Windows PowerShell

Качаем. Ставим.

- Windows Management Framework 4.0
  - Windows PowerShell 4.0
  - Интегрированная среда сценариев Windows PowerShell
  - Веб-службы Windows PowerShell (расширение IIS OData для управления)
  - Средство удаленного управления Windows (WinRM)
  - Инструментарий управления Windows (WMI)
  - Поставщик WMI диспетчера серверов
  - Новая функция, представленная только в версии 4.0, — настройка требуемого состояния (DSC) Windows PowerShell

Командлеты

Первый тип команд PowerShell— гак называемые командлеты (cmdlet). Этот термин используется пока только внутри PowerShell. Командпет представляет собой класс .NET, порожденный от базового класса Cmdlet. Единый базовый класс Cmdlet гарантирует совместимый синтаксис всех, командлетов, а также автоматизирует анализ параметров командной строки и описание синтаксиса командлетов. выдаваемое встроенной справкой.

Чтобы просмотреть список командлетов, доступных в ходе текущего сеанса, нужно выполнить командлет **Get-Command**:

```
PS C:\Documents and Settings\User> Get-Command
```

В общем случае синтаксис командлетов имеет следующую структуру:

имя_командлета -параметр1 -параметр2 аргумент 1 аргумент2

Здесь -***параметр1*** — параметр, не имеющий значения (подобные параметры часто называют *переключателями); -**параметр2***— имя параметра, имеющего значение ***аргумент1\ аргумент2*** — параметр, не имеющий имени (или аргумент).

В качестве примера переключателя рассмотрим параметр -Recurse командлета Get-Childltem (dir).

Переключатель `-Recurse`, если он указан, распространяет действие команды не только на определенный каталог, но и на все его подкаталоги. Например, следующий командлет выведет информацию обо всех файлах, которые находятся в каталоге c:\vvindows или его подкаталогах и имеют имя, удовлетворяющее маске n*d.exe (как видите, никакого аргумента после параметра `-Recurse` не указано):

```
PS C:\Documents and Settings\User> dir -Recurse -Filter n*d.exe c:\windows
```

Просмотр структуры объектов (командлет Get-Member)

Для анализа структуры объекта, возвращаемого определенной командой, проще, всего направить этот объект по конвейеру на командлет Get-Member (псевдоним gm), например:

```
PS C:\> Get-Process | Get-Member
```

Фильтрация объектов (командлет Where-Object)

В PowerShell поддерживается возможность фильтрации объектов в конвейере, то есть удаления из конвейера объектов, не удовлетворяющих определенном) условию. Данную функциональность обеспечивает командлет where-object, позволяющий проверить каждый объект, проходящий через конвейер. и передать его дальше по конвейеру лишь в том случае, если объект удовлетворяет условиям проверки.

Условие проверки в **where-Object** задается в виде ***блока сценария*** (scriptblock) — одной или нескольких команд PowerShell, заключенных в фигурные скобки {}. Блок сценария указывается после имени командлета **where**-object. Результатом выполнения блока сценария в командлете **where-Object** должно быть значение логического типа: $True (истина, в этом случае объект проходит далее по конвейеру) или $False (ложь, в этом случае объект далее по конвейеру не передается).

Например, для вывода информации об остановленных службах в системе (объекты, возвращаемые командлетом **Get-service,** у которых свойство **status** равно **"stopped")** можно использовать следующий конвейер:

```
PS C:\> Get-Service | Where-Object ($_. Status -eq "Stopped"}
```

В блоках сценариев командлета where-Object для обращения к текущему объекту конвейера и извлечения нужных свойств этого объекта используется специальная переменная которая создается оболочкой PowerShell автоматически.

Операторы сравнения в PowerShell

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

Сценарий Windows PowerShell

Сценарий Windows PowerShell не совсем похож на командный файл для командной строки, кроме того выполнение сценария не совсем то же самое, что вводить вручную те же команды в той же последовательности.

1. Get-Service
2. Get-Process

Get-Service;Get-Process

Сценарий должен создавать один и только один тип выходных данных. Единственное исключение — когда сценарий используется как репозиторий нескольких функций. В этом случае каждая функция должна генерировать один и только один тип выходных данных.

Переменные

1. $var = 'hello'
2. $number = 1
3. $numbers = 1,2,3,4,5,6,7,8,9

Чтобы получить первый элемент, нужно использовать выражение $numbers[0], второй — $numbers[1], а последний элемент изображается так: $numbers[-1], предпоследний — $numbers[-2] и так далее.

