда командная оболочка считывает командные строки из файла, называемом *«скриптом»*, интерпретация командных строк начинается после того, как встретиться символ завершения строки.

Нажатие клавиши Enter в интерактивном режиме или появление символа конца строки в пакетном режиме не приводит к началу интерпретации команды в случае, если в полученной командной строке содержится непарное количество скобок, кавычек или апостофов. В этом случае введенные строки будут соединяться в одну командную строку до тех пор, пока не появится соответствующий «закрывающий» парный символ.

Командная строка – это последовательность символов из заданного алфавита, имеющая определённую структуру, например:

```
1 ‖ PS> команда аргумент1 -параметр1 аргумент2 -параметр2 аргумент3 аргумент4
```

Поскольку PowerShell по своей сути является некоторым подобием языка программирования, то алфавитом командных строк являются печатные (имеющие визуальное отображение) символы и знаки. Некоторые символы, при этом, могут иметь специальное значение или могут быть недопустимыми в зависимости от места, в котором они встречаются в командной строке (например, &, \, символы национальных языков и т.п.).

В структуре командной строки содержится несколько полей, разделённых между собой знаком пробела. В каждой командной строке присутствует как минимум одно поле.

Первым полем в командной строке является *команда*. Это поле интерпретируется как указание к действию (то, что должна выполнить командная оболочка в соответствие с этой командной строкой). Командами могут быть: командлеты, псевдонимы, функции, внешние приложения. Регистр символов, указанных в команде, значения не имеет.

Остальные поля командной строки – это параметры(опции) или аргументы. Эти поля являются входными данными для команды. *Параметр (опция)* – это часть командной строки, начинающаяся с символа − (тире). Остальные поля – это *аргументы*. Смысловое значение параметров (опций) не зависит от порядкового номера полей, в которых они указаны, в то время как значение аргумента зависит от этого. Аргумент может быть *аргументом параметра*, если он указывается в поле, следующим за параметром и параметр требует аргумент (в примере – это аргумент2) или *аргументом команды* - если перед указанием аргумента не указан параметр или параметр указан, но он не требует указания аргумента (в примере это аргумент1, аргумент3, аргумент4).

Результатом выполнения любой командной строки является некоторый объект. Тип объекта и его содержание определяется назначением команды.

Допускается вместо команды в командной строке вводить поле, содержащее значение в одном из простейших типов данных: Int32, Double, String. В этом случае результатом выполнения командной строки будет формирование объекта заданного класса с указанным значением. Значение с типом String вводится в кавычках или апострофах.

В интерактивном режиме при вводе команды доступны традиционные механизмы редактирования строки: движение курсора по строке, удаление символов, вставка символов в строку, использование буфера обмена т.п.

Все команды, вводимые в рамках одно сеанса работы с командной строкой, сохраняются в истории. Чтобы перемещаться по истории команд используются клавиши управления курсором «Вверх» и «Вниз».

Оболочка имеет механизм подбора команд. Для этого достаточно набрать несколько первых символов команды и нажать клавишу <TAB>. После этого оболочка предложит первую подходящую команду. Если она не устраивает, нажимайте <TAB> до тех пор, пока не появится нужная команда. Комбинация клавиш <Shift>+<Tab> позволяет вернуться к предыдущему предлагаемому варианту команды.

Чтобы получить список доступных команд с указанием их параметров, необходимо выполнить команду show-command. В появившемся окне будет представлен перечень доступных команд и при выборе каждой команды будет предложен список полей. Заполнив все поля и выбрав команду в оболочку будет передана соответствующая командная строка.

Чтобы получить доступ к справочной системе Windows PowerShell необходимо выполнить команду `Get-Help`.

3.1. Командлет (cmdlet)

Основное тип команд PowerShell – *командлет* (от англ. Cmdlet). Этот термин используется потому, что командлет представляет собой класс .NET, порожденный от базового класса Cmdlet.

Имя каждого командлета имеет строго определённую структуру – «Глагол-существительное». Глагол определяет действие, которое необходимо выполнить, например `Get` – получить, `Set` – установить, `Add` – добавить и т.п. Существительное определяет объект, над которым необходимо выполнить действие, например `Service` – системная служба, `Host` – узел и т.п.

