POWERSHELL

Кое-что о технологии

Назначение PowerShell

- 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 выполнят те же
    команды.

Где используется PowerShell?

- 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?

- 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.

Внешние команды

- 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.

Иерархические хранилища

- 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. Такими хранилищами также являются:
  - Системный реестр
  - Хранилище сертификатов
  - Active Directory
- Одна из «вкусностей» PowerShell - единая система работы со всеми иерархическими хранилищами.

Единый набор команд для различных хранилищ

- Благодаря функции PowerShell, называемой провайдер PSDrive (или просто провайдер) обеспечивается возможность использования единого набора команд для навигации по разнообразным хранилищам,.
- Провайдер представляет собой разновидность адаптера, который устанавливает соединение с системой хранения и позволяет использовать ее PowerShell также, как дисковый накопитель.
- PowerShell поставляется с набором провайдеров:
  - Файловая система
  - IIS
  - Реестр
  - SQL Server
  - Переменные окружения
  - Active Directory
  - Хранилище сертификатов

Управление хранилищами PSDrive

- **Get-PSDrive** - выводит все доступные хранилища. По умолчанию, этот список включает все доступные накопители и логические диски
- **New-PSDrive** - создает новое хранилище. Требуется указать имя хранилища (без двоеточия на конце), имя провайдера и стартовую точку или путь. Тип такой стартовой точки или пути зависит от типа используемого провайдера
- **Remove-PSDrive** - уничтожает хранилище. Возможно удалить хранилища, присутствующие по умолчанию, например HKCU: или ENV:, однако они будут воссозданы при запуске нового экземпляра оболочки.

Псевдонимы

- **Get-Alias** - отображение список всех псевдонимов. Также можно использовать Dir Alias: чтобы увидеть все содержимое папки ALIAS:
- **New-Alias** - создание нового псевдонима. Здесь обязательно указать имя нового псевдонима, а также название команды, для которой он предназначен.
- **Del или** Rm - удаление псевдонима из ALIAS: drive.
- **Import-Alias** и **Export-Alias** - импорт и экспорт псевдонимов в файл и из файла.

HELP

Help *commandName*

- С помощью некоторых параметров команды Help можно получить еще более подробную информацию:
  - **-detailed** - показ более детального описания
  - **-examples** - показ примеров использования
  - **-full** - показ полной информации, включая детальное описание, описание каждого параметра и примеры использования
  - **-online** - открытие браузера и показ описания командлетов на сайте Microsoft. На сайте может содержаться обновленная или расширенная информация, которая еще не была выпущена в служебном пакете.

Расширения оболочки

- Командлеты, присутствующие по умолчанию в оболочке, не единственны доступные для нас командлеты.
  Microsoft, так же, как и сторонние разработчики программного обеспечения, могут создавать дополнительные командлеты и провайдеры PSDrive, и предоставлять их в виде оснасток или модулей. Управление оснастками осуществляется с помощью набора командлетов, название которых включает существительное PSSnapin:
  - Get-Module
  - Get-PSSnapin
  - Add-PSSnapin
  - Remove-PSSnapin
  - Import-Module
  - Remove-Module

  Get-PSSnapin -registered
  Get-Module -list

Конвейеризация (piping)

