

Курс «DevOps. Системный инженер»

Занятие 12.

MS PowerShell

Преподаватель

Денис Сойка



- Что такое Powershell и для чего он используется
- Команды, конвейеризация. Понятие хранилищ.
- Функции в PowerShell
- WMI. Для чего нужно.

Windows PowerShell – более чем язык скриптов.

Windows PowerShell – это целый механизм, созданный для решения административных задач, например:

- создание учетных записей
- конфигурация сервисов
- удаление почтовых ящиков

PowerShell предоставляет возможность проектировать GUI интерфейсы на своей базе. Вы можете выполнять задачи двумя путями:

- Набирать команды в консоли
- Выбирать графические элементы, которые выполнять те же команды.

Где используется PowerShell?

03

PowerShell используется со всеми современными продуктами Microsoft, такими как:

Microsoft Exchange Server (начиная с версии 2007)

Microsoft System Center Data Protection Manager

Microsoft System Center Operations Manager

Microsoft System Center Virtual Machine Manager

Microsoft SQL Server (начиная с версии 2008)

```
Get-Mailbox | Sort Size | Select -first 100 | Move-Mailbox Server2
```

Какие команды вам уже знакомы?

04

Какие команды вы использовали в стандартной командной строке Windows или Bash для выполнения следующих задач:

- Переход в папку
- Получение списка файлов и подпапок в папке.
- Копирование файлов
- Отображение содержимого текстового файла.
- Удаление файлов.
- Перемещение файлов.
- Переименование файлов.
- Создание папок.

С PowerShell можно также запускать большинство внешних команд (программ), таких как:

- Ipconfig.exe
- Ping.exe
- Tracert.exe
- Nslookup.exe
- Pathping.exe
- Net.exe (например, Net Use)

PowerShell распознает многие из знакомых имен команд, в том числе: Cd, Dir, Ls, Cat, Type, Mkdir, Rmdir, Rm, Del, Cp, Copy, Move и так далее. Одновременно доступны наиболее общие команды управления файлами и папками среды Cmd.exe (которая использует синтаксис команд MS-DOS) и оболочки *nix.

Файловые системы Windows (как и файловые системы других компьютеров) – иерархические.

Файловые системы состоят из папок, которые содержат или файлы, или другие папки. Папки содержат подпапки, которые, опять же, содержат свои подпапки, и так далее

Файловая система - это не единственное иерархическое хранилище в Windows.

Таковыми хранилищами также являются:

- Системный реестр
- Хранилище сертификатов
- Active Directory

Одна из «вкусностей» PowerShell - единая система работы со всеми иерархическими хранилищами.

Передача выходных данных одного командлета во входные данные другого командлета называется конвейеризацией. В других оболочках также используется конвейер. Например, это стандартная команда в Cmd.exe:

```
dir | more
```

Здесь выходные данные команды Dir перенаправляются во входные данные команды More, которая создает постраничное отображение выходных данных.

Конвейер широко применяется в PowerShell. Весьма распространенным явлением здесь является строка из множество командлетов, связанных между собой конвейером. Данные переходят из одного командлета в другой, при этом они уточняются, детализируются и превращаются в требуемую информацию.

Основные командлеты для модификации вывода

Format-Table, имеет alias Ft

Format-List, имеет alias Fl

Format-Wide, имеет alias Fw

Get-Service | Format-List

Get-Process | Fw

Перенаправление ввода-вывода и форматирование

Имеется ряд Out- командлетов

Out-GridView

Out-Printer

Out-File

Out-Host

Out-Null

Get-Process -> Format-List -> [Out-Default]

Командлеты возвращают объекты

Командлеты используют объекты как входные данные

Командлеты могут принимать свойства входящих объектов в качестве параметров

В конце каждого командного конвейера находится командлет Out-Default. Он всегда находится там, даже если вы не указали его в командной строке. Его работа заключается в том, чтобы принять окончательные выходные данные из конвейера и передать их командлету Out-Host, который отвечает за вывод информации на экран.

Если вы наберете команды:

- Get-Process
- Get-Process | Out-Default
- Get-Process | Out-Host

то получите одинаковые результаты

Вызывать Out-Default нет необходимости. Можно использовать другие командлеты для перенаправления вывода.

Преобразование данных

Sort-Object

Group-Object

Measure-Object

Select-Object

Compare-Object

Импорт и экспорт данных

Import-CSV

Export-CSV

Import-CliXML

Export-CliXML

Sort-Object позволяет изменить порядок, в котором перечисляются объекты

Sort-Object может принимать входящие данные любого типа

Необходимо указать свойства, в соответствии с которыми будет сформирован список объектов

Если объекты по умолчанию отсортированы в восходящем порядке, указав параметр

–Descending, можно изменить порядок на нисходящий

Get-Service | Sort-Object Status

Get-Service | Sort-Object Status –Descending

Get-Service | Sort Status,Name

Командлет Group-Object изучает свойства заданных объектов и объединяет объекты в группы по значениям каждого свойства

На выходе Group-Object показывает, сколько объектов находится в каждой группе.