Важно отметить, что PowerShell имеет встроенную документацию по каждому командлету. Для того, чтобы получить перечень доступных в текущем сеансе командлетов необходимо выполнить команду `Get-Command` (см. рисунок 4). По умолчанию (при запуске команды без параметров) будет выдана информация о функциях, командлетах и алиасах. Если указать первым аргументом `*`, то также будет выдана информация и о доступных внешних командах (приложениях, найденных во всех путях, указанных в переменной среды окружения PATH).

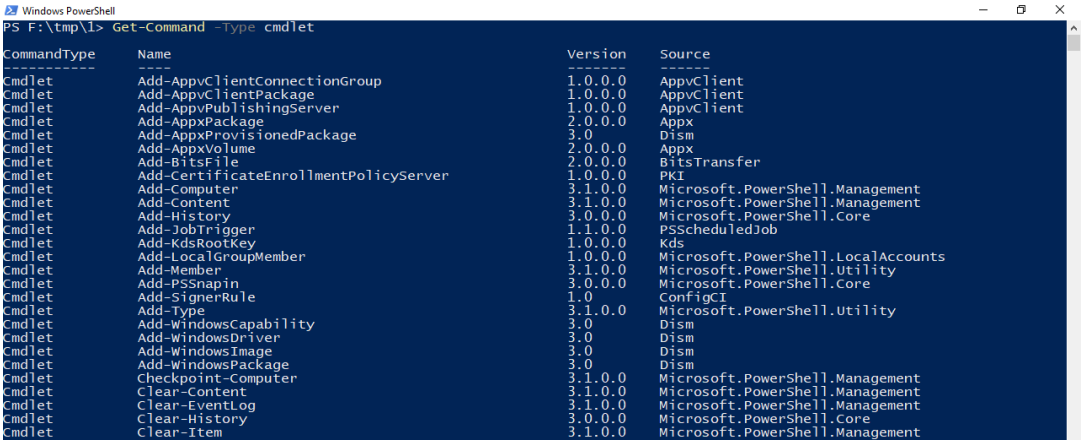


Рисунок 4 – Результат выполнения командлета Cmd-Command

Поскольку все командлеты являются потомками базового класса Cmdlet, они все имеют общие параметры (см. таблицу 1). Все командлеты поддерживают эти параметры, но не все реализуют необходимые для них действия.

Таблица 1. Общие параметры командлетов

Параметр	Тип	Описание
`-Verbose`	`Boolean`	Включает режим вывода подробных сведений об операции.
`-Debug`	`Boolean`	Включает режим создания подробного отчета об операции на уровне программирования.
`-ErrorAction`	`Enum`	Определяет, какой будет реакция командлета на возникновение ошибки. Возможные значения: Continue (по умолчанию), Stop, SilentlyContinue, Inquire.
`-ErrorVariable`	`String`	Определяет переменную, в которой будут сохраняться ошибки команды, возникшие во время выполнения.
`-OutVariable`	`String`	Задает переменную, в которой будут сохраняться выходные данные команды, формируемые во время выполнения.
`-OutBuffer`	`Int32`	Определяет количество хранящихся в буфере объектов перед вызовом следующего командлета в конвейере.
`-WhatIf`	`Boolean`	Предоставляет сведения об изменениях, которые произойдут в результате указанных действий, не производя самих этих действий. Данный параметр поддерживается командлетами в том случае, если они изменяют состояние системы
`-Confirm`	`Boolean`	Запрашивает разрешение у пользователя на выполнение каких-либо действий, вносящих изменения в систему. Данный параметр поддерживается командлетами в том случае, если они изменяют состояние системы

2.2. Псевдоним команды (alias)

Для обеспечения совместимости PowerShell со «старым» командным интерфейсом и для удобства работы пользователя в оболочке имеется возможность создания псевдонимов (алиасов) команд.

Перечень существующих псевдонимов можно получить с помощью команды `Get-Command -Type Alias` или командой `Get-Alias`.

Создать новый псевдоним можно с помощью команды `Set-Alias`, в которой имя псевдонима задается как аргумент параметра `Name`, а команда, которая будет выполняться в случае вызова псевдонима – аргумент параметра `Value`.

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

Следует отметить, что псевдоним может быть лишь альтернативным именем для команды и не может «скрывать» параметры вызова этой команды. Если необходимо создать простое имя для длинной команды, имеющей кучу параметров и аргументов, то можно использовать функцию (см. ниже).

