Scripting Basics - PowerShell

Tibor Soós – soostibor (at) citromail.hu
PowerShell MVP

Agenda

Theory

- What is PowerShell?
- Why to know?
- Elements
- Cmdlets
- Objects
- Output
- Pipeline
- Syntax
- Tools

Practice

- Running external commands
- Hello World!
- Strings
- Files
- Textfiles
- Services
- Functions
- Scripts

What is PowerShell?

- Console Environment
  - Interface for controlling Windows (and other sw) with typed in commands
- Automation tool (script)
  - Command structures supporting operations in batches with easy, intuitive syntax
- Programming language
  - With flow control, variables, error handling, operators
- Interface to .NET framework for system administrators
  - No need for deep developer background
- Framework for building graphical system administrator tools

Why to know PowerShell?

- Fast way of solving various tasks, for example:
  - Organizing, moving, cleaning up files and folders
  - Testing and checking out services
  - Intelligent user migration
  - Collecting information, creating reports and statistics
  - Processing text files and logs
- Both on the local machine or remotely
- Don’t have to be developer
- Most Microsoft server software can be managed by PS
- Windows „8” Server by default will install as ‚Core’, primary tool is PowerShell

It is also fun!

THEORY

Elements of PS as a Programming Language

- Keywords:
  - `If, Process, Begin, End, Function, While, For, Until, Param`, …
  - `Switch, ForEach, Trap, Throw, Break, Return`
- Comparison operators:
  - `-and, -or, -eq, -ne, -lt, -band, -not`, …
- Mathematical operators:
  - `+, -, *, /, %`, …
- Both in scripts and interactive mode:
  - 5*6
  - (512`mb` + 2`gb`)/1`gb`
- Help
  - `Get-Help about_*`

Commands: cmdlets

- **`Verbs-noun`**
  - Examples: **`get-help, write-host`**
  - Well…: **`Where-Object, ForEach-Object`**
- Consistent use of parameters
  - **`verb-noun -parametername parametervalue`**
  - FE.: **`Write-Host -Object Sg -ForegroundColor green`**
- Can have multiple syntaxes
- **`Get-Command`**
- Help
  - **`Get-Help cmdletname`**
  - **`Get-Help cmdletname -full`**
  - **`Get-Help cmdletname -examples`**
  - TAB extension for cmdlets and parameters
- Output, - if exists - is an **object**!

Objects

- Stored in memory
  - Can have properties and methods
  - Method is a function which operates on the object
  - Properties can be read and sometimes written
- Based on the .NET framework
  - `Object.GetType().fullname`
  - PowerShell may extend the capabilities of the original .NET objects
- Some functions are not implemented as cmdlets, but exist as methods of objects
  - No: ~~Len("Text")~~, but `"text".length`
  - No: ~~Mid$("Text",2,2)~~, but `"text".Substring(2,2)`

Object Types

- Based on .NET Framework 2.0
- Basic types:
  - [int], [string]
- Arrays:
  - `$a = 1,2,"text"`
  - `$a = @(1,2,"text")`
  - `$a[0]` → `1`
  - `$empty = @()`
- DateTime:

```
> [1] PS C:\> $today = [datetime] "2012.02.13"
> [2] PS C:\> $today
> 2012. február 13. 0:00:00
```

- HashTable:

```
> [3] PS C:\> $h = @{label = "Value"; key = 12}
> [4] PS C:\> $h
> Name                           Value
> ----                           -----
> label                          Value
> key                            12
```

  - `$h.key` → `12`
- Smart array:
  - $smart = new-object –typename collections.arraylist

Cmdlets for Manipulating Objects

- **Get-Member**
- **Select-Object**
  - Selecting a subset of objects
  - Selecting a subset of properties
- **Group-Object**
- **Sort-Object**
- **Compare-Object**
  - Should be called: Compare-Collection
- **New-Object**
- **Add-Member**
- **Measure-Object**

Output

- Most cmdlets have output
  - `Get-Process` → process objects
  - `Get-ChildItem` → file and folder objects
  - `New-Item` → the newly created objects
- No output by default, but can be requested
  - `Add-Member`: extends the object with a new member
    -passthru switch outputs the extended object
  - `Rename-Item`: renames an object
    -passthru switch outputs the renamed object
- Can be suppressed
  - …`> $null`, `[void] (…)`, … `| Out-Null`
- No output
  - `Write-Host`
- In these cases output must not be used:
  - `Format-Table`, `Format-List`, `Format-`…