Group-Object полезен, когда свойства объектов имеют повторяющиеся значения

Get-Service | Group-Object status

Командлет Measure-Object считает число включенных объектов, а также считает составные значения числовых свойств объектов.

–average

–sum

–minimum

–maximum

Measure-Object забирает в себя входящие объекты, то есть, поступив в него, они убираются с конвейера

Measure-Object обычно последний командлет в цепочке

```
Get-Process | Sort VM –desc | Select –First 10 | Measure-Object
```

```
Get-Process | Sort VM –desc | Select –First 10 | Export-CSV top-vm.csv
```


Командлет Select-Object используется для двух целей
для выбора подмножества объектов в конвейере: -First, -Last и -Skip

```
Get-Process | Select-Object -First 10
```

Для выбора свойств объектов

```
Get-Process | Select-Object Name,ID,VM,PM
```

В комбинации

```
Get-Process | Select Name,ID -First 10
```

PowerShell может читать и записывать файлы, в которых значения разделены запятой (CSV), а также простые XML файлы

Import-CSV, Export-CSV, Import-CliXML, Export-CliXML

Get-EventLog Security –newest 20 | Export-CSV new-events.csv

Get-Process | Sort VM –desc | Select –First 10 | Export-CSV top-vm.csv

Import-CliXML procs.xml | Get-Member

Операторы сравнения

Основные	Чувствительные к регистру	Нечувствительные к регистру
-eq – равно	-ceq – равно	-ieq – равно
-ne – не равно	-cne – не равно	-ine – не равно
-le – меньше или равно	-cle – меньше или равно	-ile – меньше или равно
-ge – больше или равно	-cge – больше или равно	-ige – больше или равно
-gt – больше, чем	-cgt – больше, чем	-igt – больше, чем
-lt – меньше чем	-clt – меньше чем	-ilt – меньше чем

Операторы булевой алгебры

В сложных сравнениях используются операторы –and и –or

`4 > 10 or 10 > 4 # $True`

`4 < 10 and "Hello" ne "hello" # $False`

Обычно сравнения выполняются слева направо, однако можно группировать выражения

`(4 > 10 and (10 < 4 or 10 > 5)) and 10 <= 10`

Командлет Where-Object

Используется для выборки объектов из конвейера

Выбирает объекты, подходящие по критериям

Where-Object использует специальную переменную \$_, обозначающую текущий объект.

```
Get-Service | Where-Object { $_.Status -eq "Running" }
```

Командлет ForEach-Object:

Позволяет выполнить операции над набором объектов

Where-Object использует \$_ для обозначения текущего объекта

Использует блок скрипта { }

```
Get-Service | Where-Object { $_.Status -eq "Stopped" } | ForEach-Object { $_.Start() }
```

Позиционные параметры

Позиционные параметры не требуют указания их имен в командной строке.

Использование таких параметров основывается на их расположении (позиции) в командной строке.

Это упрощает ввод команд:

```
Stop-Process -id 53 # Исполняется корректно
```

```
Stop-Process 53 # Исполняется корректно
```

Имена позиционных параметров указываются в справке в квадратных скобках []

```
Stop-Process [-Id] <Int32[]>
```

Имена параметров, кроме позиционных, могут находиться в любой части командной строки.

Использование имен параметров делает код легче для понимания.

Привязка данных конвейера по значению

Многие параметры предназначены для того, чтобы принимать данные из конвейера. Этот процесс называется привязкой (binding).

Этот процесс использовался ранее:

```
Get-Service | Where-Object { $_.Status -eq "Running" }
```

-InputObject <psobject>

Specifies the objects to be filtered. You can a...

Required? false

Position? named

Default value

Accept pipeline input? true (ByValue)

Accept wildcard characters? False

passThru (Passthrough)

Большинство «командлетов действия» могут принимать входящие данные, но не передают объекты далее по конвейеру.

```
New-ADUser -name JohnD -samaccountname JohnDoe
```

```
# созданный пользователь будет находиться в отключенном состоянии
```

Командлеты, по умолчанию не передающие по объекты на конвейер, требуют дополнительного параметра `-passThru`

```
New-ADUser -name JohnD -samaccountname JohnDoe -passThru |  
Enable-ADAccount
```

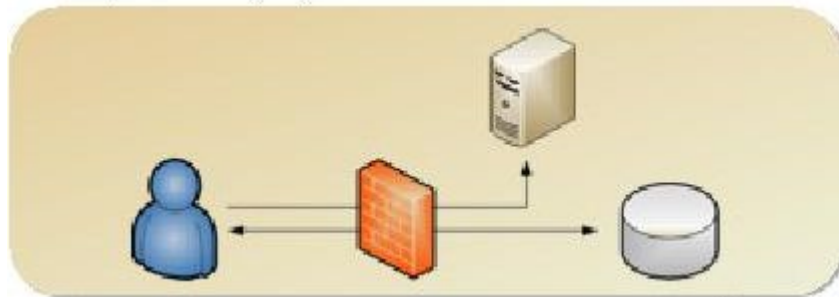
```
# созданный пользователь будет передан Enable-ADAccount  
-passThru используется многими командлетами (см. справку)
```

Windows Management Instrumentation – технология управления, являющаяся частью операционной системы Windows. Впервые она появилась в Windows NT 4.0, и обеспечивала стабильный доступ к настройкам как локального, так и удаленных компьютеров.