Для удаления псевдонима используется команда Remove-Item.

```
PS> Remove-Item alias:gl
```

Псевдонимы можно экспортировать в текстовый файл (командлет `Export-Alias`) и импортировать их из файла (командлет `Import-Alias`).

Благодаря псевдонимам синтаксис PowerShell приобретает гибкость: одни и те же команды могут быть записаны и очень кратко, и в развернутом виде. Это очень важно для языка, который одновременно является оболочкой командной строки и языком написания сценариев. При интерактивной работе для ускорения ввода команд удобнее применять краткие псевдонимы и не полностью указывать имена параметров. Если же вы пишете сценарии, то лучше использовать полные названия командлетов и их параметров, это значительно упростит в дальнейшем разбор программного кода.

2.3. Функция (function)

Для выполнения нескольких действий, возможно даже зависящих от определённых параметров, в PowerShell возможно использовать **функции** - блоки кода на языке PowerShell, имеющие название и находящиеся в памяти до завершения текущего сеанса командной оболочки.

Функция создается командой `function` следующим образом:

```
PS> function MyFunc {
PS> Get-Location
PS> Get-Host
PS> }
```

После этого в системе создается команда `MyFunc` при выполнении которой будет сначала выполнена команда `Get-Location`, а затем команда `Get-Host`. Обратите внимание, что тело функции оформлено как блок между фигурными скобками. Такое оформление последовательности команд используется и в других случаях, где надо в качестве аргументы указать последовательность выполняемых действий.

Функции могут иметь параметры, указываемые при определении функции в скобках после имени функции:

```
PS> function MyFunc ($par) {
PS> Get-Location -PSDrive $par
PS> Get-Host
PS> }
```

Для обозначения параметров используется знак $ (доллар). Такой же способ используется для обозначения переменных (см. ниже).

Следует отметить, что проверка синтаксиса функции выполняется PowerShell один раз при определении функции.

2.4. Числовые и символьные константы в командной строке

Командная оболочка PowerShell умеет распознавать в командной строке основные типы данных платформы .NET: System.Int32, System.Int64, System.Double, System.String. При этом явно задавать тип данных нет необходимости — система сама выбирает подходящий тип для указываемого объекта.

Для упрощения ввода констант предусмотрены специальные суффиксы-множители, часто используемыми системными администраторами: килобайтами, мегабайтами и гигабайтами.

```
PS> 10+9.35
19,35
PS> 10kb+1Mb
1058816
PS> 2.5Gb+10Mb
2694840320
PS>
```

Символьные строки являются объектами типа System.String и представляют собой последовательность 32-битовых символов в кодировке Unicode. Все строки в командной сроке заключаются в кавычки или апострофы. При этом кавычки и апостровы могут располагаться внутри самих строк, если либо они отличаются от символов, в которые заключена сама строка (например: `'это строка "с кавычками"'`), либо символ кавычки или апострофа «экранирован» - повторен дважды подряд.

Для формирования многострочной строки используется комбинация `@"<enter>текст<enter>"@`:

```
PS> @"
>>Это многострочная
>>строка
>>в которой можно размещать что угодно
>>включая "текст в любых кавычках"
>>"@
PS>
```

Исключением для многострочного текста является интерпретация символа `$` как признака следующего за ним блока кода для вычисления (или подстановки значения переменной).

```
PS> @"
>> 1 Первая строка
>> $(1+1) вторая строка
>> $(1+2) третья строка
>> "@
1 Первая строка
2 вторая строка
3 третья строка
PS>
```

Все константы – это объекты определённого класса и, соответственно, имеют значение и определённые интерфейс (набор методов). Узнать перечень методов, входящих в интерфейс объекта, можно с помощью командлета Get-Member (о конвейерах читайте ниже):

```
PS> 2 | Get-Member
   TypeName: System.Int32
Name        MemberType Definition
----        ---------- ----------
CompareTo   Method     int CompareTo(System.Object value)...
Equals      Method     bool Equals(System.Object obj)...
GetHashCode Method     int GetHashCode()...
...
PS>
```

В таблице 2 приведено описание некоторых методов.