Кавычки

Вместо знака доллара с именем переменной будет вставлено ее содержимое.

```
$name = 'Don'
$prompt = "My name is $name"
```

Также в тексте, ограниченном двойными кавычками, Windows PowerShell будет искать управляющий символ, или обратную одинарную кавычку, и действовать соответствующим образом. Вот несколько примеров:

```
$debug = "`$computer contains $computer"
$head = "Column`tColumn`tColumn"
```

В первом примере первый знак доллара $ отменяется. Обратная кавычка отменяет его значение как аксессора переменной. Если переменная $computer содержит строку «SERVER», тогда в $debug будет содержаться такая строка: «$computer contains SERVER».

Во втором примере «`t» представляет собой символ табуляции, поэтому Windows PowerShell разместит между словами Column знаки табуляции. Подробнее об управляющих символах см. веб-страницу.

Члены и переменные объектов

В Windows PowerShell все состоит из объектов. Даже простая строка, такая как «name», является объектом типа System.String. Чтобы узнать тип объекта (то есть, что представляет собой объект), а также его членов, к которым относятся его свойства и методы, достаточно передать объект в Get-Member по конвейеру:

```
$var = 'Hello'
$var | Get-Member
```

```
$svc = Get-Service
$svc[0].name
$name = $svc[1].name
$name.length
$name.ToUpper()
```

Скобки

Помимо указания на методы объектов, скобки также служат в Windows PowerShell маркерами, определяющими порядок выполнения, — как в обычных алгебраических выражениях. Выражение в скобках заменяется тем, что получается в результате вычисления этого выражения.

```
$name = (Get-Service)[0].name
```

```
Get-Service -computerName (Get-Content names.txt)
```

Область действия

По окончании работы сценария его область действия отбрасывается, а все созданное в ней исчезает. Например, создайте следующий сценарий и выполните его в окне консоли:

```
New-PSDrive -PSProviderFileSystem -Root C:\ -Name Sys
Dir SYS:
```

Язык сценариев Windows PowerShell

```
if ($those -eq $these)
{
  #commands
}

function mine {
 if ($this -eq $that){
  get-service
 }
}
```

```
Do {
  # commands
} While ($this -eq $that)

While (Test-Path $path) {
  # commands
}

$services = Get-Service
ForEach ($service in $services) {
  $service.Stop()
}
```

Функции

```
function Mine {
  Get-Service
  Get-Process
}
Mine
```

```
1.  function One {
2.    function Two {
3.  Dir
4.    }
5.    Two
6.  }
7.  One
8.  Two
```

Выполняемая строка номер восемь приведет к ошибке. В сценарии нет функции по имени Two. Функция Two спрятана в функции One. Поэтому функция Two существует только в области действия функции One и видна только из последней. Попытка вызвать Two из любого другого места приведет к ошибке.

Параметры в сценарии

Параметры определяются особым образом вначале сценария.

```
param (
  [string]$computername,
  [string]$logfile,
  [int]$attemptcount = 5
)
```

Вот несколько способов запуска сценария (предполагается, что сценарий содержится в файле Test.ps1):

```
./test -computername SERVER
./test -comp SERVER -log err.txt -attempt 2
./test SERVER err.txt 2
./test SERVER 2
./test -log err.txt -attempt 2 -comp SERVER
```

Примеры

1. Выводит список файлов из папки C:\Windows\temp, измененных за последние 30 минут:

```
Get-ChildItem C:\Windows\temp\*.* | where {$_.LastWriteTime -ge (Get-Date).AddMinutes(-30)}
```

2. Выводит список файлов из папки C:\Windows с расширением dll:

```
Get-ChildItem -Path C:\Windows | Where {$_.extension -eq ".dll"}
```

3. Выполняется 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)}
```

4. Аналогом команды copy стал командлет Copy-Item. Для перезаписи целевого файла используется параметр Force.

```
Copy-Item -Path D:\Script\script-01.ps1 -Destination E:\Backup\27-09-2011\script-01.ps1 -Force
```

5. Копирование дерева папок выполняется той же командой, но с указанием ключа Recurse.

```
Copy-Item -Path D:\Script -Recurse E:\Backup\27-09-2011
```

6. При необходимости копирования определенных объектов, например только скриптов PowerShell-а вы можете задать фильтр.

```
Copy-Item -Filter *.txt  -Path D:\Script -Recurse E:\Backup\27-09-2011
```

7. Создание файлов и папок средствами PowerShell производится с помощью командлета New-Item. Для создания папки придется указать тип элемента “Directory”, а для создания файла “File”

```
New-Item -Path ‘D:\Script\New Folder’ -ItemType "directory"
New-Item -Path ‘D:\Script\New Folder\script-02.ps1′ -ItemType "file«
```

8. Переименование элементов производится с помощью команды Rename-Item.

```
Rename-Item .\File-01.ps1 .\File-01.ps1.bak
```

9. Удаление производится с помощью Remove-Item.