- 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

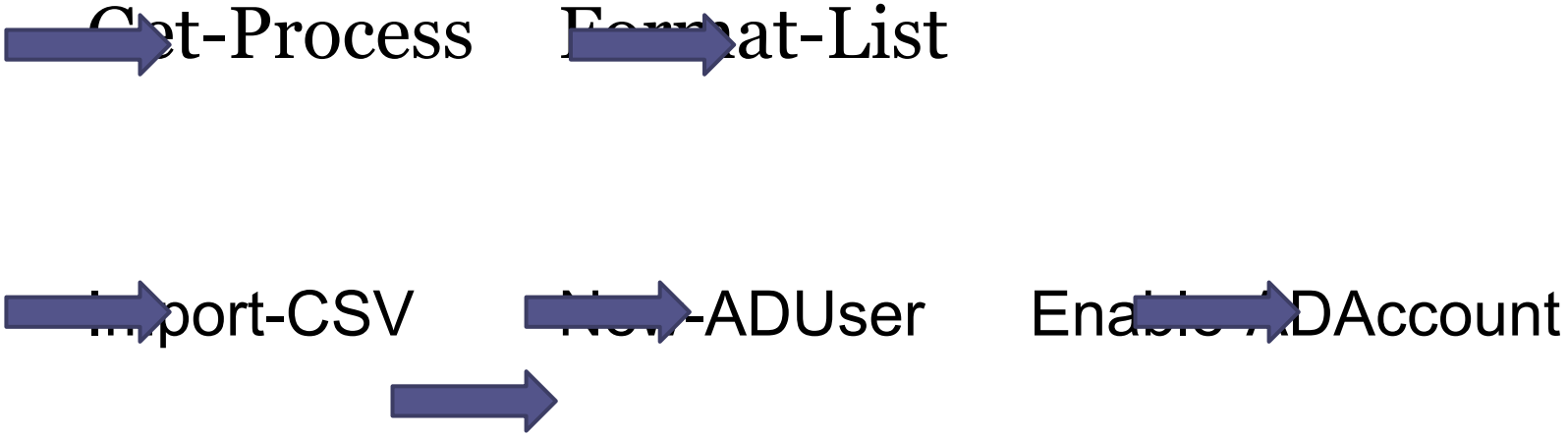
- Здесь выходные данные команды **Dir** перенаправляются во входные данные команды **More**, которая создает постраничное отображение выходных данных.
- Конвейер широко применяется в PowerShell. Весьма распространенным явлением здесь является строка из множество командлетов, связанных между собой конвейером. Данные переходят из одного командлета в другой, при этом они уточняются, детализируются и превращаются в требуюмую информацию.

Вывод командлетов

- 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
```

Конвейер


Get-Process
Format-List
Import-CSV
New-ADUser
Enable-ADAccount

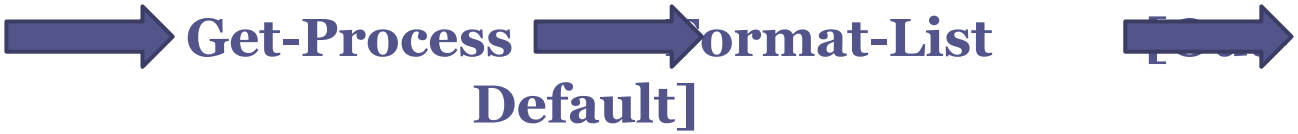
Терминология

- командлет Get-Process помещает элементы процесса в конвейер. Иными словами, командлет Get-Process помещает объекты процесса в конвейер.
- Эти объекты имеют атрибуты. Для процесса атрибутами могут быть имя, ID, объем занимаемой памяти и.т.д. То есть, объект процесса обладает свойством имени, свойством ID, и.т.д

Pid	**Name**	**Image**
092	Notepad	notepad.exe
098	Windows Paint	mspaint.exe
112	Calculator	calc.exe
164	PowerShell	powershell.exe

Перенаправление ввода-вывода и форматирование

- Имеется ряд **Out-** командлетов
  - **Out-GridView**
  - **Out-Printer**
  - **Out-File**
  - **Out-Host**
  - **Out-Null**



Свойства объектов

- Командлеты возвращают объекты
- Командлеты используют объекты как входные данные
- Командлеты могут принимать свойства входящих объектов в качестве параметров

Get-Service –computerName HOST01

- Параметры можно получить из свойства, переданного другим командлетом

Get-ComputerInventory ⇒ Get-Service

Правило 1 – определен ли формат?

- После того, как оболочка определила имя типа того элемента, который требуется отобразить, первым делом она проверяет, определен ли формат просмотра для данного типа
  - Форматы просмотра указаны в специальных конфигурационных XML-файлах
  - Файлы, название которых имеет расширение .format.ps1xml, содержат определенный формат просмотра.
  - Для нахождения этих файлов используется команда