Таблица 2. Методы из интерфейсов констант

Метод	Тип объекта	Описание
CompareTo	Все	Сравнивает значение текущего экземпляра с указанным экземпляром. Возвращает -1, если значение объекта меньше указанного, 0 – если равно и 1, если больше
Equals	Все	Возвращает True если два объекта имеют одинаковое значение и False в противном случае
GetType	Все	Возвращает тип объекта
GetTypeCode	Все	Возвращает идентификатор типа объекта
ToInt16	Все	Преобразует значение в объект с типом Int16
ToString	Все	Преобразует в строковый объект
IndexOf	String	Возвращает индекс первого вхождения указанного символа
IndexOfAny	String	Возвращает индекс первого вхождения одного из указанных символов
Split	string	Разделить строку на подстроки по указанному разделителю
Chars	string	Возвращает символ строки, расположенный по указанному индексу
ToUpper	string	Изменяет регистр букв на заглавные
ToLower	string	Изменяет регистр букв на строчные
Replace	string	Заменяет указанные символы на новые
Insert	string	Вставляет подстроку в строку с указанного места
Length	String	Возвращает длину строки в символах
ToCharArray	String	Возвращает массив символов из символов строки

2.5. Переменные, объекты и подстановка значений в командной строке

В командной строке PowerShell символ $ интерпретируется как указание заменить следующий за ним текст соответствующим объектом. Создать объект с определёнными именем можно с помощью оператора присваивания =. Тип объекта определяется автоматически и может меняться в время выполнения командных строк.

```
PS> $a="C:\"
PS> Get-ChildItem $a
    Каталог: C:\
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        30.11.2018      0:39                CareLink Data
d-----        31.05.2018     22:10                Intel
d-----        12.04.2018      6:38                PerfLogs
d-r---        07.01.2019      1:44                Program Files
d-r---        07.01.2019      1:22                Program Files (x86)
d-r---        03.07.2018     23:02                Users
d-----        18.12.2018     10:27                Windows
PS>
```

Напомним, что все элементы, которыми манипулирует командная оболочка PowerShell – это экземпляры определённого класса (объекты). Как известно, объекты имеют определенный интерфейс, состоящие из методов и свойств. Получать доступ к интерфейсу объекта можно традиционным образом, добавляя после имени объекта символ . (точка) и указывая после него элемент интерфейса (метод или свойство). Например:

```
PS> $a="Строка. Объект типа Net.String"
PS> $a.Length
30
PS> $a.ToLower()
строка. объект типа net.string
PS> $a.ToUpper()
СТРОКА. ОБЪЕКТ ТИПА NET.STRING
PS> ”Это тоже объект”.Length
```

```
15
PS>
```

Получить перечень всех переменных можно с помощью командлета Get-ChildItem с указанием в качестве аргумента хранилище с типом Variable (признак хранилища – символ : (двоеточие) в конце):

```
PS> Get-ChildItem Variable:
…
```

В качестве переменных могут использоваться **массивы**. Для указания перечня элементов массива они перечисляются через запятую. Для записи в массив последовательности из целых чисел можно использовать запись `<начальное значение>..<конечное значение>`. Обращение к элементам массива осуществляется с использованием индексации с указанием номера элемента массива в квадратных скобках:

```
PS> $a=1,2,3,5
PS> $b=1..6
PS> $a
1
2
3
5
PS> $b
1
2
3
4
5
6
PS> $b[3]
4
PS> $a[-1]
5
PS> $a[2..3]
3
5
PS> $b[1,3,4]
2
4
5
PS>
```

Допускается использование **ассоциативных массивов**. Для этого в качестве индексов используется символьные строки. Пустой ассоциативный массив создается записью - `$a=@{}`.

Внутри оболочки PowerShell имеются служебные переменные, позволяющие управлять её работой (ст. таблицу 2).