Pipeline (more like an assembly line)

- Form of a general sysadmin expression
  - Get-Sg –filter Value | Where-Object {$_ -gt 123} |
    Do-Action –parameter Value
- Pipeline controller cmdlets
  - `Where-Object` (alias: ?)
    - `Get-Process | Where-Object {$_.product -notmatch "Microsoft"}`
  - `ForEach-Object` (alias: %)
    - `Get-ChildItem c:\x | Foreach-Object {$_.length -gt 1mb}`
- Do not overuse them
  - Briefness vs. speed
  - Memory consumption

PowerShell Drives

- Similar data stores
  - Hierarchical
  - Containers and leaves
  - Some have ACLs
- Let's use the same cmdlets to manage them
  - Noun: Item, ItemProperty
  - Specialties are handled by dynamic parameters
- What we currently have
  - Filesystem
  - Registry
  - Certificate store
  - Active Directory
  - IIS, SQL, …

How to Enter Expressions

- PowerShell is case insensitive
  - But there are other technologies used within PS which are case sensitive: regex
- „`=`” is the assignment operator, not equality (`-eq`)
- Parameter name can be omitted if positional
  - See `get-help -full` or `-parameter *`
- Minimal number of characters can be entered as parameter name
  - I recommend the use of TAB completion
- Use of parenthesis, brackets, etc.
  - `(), $(), @(), @{}, {}, []`

Variables

- Loose typed by default
- Strict typed
  - `[int] $x = 1`
  - `$x = "Text"` → Error
- Fast definition
  - `$a = 1`
- Thorough definition
  - `New-Variable -Name b -Value 1 -Description "This is my var" -option readonly -Visibility public -Scope script`
- Removal
  - `Remove-Variable $a`
  - `Remove-Item variable:a`
  - `Clear-Variable $b`

Command vs. Expression mode, Expansion

- Command mode
  - `Write-Host -Object "sg"` → `Write-Host -Object sg`
  - `Write-Host -Object 1+2` → `1+2`
  - But: `$a = 1; Write-Host $a` → `1`
- Expression mode
  - `1+2` → `3`
  - `Write-Host -Object (1+2)` → `3`
- Expansion
  - `PS C:\> $a = "Some text"`
  - `PS C:\> Write-Host "The a: $a"`
  - `The a: Some text`
  - `PS C:\> Write-Host "`$a is $a"`
  - `$a is Some text`
  - `PS C:\> Write-Host "Length of `$a is $($a.length)"`
  - `Length of $a is 9`

Outputting Results

- General output is table
  - Defined in format XML files
  - Columns: properties
  - Rows: objects
  - Not all properties are displayed
- Overriding default output format
  - `Get-Process | Format-Table -property name, product`
  - `Get-Process powershell | Format-List -property *size*`
  - `Get-Item c:\Windows | gm | ft -wrap`
  - `New-Object -Property @{Name = "MyObject"; Length = 10} | ft -autosize`

Error Handling Basics

- Non-defined variable does not produce error
- Non-existant property does not produce error
- Use `Set-Strictmode -version` latest
- Most errors are non-terminating
- Make an error terminating use ... `-erroraction Stop`
- Suppressing errors
  - ... `-erroraction silentlycontinue`
  - `Try {} catch{} finally{}`
- Getting errors
  - `$error[0]`
  - `$?`

What tools will you need?