```
Dir $pshome
```

Правило 2 – какие свойства должны отображаться?

- Если формат не определен
  - Чтобы принять решение, оболочка проверяет, зарегистрирован ли тип расширения DefaultDisplayPropertySet для того имени, которое необходимо отобразить.
  - Типы расширений хранятся в XML-файлах, так же, как и форматы просмотра.
  - Находятся там же

    ```
    Dir $pshome
    ```
- Не модифицируйте файлы форматов: они подписаны. Создавайте свои.

Правило 3 – Table или List?

- Сколько свойств элемента необходимо показать – либо те, что определены в DefaultDisplayPropertySet, либо все.
- Если оболочке требуется отобразить не более четырех свойств, используется таблица. Если пять и более – используется список. Это правило гарантирует, что таблица поместится в стандартное окно консоли стандартного размера.
- После того, как оболочка определила, какую форму (таблицу или список) следует использовать, она начинает создавать эту форму. Для этого происходит внутреннее обращение к командлету Format-List или Format-Table, куда и передаются элементы, которые необходимо отформатировать и показать.

Out-Default

- 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:

```
1. Get-Process
2. Get-Process | Out-Default
3. Get-Process | Out-Host
```

  то получите одинаковые результаты
- Вызывать Out-Default нет необходимости. Можно использовать другие командлеты для перенаправления вывода.

Дополнительные данные

- Для того чтобы добавить пользовательские свойства к объекту, используется команда Select-Object. Например, чтобы добавить атрибут ComputerName к элементу Computer, у которого уже есть атрибут Name, можно запустить команду:

```
Get-ADComputer -Filter * | Select *,@{Label='ComputerName';Expression={$_.Name}
```

- Если нужно добавить свои собственные колонки в таблицу, то вместо того, чтобы придавать новые свойства объекту, можно запустить команду:

```
Get-ADComputer -Filter * | Ft DnsHostName,Enabled,@{Label='ComputerName';Expression={$_.Name}}
```

Создание HTML-файлов

- Может потребоваться просмотреть данные через браузер
- Командлет ConvertTo-HTML
  - Преобразует данные в таблицу HTML
  - Данные выводятся не в файл, а в стандартный вывод.
  - Данные могут быть сохранены в файл посредством командлета Out-File.

```
1.  Get-EventLog Security -newest 20 |
    ConvertTo-HTML |  Out-File events.htm
```

- Параметры ConvertTo-HTML позволяют изменить заголовки и подгрузить таблицы CSS.

Основные командлеты PowerShell

- Пребразование данных
  - `Sort-Object`
  - `Group-Object`
  - `Measure-Object`
  - `Select-Object`
  - `Compare-Object`
- Импорт и экспорт данных
  - `Import-CSV`
  - `Export-CSV`
  - `Import-CliXML`
  - `Export-CliXML`

Сортировка объектов

- `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

```
1. Get-Service | Sort-Object Status
2. Get-Service | Sort-Object Status
   -Descending
3. Get-Service | Sort Status,Name
```

Группировка объектов

- 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
```

Измерения…

- 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

```
1. Get-EventLog Security -newest 20 | Export-CSV
   new-events.csv
2. Get-Process | Sort VM -desc | Select -First 10
   | Export-CSV top-vm.csv
3. Import-CliXML procs.xml | Get-Member
```

Выбор объектов и свойств

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

     ```
     Get-Process | Select-Object –First 10
     ```
  2. Для выбора свойств объектов

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

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

CSV и XML

- 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`

```
1. Get-EventLog Security -newest 20 | Export-CSV
   new-events.csv
2. Get-Process | Sort VM -desc | Select -First 10
   | Export-CSV top-vm.csv
3. 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

```
1.  4 -gt 10 -or 10 -gt 4 # $True
2.  4 -lt 10 -and “Hello” -ne “hello” # $False
```

- Обычно сравнения выполняются слева направо, однако можно группировать выражения

```
1.  (4 -gt 10 -and (10 -lt 4 -or 10 -gt 5)) -and 10 -le 10
```

Фильтрация конвейеров

- 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" }
```

Перечисление объектов

- 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 позиционых 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
```

Привязка данных конвейера по имени свойства

- В рамках данной техники оболочка ищет имя параметра, после чего проверяют, обладают ли входящие объекты соответствующими свойствами. Если да – соответствующее свойство привязывается к параметру. Обратите внимание, что данный тип привязки встречается только тогда, когда оболочка не смогла привязать входящие данные по значению. Например, изучите справочную информацию по командлету `Get-Service`.