Таблица 2. Перечень служебных переменных PowerShell

Переменная	Описание
$$	Содержит последнюю лексему последней полученной оболочкой строки
$?	Показывает, успешно ли завершилась последняя операция
$^	Содержит первую лексему последней полученной оболочкой строки
$_	При использовании в блоках сценариев, фильтрах и инструкции Where содержит текущий объект конвейера
$Args	Содержит массив параметров, передаваемых в функцию
$DebugPreference	Указывает действие, которое необходимо выполнить при записи данных с помощью командлета Write-Debug
$Error	Содержит объекты, для которых возникла ошибка при обработке в командлете

Таблица 2. Перечень служебных переменных PowerShell

Переменная	Описание
$ErrorActionPreference	Указывает действие, которое необходимо выполнить при записи данных с помощью командлета Write-Error
$ForEach	Обращается к итератору в цикле ForEach
$Home	Указывает домашний каталог пользователя.
$Input	Используется в блоках сценариев, находящихся в конвейере
$LASTEXITCODE	Содержит код завершения последнего выполнения исполнимого файла Win32
$PSHome	Показывает имя каталога, в который установлен PowerShell
$Host	Содержит сведения о текущем узле
$OFS	Используется в качестве разделителя при преобразовании массива в строку. По умолчанию данная переменная имеет значение пробела
$ReportErrorShowExceptionClass	Если значение переменной равно $True, показывает имена класса выведенных исключений
$ReportErrorShowInnerException	Если значение переменной равно $True, показывает цепочку внутренних исключений. Вывод каждого исключения управляется теми же параметрами, что и корневого исключения, то есть параметры $ReportErrorShow* используются для вывода каждого исключения
$ReportErrorShowSource	Если значение переменной равно $True, показывает имена сборок выведенных исключений
$ReportErrorShowStackTrace	Если значение переменной равно $True, происходит трассировка стека исключений
$StackTrace	Содержит подробные сведения трассировки стека на момент последней ошибки
$VerbosePreference	Указывает действие, которое нужно выполнить, если данные записываются с помощью командлета Write-Verbose
$WarningPreference	Указывает действие, которое необходимо выполнить при записи данных с помощью командлета Write-Warning в сценарии

2.6 Приоритезация выполнения команд

Командная оболочка PowerShell позволяет изменить порядок анализа и интерпретации полей командной строки. Делается это традиционным образом – с использованием блоков внутри круглых скобок. Команды будут исполняться в соответствии с вложенностью блоков. Каждый блок возвращает один объект или массив объектов.

```
PS> (1,2,3).gettype()
IsPublic IsSerial Name                BaseType
-------- -------- ----                --------
True     True     Object[]            System.Array
PS>
```

2.7. Операторы командной строки

Оболочка PowerShell поддерживает ряд операторов, которые позволяют выполнить определённые действия над объектами: сравнение, арифметические операции, объединение.

Перечень операторов сравнения представлен в таблице 3.

Таблица 3. Перечень логических операторов и операторов сравнения

Оператор	Значение	Пример (возвращается значение True)
-eq	равно	`10 -eq 10`
-ne	не равно	`9 -ne 10`
-lt	меньше	`3 -lt 4`

-le	меньше или равно	`3 –le 4`
-gt	больше	`4 -gt 3`
-ge	больше или равно	`4 -ge 3`
-like	сравнение на совпадение с учетом подстановочного знака во втором операнде	`"file.doc" –like "f*.doc"`
-notlike	сравнение на несовпадение с учетом подстановочного знака во втором операнде	`"file.doc" –notlike "f*.rtf"`
-contains	содержит	`1,2,3 –contains 1`
-notcontains	не содержит	`1,2,3 –notcontains 4`
-and	логическое И	`(10 -eq 10) –and (1 –eq 1)`
-or	логическое ИЛИ	`(9 -ne 10) –or (3 –eq 4)`
-not	логическое НЕ	`-not (3 –gt 4)`
!	логическое НЕ	`!(3 -gt 4)`

Арифметические операторы приведены в таблице 4. Обратите внимание, что арифметические операторы полиморфны (т.е. имеют свою реализацию под каждый тип данных)6.

Таблица 4. Перечень основных арифметических операторов

Оператор	Описание	Пример	Результат

+	Складывает два значения	`2+4`
		`"aaa"+"bbb"`
		`1,2,3+4,5`
			`6`
			`"aaabbb"`
			`1,2,3,4,5`

*	Перемножает два значения	`2*4`
		`"a"*3`
		`1,2,3*2`
			`8`
			`"aaa"`
			`1,2,3,1,2,3`

-	Вычитает одно значение из другого	`5-3`	`2`
/	Делит одно значение на другое	`6/3` `7/4`	`2` `1.75`
%	Возвращает остаток при целочисленном делении одного значения на другое	`6%3` `6%4`	`0` `2`

