Основы PowerShell

Artem Beresnev

t.me/ITSMDao

История PowerShell

- Что было до
  - Командная строка Windows (cmd.exe)
    - внутренние команды (нет отдельных исполняемых файлов),
    - внешние команды — реализованы в отдельных исполняемых файлах.
- Windows Script Host (WSH)
  - VBScript
  - Jscript
  - WMI

WMI

- Windows Management Instrumentation (WMI) — это одна из базовых технологий для централизованного управления и слежения за работой различных частей компьютерной инфраструктуры под управлением платформы Windows.

- WMI это объектный интерфейс :
  - позволяющий получать информацию о системе
  - позволяющий менять параметры системы
  - доступный через утилиты, .NET Framework и .NET
- В WMI доступны четыре подпространства:
  - CIMv2, Default, Security и WMI.
- Утилиты:
- wmimgmt.msc, winmgmt.exe, wbemtest.exe, wmic.exe
- mofcomp.exe

История PowerShell

- Что было не так
  - Командная строка Windows (cmd.exe)
    - Бедный и плохо читаемый встроенный язык
    - Работа с объектами системы через утилиты, у которых могло быть странное поведение
  - Windows Script Host (WSH)
    - VBScript и Jscript
    - Сложный интерфейс доступа к объектам системы

Что такое PowerShell?

- язык сценариев,
- язык сценариев, с объектной моделью,
- язык сценариев, разработанный на основе CLR (Common Language Runtime — общеязыковая исполняющая среда) — исполняющей среда для байт-кода CIL, в которой компилируются программы, написанные на .NET-совместимых языках программирования C#, Managed C++, Visual Basic .NET, F# и др.
- командная оболочка, которая разработана для администрирования и конфигурирования операционных систем Windows,
- продукт с открытым исходным кодом под лицензией MIT (с 2016 .NET), https://github.com/PowerShell/PowerShell
- средство кроссплатформенного взаимодействия (Windows, Linux, macOS).

Версии PowerShell

.NET Framework

PowerShell 1.0 - 2006 (XP SP2, Server 2003 SP1, Vista, Server 2008)

PowerShell 2.0 - 2008 (7, XP SP2, Server 2003 SP1, Vista, Server 2008, Server 2008 R2)

Windows PowerShell ISE v2.0, +240 командлетов, PowerShell remoting, Background jobs, Modules, Network file transfer и д.р.

PowerShell 3.0 (8, Server 2012, 7 +SP1, Server 2008 SP1, Server 2008 R2 +SP1)
Windows Management Framework 3.0 (WMF3), WinRM, Scheduled jobs и новые командлеты.

PowerShell 4.0 (8.1, Server 2012 R2, 7 +SP1, 2008 R2 +SP1, 2012)
Desired State Configuration (DMTF), Enhanced debugging, Where and ForEach.

PowerShell 5.0 – 2016
WMF 5.0 RTM, Debugging for PowerShell Runspaces in remote processes, Debugging for PowerShell Background Jobs

PowerShell 5.1 – 2017 – в двух версиях "Desktop" (.NET Framework stack) and "Core" (.NET Core stack).

Версии PowerShell

.NET (.NET Core)

PowerShell Core 6.0 2018 Windows, macOS and Linux.

PowerShell Core 6.1 2019 Windows, macOS, and Linux,

PowerShell 7 должен заменить 5.1 и 6.x.

Как получить PowerShell

- Встроенные версии
- WMF
- github

Стандартные расширения

- Предусмотрены следующие расширения для файлов PowerShell:
- .ps1 - файлы скриптов,
- .psd1 - файлы данных скриптов,
- .psm1 - файлы модулей скриптов,
- .ps1xml - файлы конфигурации.

Основное понятие - командлет

- **Командлет** (*cmdlet*) – это команда Windows PowerShell, с помощью которой можно осуществлять взаимодействие с объектами операционной системы с целью их управления.