- PowerShell
- PowerShell ISE
- PowerGUI
- Reflector or ILSpy
- RegexBuddy

Main points so far

- Everything is an object in PowerShell
  - FE: `get-help write-host | get-member`
  - Generally don't treat output as strings
- Examine members of the objects
  - `$variable | Get-Member`
- Use `Get-Help -full` or `Get-Help -examples` or `Get-Help -online`
- Use the TAB copletion
- Don't use any additional pipeline element after `format-...`
- Use `Write-...` if you don't want output, but want to see the result
- Use Strict Mode to avoid spelling errors

PRACTICE

Running External Commands

- Similar commands are aliased
  - `Dir c:\` → `Get-ChildItem c:\`
- Windows app
  - Notepad.exe
- Console app
  - IPConfig
- Return code of external command
  - `$lastexitcode`
- Leverage the use of CSV output

```
PS C:\> whoami -user /fo CSV | ConvertFrom-Csv | ft -AutoSize
User Name      SID
---------      ---
soost-pc\soost S-1-5-21-1879730660-3170798090-2708336039-1000
```

Hello, World!

- ```
  PS C:\> "Hello, World!"
  Hello, World!
  ```
- ```
  PS C:\> Write-Host -Object "Hello, World!"
  -ForegroundColor Green
  Hello, World!
  ```
- First example uses default outputting
- Second example uses `Write-Host` but there is no real output, that is just „text on screen”

Managing Text Files

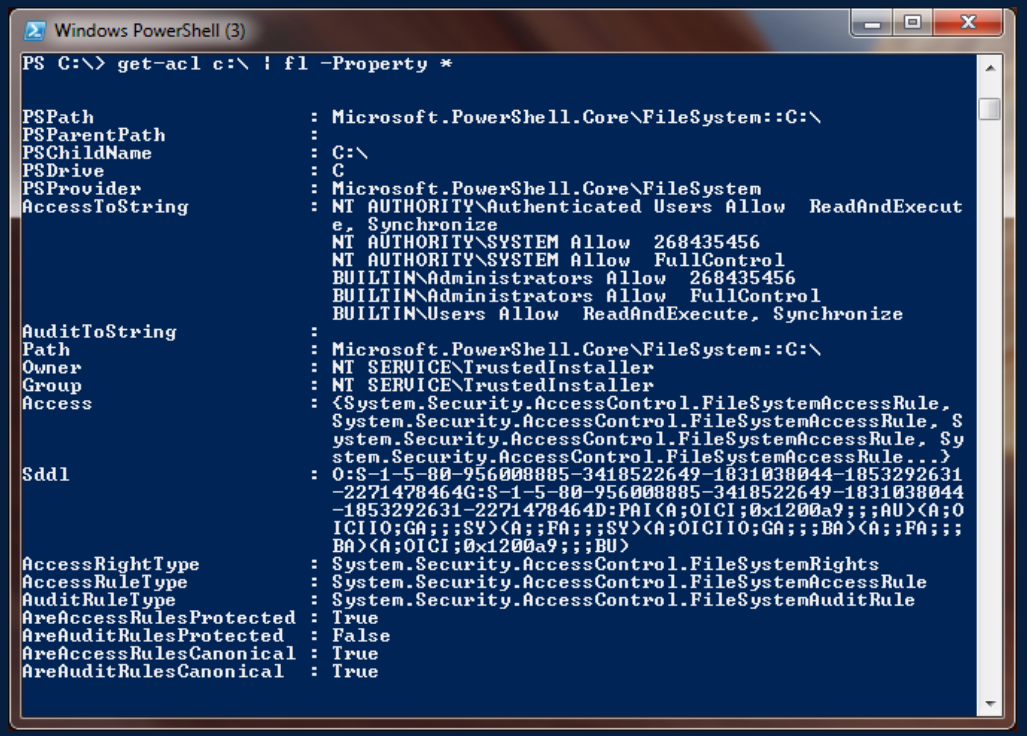
- Read text files
  - `Get-Content C:\Windows\WindowsUpdate.log` → Collection of strings of each line
  - `import-csv C:\ee\cars.csv -UseCulture`
- Write text files
  - `"XYZ" | Set-Content c:\ee\s.txt -Encoding ascii`
  - `... > c:\ee\textfile.txt`
  - `$myobject | export-csv c:\ee\myobj.csv -notypeinformation`
- Search text files
  - `select-string C:\Windows\WindowsUpdate.log -Pattern "error"`