Для управления массивом объектов могут использоваться операторы `–join` и `–split`. Первый оператор объединяет массив объектов в один объект, используя разделитель, указанные в правой части. Второй оператор делает противоположное действие.

```
PS> ("фывв ываыва выаыва выаыва" -split "ы") -join "|"
ф|вв |ва|ва в|а|ва в|а|ва
PS> ((13,23,31) -join 4) -split 3
1
42
4
1
PS>
```

Операторы `–split` и `–join` могут быть унарными (иметь только правое значение), но в этом случае они должны располагаться в первом поле команды. Например:

```
PS> -split(1,2,3)
1
2
3
PS> -join (1,2,3)
123
PS>(-join (1,2,3)) -split “2”
1
3
PS>
```

2.8. Конвейеризация команд

Конвейер в PowerShell — это последовательность команд, разделенных между собой знаком | (вертикальная черта). Каждая команда в конвейере получает объект(ы) от предыдущей команды, выполняет определенные операции над ним(и) и передает следующей команде в конвейере. С точки зрения пользователя, объекты упаковывают связанную информацию в форму, в которой информацией проще манипулировать как единым блоком и из которой при необходимости извлекаются определенные элементы.

Используя конвейризацию можно получать отдельные свойства объектов, фильтровать объекты, манипулировать ими.

Чтобы получить интерфейс объекта используется командлет `Get-Member`:

```
PS> $a | get-member
   TypeName: System.Int32
Name        MemberType Definition
----        ---------- ----------
CompareTo   Method     int CompareTo…
Equals      Method     bool Equals…
GetHashCode Method     int GetHashCode()
GetType     Method     type GetType()
GetTypeCode Method     System.TypeCode …
ToBoolean   Method     bool IConvertible.ToBoolean…
ToByte      Method     byte IConvertible.ToByte…
ToChar      Method     char IConvertible.ToChar…
ToDateTime  Method     datetime IConvertible.ToDateTime…
ToDecimal   Method     decimal IConvertible.ToDecimal…
…
PS>
```

Чтобы отфильтровать объекты можно использовать командлет `Where-Object`. Например, для получения процессов с идентификаторами меньшими 100 можно выполнить команду:

```
PS > Get-Process | where-object {$_.id -lt 1000}
Handles NPM(K)    PM(K)      WS(K)     CPU(s)     Id  SI ProcessName
------- ------    -----      -----     ------     --  -- -----------
    141     11     2004       8768       0,03    912  40 chrome
    590     20     1828       2240               536   0 csrss
…
PS>
```

Обратите внимание, что для формирования условия фильтрации командлет `where-object` использует блок кода, который может содержать любые команды и в обязательном порядке должен возвращать объект с типом System.Boolean.

Логика работы конвейера следующая. Команды начинают интерпретироваться по очереди, начиная с самой левой команды. Результат первой команды всегда будет массив, состоящий из одного или нескольких элементов. Далее этот массив передается в качестве входного элемента для следующей команды. Эта команда работает либо со всем массивом сразу, либо обрабатывает каждый элемент этого массива по отдельности и формирует результат – итоговый массив. И так далее, до тех пор, пока не закончатся этапы конвейера.

В командах, которые обрабатываются элементы массива поэлементно, для доступа к текущему обрабатываемому элементу используется служебная переменная `$_`.

Чтобы выполнить какие-либо действия над каждым объектом из полученного массива можно использовать комадлет `foreach` (или его краткий алиас `%`):

```
PS > 0..10 | foreach {"Это строка номер $_"}
Это строка номер 0
Это строка номер 1
Это строка номер 2
Это строка номер 3
Это строка номер 4
```

```
Это строка номер 5
Это строка номер 6
Это строка номер 7
Это строка номер 8
Это строка номер 9
Это строка номер 10
PS>
```

Следует обратить внимание, что командлет `foreach` выбирает по очереди объекты из входного потока, выполняет блок кода и полученный объект или массив объектов помещает в выходной поток (как бы замещая этим результатом входящий объект):

```
PS > 0..3 | % {1,2}
1
2
1
2
1
2
PS>
```