Использование WMI не всегда было простым: такие технологии, как VBScript требуют программного подхода к использованию WMI, и администраторам было сложно справиться с этой задачей.

PowerShell предлагает администраторам самый легкий и доступный способ работы с WMI.

Взаимодействие WMI использует протокол Remote Procedure Call или RPC. Используется распределитель конечной точки. Распределитель конечной точки открывает произвольный TCP порт для взаимодействия компьютеров. Сложно создавать статические правила фаерволов, разрешающие RPC трафик



Windows Firewall поддерживает правила для Remote Management
Эти правила позволяют динамически открывать порты для WMI RPC трафика

Windows Management Instrumentation – технология управления, являющаяся частью операционной системы Windows. Впервые она появилась в Windows NT 4.0, и обеспечивала стабильный доступ к настройкам как локального, так и удаленных компьютеров.

Использование WMI не всегда было простым: такие технологии, как VBScript требуют программного подхода к использованию WMI, и администраторам было сложно справиться с этой задачей.

PowerShell предлагает администраторам самый легкий и доступный способ работы с WMI.

```
Get-WmiObject Win32_Service
```

```
Get-WmiObject Win32_Service | Get-Member
```

```
Get-WmiObject Win32_Service -computerName "COMP1","COMP2"
```

```
Get-WmiObject Win32_Service -computerName (Get-Content names.txt)
```

Функция в PowerShell объявляется с помощью ключевого слова `function`, за которым следует имя функции, а затем — открывающая и закрывающая фигурные скобки. В этих скобках содержится код, который будет выполнять функция.

```
function Get-Version {  
    $PSVersionTable.PSVersion  
}
```

Показан простой пример функции, возвращающей версию PowerShell.

Существует большая вероятность возникновения конфликта имен функций, названных примерно как `Get-Version`, и команд по умолчанию в PowerShell или команд, которые могут написать другие пользователи. Именно поэтому в целях предотвращения конфликтов имен рекомендуется добавлять префикс к части функции, выраженной существительным. В следующем примере используется префикс `PS`.

```
function Get-PSVersion {  
    $PSVersionTable.PSVersion  
}
```

`Get-PSVersion`

Функции, загруженные в память, можно увидеть на `PSDrive Function`:

```
Get-ChildItem -Path Function:\Get-*Version
```

Чтобы удалить эти функции из текущего сеанса, необходимо удалить их из PSDrive Function или закрыть и повторно открыть PowerShell.

```
Get-ChildItem -Path Function:\Get-*Version | Remove-Item
```

Убедитесь, что функции действительно удалены.

```
Get-ChildItem -Path Function:\Get-*Version
```



```
function Get-MrParameterCount {  
    param (  
        [string[]]$ParameterName  
    )  
  
    foreach ($Parameter in $ParameterName) {  
        $Results = Get-Command -ParameterName $Parameter -ErrorAction  
        SilentlyContinue  
  
        [pscustomobject]@{  
            ParameterName = $Parameter  
            NumberOfCmdlets = $Results.Count  
        }  
    }  
}
```

```
Get-MrParameterCount -ParameterName ComputerName, Computer, ServerName,  
Host, Machine
```

Имена параметров рекомендуется указывать в том же регистре, что и имена командлетов по умолчанию. Используйте `ComputerName`, а не `computername`. В этом случае функции выглядят и работают как командлеты по умолчанию. Это очень удобно для пользователей, уже знакомых с PowerShell.

Инструкция `param` позволяет определить один или несколько параметров. Определения параметров разделяются запятой (,).

Расширенная функция

```
function Test-MrCmdletBinding {  
  
    [CmdletBinding()] #<-- This turns a regular function into an advanced function  
    param (  
        $ComputerName  
    )  
  
    Write-Output $ComputerName  
  
}  
  
{  
    [CmdletBinding(ConfirmImpact=<String>,  
        DefaultParameterSetName=<String>,  
        HelpURI=<URI>,  
        SupportsPaging=<Boolean>,  
        SupportsShouldProcess=<Boolean>,  
        PositionalBinding=<Boolean>)]  
  
    Param ($Parameter1)  
    Begin{}  
    Process{}  
    End{}  
}
```

[PowerShell Documentation - PowerShell | Microsoft Docs](#) - полная документация Powershell

[Введение в Windows PowerShell 5 - YouTube](#) - бесплатный курс Powershell 5 на русском

**Спасибо
за внимание!**