«*Глагол-Существительное*»

Add – добавление данных;
Clear – очистить;
Enable – включить;
Disable – выключить;
New – создать;
Remove – удалить;

Set – задать;
Start — запустить;
Stop – остановить;
Export – экспортировать;
Import – импортировать,
И др..

Получение справки

- Get-Command
- Get-Help
- Get-Member

- Save-Help
- Update-Help

Основное понятие - командлет

- бывают как системными, так и пользовательскими,
- выводят результаты в виде объектов или их коллекций;
- могут как получать данные для обработки, так и передавать данные по конвейеру;
- Имена не чувствительны к регистру (можно написать и get-process, и Get-Process, и GeT-pRoCeSs);
- После командлетов не обязательно ставить ";", за исключением, когда выполняется несколько командлетов в одну строку (Get-Process; Get-Services).

Конвейер в PowerShell

Конвейер – это передача результата работы командлета через вертикальную черту (|) другому командлету.

В PowerShell командлеты работают с объектами и возвращают объекты, соответственно по конвейеру передаются также объекты.

Политика выполнения скриптов

- Restricted – блокируется выполнение любых сценариев (значение по умолчанию);
- AllSigned – разрешено выполнение сценариев, которые имеют цифровую подпись;
- RemoteSigned – разрешено выполнение локальных сценариев, все скачанные сценарии должны иметь цифровую подпись;
- Unrestricted — разрешено выполнение любых сценариев (не рекомендуется, так как небезопасно!).

Get-ExecutionPolicy
Set-ExecutionPolicy

Удаленное управление на PowerShell

- Служба WinRM для HTTP\HTTPS трафика на порту 5985

Enable-PSRemoting

- Выполнение команды на удаленном хосте:
  - С помощью параметра –ComputerName (есть у многих команд). Работает только для одной команды;
  - Сессия 1-to-1
  - Сессии 1-to-many

Удаленное управление на PowerShell

Сессия 1-to-1

- Командлет Enter-PSSession (интерактивный сеанс)

Enter-PSSession -ComputerName SRV-01

- нельзя сделать последующее подключение
- нельзя использовать команды имеющие графический интерфейс
- нельзя запускать команды имеющие свой собственый шел, например nslookup, netsh
- вы можете запускать скрипты если политика запуска на удаленной машине позволяет их запускать
- нельзя взаимодействовать с пользователем на удаленной машине

Удаленное управление на PowerShell

Сессии 1-to-many

- Командлет Invoke-Command

Invoke-Command -Command { dir } -ComputerName SRV-01, SRV-02

- Через -ScriptBlock можно передать набор команд
- Через –FilePath – путь к скрипту на вашем компьютере,
- При выполнении скрипта по –FilePath, область выполнения – удалённые машины, следим за переменными.

Удаленное управление на PowerShell

Можно сохранять настройки сессий и использовать их в пределах сеанса (-Session)

```
$sessions = New-PSSession
    -ComputerName s1, s2, c1
    -Port 5555
    -Credential DOMAIN\Admin
```

Фоновое исполнение заданий

В PowerShell есть возможность фонового исполнения заданий

- Start-Job – запустить фоновую задачу;
- Stop-Job – остановить фоновую задачу
- Get-Job – посмотреть список фоновых задач;
- Receive-Job – посмотреть результат выполнения фоновой задачи;
- Remove-Job – удалить фоновую задачу;
- Wait-Job – перевести фоновую задачу на передний план, для того чтобы дожидаться ее окончания.

Start-Job {Get-Service}

Комментарии

Комментарии в PowerShell бывают:

- # строчные
- <#

  и

  блочные

  #>

Переменные в Windows PowerShell

- $имя_переменной
- Get-Varible
- Накопитель Variable:
- $_ - текущий элемент в конвейере
- можно объявлять с указанием типа и без
- можно сразу инициализировать
- Переменные могут менять свой тип, но только в том случае если не указан тип при объявлении.
- $Global: переменная = значение – можно сделать глобальную переменную.

```
[int]$MyIntVar = 5
```

[int] — целое число, 32 бита;
[single] — число с плавающей запятой одинарной точности;
[double] — число с плавающей запятой двойной точности;

[char] — один символ;
[Boolean] — значения «Истина» или «Ложь»;
[datetime] — дата или время;
[string] — строка символов, т.е.