Результат выполнения очередного этапа конвейера может сохраняться в отдельной переменной для дальнейшего использования. Для этого в командлете foreach необходимо указать параметр -PipelineVariable и в аргументе задать имя переменной. Например, следующая конвейеризованная команда выдаст таблицу умножения:

```
PS > 1..10 | % -PipelineVariable LEFT {$_} | % -PipelineVariable
RIGHT {1..10} | % {"$LEFT * $RIGHT = "+($LEFT*$RIGHT)}
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 8 = 8
1 * 9 = 9
1 * 10 = 10
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
…
PS>
```

Изменить последовательность (отсортировать) объекты можно с помощью командлета `Sort-Object`:

```
PS > ("a","q", "b", 2) | sort-object
2
a
b
q
PS>
```

Объекты по умолчанию сортируются, используя стандартный оператор <. Если необходимо сортировать объекты по значению какого-либо свойства, то можно использовать параметр -Property. Если необходимо использовать обратный порядок сортировки, то указывается параметр -Descending.

```
PS > ("aa","qaaa", "ba", 2) | sort-object -Property length
ba
aa
```

```
qaaa
2
PS > ("aa","qaaa", "ba", 2) | sort-object -P length -descending
qaaa
ba
aa
2
PS>
```

Для получения псевдослучайного значение используется командлет Get-Random. В качестве параметров можно передать значения верхней и нижней границы, в пределах которых должно генерироваться целое число или массив объектов, из которых случайным образом должен быть выбран один элемент. Например, генерирование случайного набора символов из заданного алфавита можно выполнить следующим образом:

```
PS > -join (1..15 | % { Get-Random -InputObject
("asdfrewqASDFREWQ98765".toCharArray()) })
aE98eq6dAeqQSER
PS >
```

Возможна другая реализация этой задачи, с использованием механизма доступа к таблице символов и преобразования типов:

```
PS > -join (1..15 | %{ [char[]](0..127) -match '[0-9a-z]' | Get-
Random })
w0RRQ89lbItCPtI
PS >
```

4. Сценарии

Сценарий представляет собой блок кода на языке PowerShell, хранящийся во внешнем файле с расширением ps1. Анализ синтаксиса сценария производится при каждом его запуске.

Сценарии позволяют работать с PowerShell в пакетном режиме, то есть заранее создать файл с нужными командами, определить логику работы с помощью различных управляющих инструкций языка PowerShell и пользоваться этим файлом как исполняемым модулем.

Для запуска сценария необходимо указать в командной строке имя файла (и возможно путь к нему внутри файловой системы), в котором он содержится.

Выполнение сценария зависит от политики запуска внешних сценариев, установленной в текущем сеансе PowerShell. Получить значение политики можно с помощью команды `Get-ExecutionPolicy`.

5. Источники информации

Все объекты, доступные оболочке PowerShell расположены в хранилищах (дисках). Перечень хранилищ можно получить с использованием командлета `Get-PSDrive`. Среди хранилищ, созданных по умолчанию, будут присутствовать: псевдонимы команд (алиасы), файловая система (диски, в которых располагается корневая часть файловой системы), системный реестр, системное окружение, хранилище электронных сертификатов (электронные подписи и открытые ключи), локальные переменные и др.

Хранилища формируются за счет наличия в оболочке провайдеров – модулей, позволяющие формировать коллекции объектов. Для просмотра списка зарегистрированных в оболочке PowerShell провайдеров нужно воспользоваться командлетом `Get-PSProvider`.

Дополнительно к встроенным, можно создавать собственные хранилища PowerShell и устанавливать провайдеры, созданные другими разработчиками (например, для доступа к каталогам Active Directory или к почтовым ящикам Microsoft Exchange).

6. Практическое задание

1. Напишите конвейер команд, создающий 1535 файлов. Имена файлов генерируются случайным образом из следующего перечня символов: afgtdUHS45B-7634kLqO. Длина имен файлов – случайное число

из множества 5..17. Генерирование имени файла должно быть оформлено как функция, принимающая в качестве параметра длину имени.

2. Выполните команду, которая выведет список файлов, в именах которых присутствует символ -.

3. Оформите конвейер, разработанный в п. 1 в виде скрипта, добавив функцию, генерирующую содержание файлов. Содержание также генерируется из символов с кодами из диапазона от 0 до 127. Длинна файлов выбирается случайна из диапазона от 100 до 1kb.

3. Напишите скрипт, изменяющий имена файлов таким образом, чтобы они были пронумерованы по порядку, если бы имена были расположены в алфавитном порядке.