```
Remove-Item d:\Backup\20-09-2011 -Recurse
```

10. Задача поиска в лог файлах определенного выражения по маске ip-адреса “192.168.100.253” с последующим копированием найденного в папку D:\TEMP:

```
$current = Get-Date
$Days = "-30"
$start = $current.AddDays($days)
Get-ChildItem D:\Logs -Filter *.log -Recurse | Where-Object {($_.LastWriteTime.Date -ge $Start.Date) -and ($_.LastWriteTime.Date -le $End.Date)} | Select-String "192.168.100.253" | Copy-Item -Destination D:\TEMP
```

11. Сценарий, когда требуется из определенной папки удалять все файлы и папки старше 30 дней. Удаляется все, что есть в папке и в подпапках т.е. удаление еще и самих папок.

```
$Path = "C:\temp"
$Days = "-30"
$CurrentDate = Get-Date
$OldDate = $CurrentDate.AddDays($Days)
Get-ChildItem $Path -Recurse | Where-Object { $_.LastWriteTime -lt $OldDate } | Remove-Item
```

Технология COM

- До появления платформы .NET технология ActiveX была ключевой в продуктах Microsoft. В настоящее время COM-объекты по-прежнему широко используются в Windows. Многие приложения как компании Microsoft (Microsoft Office. Internet Explorer. Internet Information Service (IIS) и т.д.). так и сторонних разработчиков являются серверами автоматизации, предоставляя через COM-объекты доступ к своим службам. Другими словами, сервера автоматизации позволяют управлять некоторыми аспектами соответствующего приложения внешним программам.
- Работая в PowerShell, мы будем идентифицировать COM-объекты по их программным идентификаторам (ProgID) — символьным псевдонимам, назначаемым при регистрации объектов в системе. Согласно общепринятому соглашению идентификаторы ProgID имеют следующий вид (при этом общая длина программного идентификатора не должна превышать 39 символов):

Библиотека_типов.Класс.Версия или просто

Библиотека_типов.Касс

Объекты и их свойства

Например, экземпляр COM-объекта с программным идентификатором wscript.shell создастся следующим образом:

```
PS  C:\> $Shell = New-Object -ComObject WScript.Shell
```

"Общение" с COM-объектами в PowerShell происходит с помощью соответствующих механизмов .NET Framework (создаются экземпляры .NET-класса System.__ComObject). Поэтому на командлет New-Object действуют те же ограничения, какие применяются в платформе .NET во время вызова COM-объектов.

Посмотрим, какие свойства и методы имеются у COM-объекта wscript.Shell. Для этого воспользуемся командлетом Get-Member, передав ему по конвейеру переменную $Shell, в которой сохранена ссылка па данный COM-объект:

```
PS  C:\>$Shell  I  Get-Member
```

Управление приложением Microsoft Word

```
#Создаем экземпляр сервера автоматизации
$Word = New-Object -ComObject Word.Application
#Создаем новый документ
$doc = $word.Documents.Add()
$Word.Visible = $True
#Создаем объект Selection
$sel = $Word.Selection
#Устанавнавливаем размер шрифта
$sel.Font.Size=14
#Устанавливаем полужирный шрифт
$sel.Font.Bold=$True
#Печатаем строку текста в Word
$sel.TypeText(“Привет из  PowerShell!”)
```

Управление приложением Microsoft Excel

```
#Создаем объект-приложение Microsoft Excel
$XL=New-Object -ComObject Excel.Application
#Делаем окно Microsoft Excel видимым
$XL.Visible = $True
#Открываем новую рабочую книгу
$XL.WorkBooks.Add()>$Null
#Устанавливаем нужную ширину колонок
$XL.Columns.Item(1).ColumnWidth = 40
$XL.Columns.Item(2).ColumnWidth = 40
$XL.Columns.Item(4).ColumnWidth = 40
#Печатаем в ячейках текст
$XL.Cells.Item(1,1).Value()="Фамилия"
$XL.Cells.Item(1,2).Value()="Имя"
$XL.Cells.Item(1,3).Value()="Телефон“
#Выделяем три ячейки
$XL.Range("A1:C1”).Select() > $Null
#Устанавливаем полужирный текст для
$XL.Selection.Font.Bold = $true
#Печатаем в ячейках текст
$XL.Cells.Item(2,1).Value()="Иванов"
$XL.Cells.Item(2,2).Value()=“Иван"
$XL.Cells.Item(2,3).Value()="555555"
```

1. А.Попов. Введение в Windows PowerShell
2. Don Jones Learn Windows PowerShell 3 in a Month of Lunches
3. http://technet.microsoft.com/ru-ru/magazine/hh551144.aspx
4. http://blog.wadmin.ru/2011/09/powershell-lessons-working-with-files-and-folders/