Массивы и хэштаблицы

- Массивы (нумерация с 0)

```
[int32[]]$Array = 1, 3, 5, 7, 9
$Array[2]
```

```
$Array = 1, 3, "Пример", 7, 9
$Array[2]
```

- Хеш-таблицы строятся по принципу: **@{ ключ = «значение» }**

```
$ht = @{w1="I"; w2="LOVE"; w3="POWERSHELL"}
$ht
```

```
$ht.Add("w4","REALLY")
$ht
```

Условные конструкции в PowerShell

- IF
- IF…ELSE
- IF…ELSEIF…ELSE
- SWITCH

Операторы сравнения PowerShell

- -eq – равно (знак =);
- -ne – не равно (эквивалентно знакам <> или !=);
- -gt – больше (знак >);
- -lt – меньше (знак <);
- -ge – больше или равно (эквивалентно знакам >=);
- -le – меньше или равно (эквивалентно знакам <=).

Условные конструкции в PowerShell

```
[int]$TestVar = 150
 If ($TestVar -eq 100){
   Write-Host "Переменная TestVar = 100"
 }
 ELSEIF ($TestVar -gt 100){
   Write-Host "Переменная TestVar > 100"
  }
 ELSE {
    Write-Host "Переменная TestVar < 100"
 }
```

Условные конструкции в PowerShell

```
[int]$TestVar = 2
 SWITCH ($TestVar)
 {
   0 {Write-Host "Переменная TestVar = 0"}
   1 {Write-Host "Переменная TestVar = 1"}
   2 {Write-Host "Переменная TestVar = 2"}
default {Write-Host "Неопределенное значение"}
 }
```

Условные конструкции в PowerShell

Дополнительные операторы сравнения PowerShell

- -like (символ подстановки) “PowerShell " -like “PowerS*" #true
- -notlike (не символ подстановки) “PowerShell " -nolike “PowerS*" #false
- -contains Содержит ли значение слева значение справа

1, 2, 3, 4, 5 -contains 3 #$true

- -notcontains Если значение слева не содержит значение справа, получим истину

1, 2, 3, 4, 5 -notcontains 3 #$false

- -match Использование регулярных выражений

$str = “http://ifmo.ru"; $str -match "^http://(\S+)+(.ru)$" #$true

- -notmatch Использование регулярных выражений

$str = "http://ifmo.ru"; $str -notmatch "^http://(\S+)+(.com)$" #true

Логические операторы

-and Логическое **и**

-or Логическое **или**

-not Логическое **не**

Циклы

- WHILE
- DO WHILE
- DO UNTIL
- FOR
- FOREACH

```
[int]$TestVar = 1
  DO {
     Write-Host $TestVar
     $TestVar = $TestVar + 1
  }
  UNTIL ($TestVar -gt 10)
```

```
$PSService = Get-Service
  FOREACH ($Service In $PSService){
     $Service.Name + " - " + $Service.Status
}
```

Обработка ошибок в Windows PowerShell

Механизм **Try…Catch**.

- Потенциально опасный код в блок Try
- В блок Catch – код, выполняемый при возникновении ошибки.

```
Try {
    [int]$Number = Read-Host "Введите число"
    10 / $Number
  } catch {
   Write-Warning "Некорректное число"
  }
```

Работа с вводом-выводом

- Write-Host – вывод на экран
- Read-Host – чтение с консоли
- Out-File - Запись в файл (аналогично >)
- Export-Csv - Экспорт данных в .csv файл
- ConvertTo-Html - Запись в файл HTML

Функции

- function Get-DayLog ($param1,$param2) {

  Get-EventLog -LogName Application -Newest $param1 ….

}

- Get-DayLog -param1 50 -param2 -14
- function Get-DayLog ($param1=50,$param2) {….}

- return $InternalVariable

Расширение функциональности

- Install-Module
- Install-Script