Managing Files and Folders

- `New-Item -path c:\where -name What -itemtype directory`
- `Test-Path -path c:\where`
- Join-Path
  - `PS C:\> Join-Path "c:\" "subfolder"`
    `c:\subfolder`
  - `PS C:\> Join-Path "c:" "subfolder"`
    `c:\subfolder`
  - `PS C:\> Join-Path "c:\" "\subfolder"`
    `c:\subfolder`
- Split-Path
  - `PS C:\> Split-Path c:\subfolder\file.txt`
    `c:\subfolder`
  - `PS C:\> Split-Path c:\subfolder\file.txt -Leaf`
    `file.txt`
  - `PS C:\> Split-Path c:\subfolder\file.txt -Qualifier`
    `c:`
- `Get-Item`, `Get-ChildItem`, `Copy-Item`, `Remove-Item`, `Rename-Item`, …

Managing Security

- **Get-ACL**
  - Reads security descriptor
- **Set-ACL**
  - Writes security descriptor, commits changes



Managing the Registry

- Very similar to managing files and folders
  - `Get-ChildItem HKLM:\SOFTWARE`
  - `New-Item -Path HKCU:\SOFTWARE -Name BME -ItemType key`
    - ItemType: dynamic parameter
- Get-ACL, Set-ACL also works
- Actual registry data are properties
  - `(get-itemproperty 'HKCU:\Control Panel\Desktop').screensavetimeout`
  - For remote registry use the .NET class
    - `[Microsoft.Win32.RegistryKey]::OpenRemoteBaseKey($base,$COMPUTERNAME)`

Remote Execution

- Ad-hoc
  - `Get-Service -computername W7`
  - `Invoke-Command -ComputerName W7 -ScriptBlock {get-service}`
  - `Get-Process -computername W1, W2, W3`
- Persistent
  - `$s = New-PSSession -ComputerName vmvpc010963`
  - `Invoke-Command -Session $s -ScriptBlock {Get-Service -Name s*}`
  - `...`
  - `Remove-PSSession $s`
- Entering into a remote session
  - `PS C:\> $s = New-PSSession -ComputerName vmvpc010963`
  - `PS C:\> Enter-PSSession $s`
  - `[vmvpc010963]: PS C:\Windows\system32> $env:COMPUTERNAME VMVPC010963`
  - `[vmvpc010963]: PS C:\Windows\system32> exit`

Using WMI

- Enumerating classes
  - `Get-WmiObject -Class *system* -List`
- Get instances
  - `Get-WmiObject -Class *system* -List`
  - Only subset of properties are shown by default, use `...| fl *`
- Works remotely
  - `Get-WmiObject -Class win32_process -ComputerName vmvpc010963`

Using .NET

- Simple web client – instance of a class
  - `$wc = New-Object -TypeName net.webclient`
  - `$wc.DownloadString("http:\\portal.bme.hu")`
- DNS name resolution – use of static methods
  - `[net.dns]::GetHostEntry("portal.bme.hu")`
- Maths – enumeratung static members
  - `[math] | Get-Member -Static`

Functions

- Simple function

```
function Greeting ($name = "Default Value"){
    Write-Host "Hello, $name!"
}
```

- Calling a function
  - `Greeting -name Tibi`
- Managing pipelines

```
function SumProduct ([double] $multiplicator){
begin{ $sum = 0}
process { $sum += $_ * $multiplicator}
end { $sum }
}
```

  - `1,2,3 | SumProduct -multiplicator 2`
- Scopes
  - Global, local, parent

Scripts

- Saved as .ps1 files
- `Set-ExecutionPolicy remotesigned`
- Very similart to functions

```
function Greeting ($name = "Default Value"){
Param ($name = "Default Value")
    Write-Host "Hello, $name!"
}
```

- Calling a script
  - `.\greeting.ps1 -name Tibi`
- Scopes
  - Global, script, local, parent
- Dotsourcing
  - `. .\greeting.ps1 -name Tibi`
  - Local variables promoted to parent scope

Questions?

Q&A

Links

- In Hungarian – by Tibor Soós
  - PowerShell book – on line browsing
  - PowerShell book – downloadable
  - PowerShell environment - video – use TechSmith codec for videos
  - Language elements video
  - Some examples video
  - Managing Active Directory - video
  - Managing WMI - video
  - Managing Security - video
  - 10 short videos on TechNetKlub.hu
- In English
  - Survival guide
  - http://PowerShell.com
  - PowerShell TechNet Wiki
  - Hey, Scripting Guy blog