```
-ComputerName <string[]>
Gets the services running on the specified computers…

Required? false
Position? named
Default value Localhost
Accept pipeline input? true (ByPropertyName)
Accept wildcard characters? False
```

Переименование свойств

- Когда требуется связывать командлеты, использующие разные названия для одинаковых свойств.
  - Просто по значению
  - По имени свойства –требуется переименования данного свойства
- Переименование с использованием Select-Object
- ```
  Select-Object @{Label="Newname";Expression={$_.Oldname}}
  ```

1. ```
   Get-ADComputer -Filter * | Select *, @{Label="ComputerName";Expression={$_.Name}}
   ```
2. ```
   Get-ADComputer -Filter * | Select *, @{Label="ComputerName";Expression={$_.Name}} | Get-Service
   ```
   ```

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
```

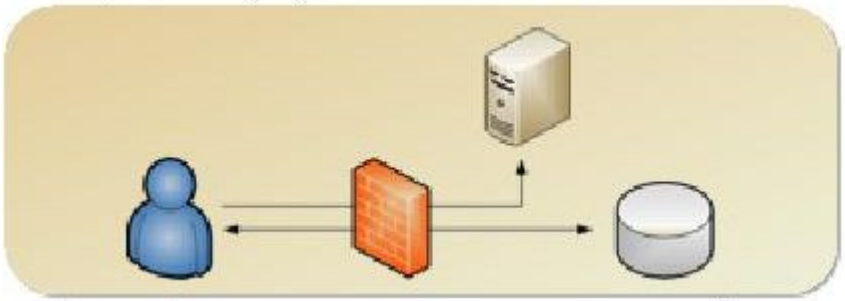
- –passThru используется многими командлетами (см. справку)

WMI

- 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

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



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

Структура WMI

- Root\Cimv2
- Root\MicrosoftDNS
- Root\MicrosoftActiveDirectory
- Root\SecurityCenter
  - Win32_Account
  - Win32_BIOS
  - Win32_Desktop
  - Win32_Fan
  - Win32_Group
  - Win32_Keyboard
  - Win32_LogicalDisk
  - Win32_NetworkAdapterConfiguration
  - Win32_NTDomain
  - Win32_Product
  - Win32_Service

Поиск нужных классов

- Командлет ***Get-WmiObject***

1. `Get-WmiObject -namespace Root\Cimv2 -List`
2. `Get-WmiObject -namespace Root\Cimv2 -List – computername COMP1`
3. `Get-WmiObject -namespace Root\Cimv2 -list – computername COMP1 -credential DOMAIN\Administrator`
4. `Get-WmiObject -namespace Root -class “__namespace” | ft name`

Использование WMI

1. `Get-WmiObject Win32_Service`
2. `Get-WmiObject Win32_Service | Get-Member`
3. `Get-WmiObject Win32_Service -computerName “COMP1","COMP2“`
4. `Get-WmiObject Win32_Service -computerName (Get-Content names.txt)`

Хитрости и уловки

- **Вычисления**
  1. ```
     gwmi win32_logicaldisk | Select deviceid,drivetype, @{Label='freespace(gb)';Expression={$_.freespace/1GB}}
     ```
  2. ```
     Gwmi win32_operatingsystem | Select caption, @{Label='PhysMemory'; Expression={(gwmi win32_computersystem).totalphysicalmemory}}
     ```
- **Фильтрация**
  1. ```
     Gwmi Win32_Service | Where { $_.Name -eq 'BITS' }
     ```
  2. ```
     Gwmi Win32_Service -Filter "Name = 'BITS'"
     ```

ForEach-Object

- WMI не является частью PowerShell, поэтому оболочка содержит ограниченное количество командлетов для работы с WMI-объектами. В большинстве случаев основные командлеты оболочки могут манипулировать WMI-объектами так, как это требуется для выполнения ваших задач. Но иногда, приходится отправлять WMI-объекты командлету `ForEach-Object`, чтобы выполнить определенные действия над каждым из них по очереди:

```
Gwmi Win32_Process | ForEach-Object { Something }
```