Serverové systémy Microsoft Windows

IW2/XMW2 2013/2014

Jan Fiedor

ifiedor@fit.vutbr.cz

Fakulta Informačních Technologií

Vysoké Učení Technické v Brně

Božetěchova 2, 612 66 Brno

Revize 7. 5. 2014

Windows PowerShell

Pro správu serveru a Active Directory

Windows PowerShell

- Konzole (příkazový řádek) systému Windows
  - Běží na platformě .NET (*.NET Framework*)
  - Pracuje s .NET objekty (*object-based*) namísto textu
  - Manipulace s objekty pomocí *cmdletů*
    - Jednoduché (*single-function*) vestavěné příkazy
  - Podpora všech nástrojů pro příkazovou řádku (**cmd**)
- Skriptovací jazyk
  - Imperativní, dynamicky typovaný skriptovací jazyk
  - Přístup ke všem objektům a funkcím platformy .NET

Základy práce s cmdlety

- Výpis seznamu všech cmdletů / aliasů (zkratek)

```
PS C:\> get-command * -CommandType { cmdlet | alias }
```

- Výpis nápovědy k cmdletu

```
PS C:\> get-help <cmdlet> [ -detailed ] [ -full ]
```

- Definice nového aliasu (zkratky) cmdletu

```
PS C:\> set-alias <alias> <cmdlet>
```

Formátování výstupu

- Cmdlet **Format-List** (alias **fl**)
  - Seznam objektů, více vlastností, objekty za sebou

```
PS C:\> format-list –Property <names> -GroupBy <prop>
```

- Cmdlet **Format-Table** (alias **ft**)
  - Tabulka objektů, více vlastností, každý objekt 1 řádek

```
PS C:\> format-table –Property <names> -GroupBy <prop>
```

- Cmdlet **Format-Wide** (alias **fw**)
  - Tabulka objektů, jedna vlastnost ve více sloupcích

```
PS C:\> format-wide –Property <name> -GroupBy <prop>
```

Řazení, filtrování a výběr objektů

- Cmdlet **Sort-Object** (alias **sort**)
  - Řazení objektů podle hodnot jejich vlastností

```
PS C:\> sort-object –Property <names> [-CaseSensitive]
  [-Culture <name>] [-Descending] [-Unique]
```

- Cmdlet **Where-Object** (aliasy **where** a **?**)
  - Filtrování objektů na základě předloženého skriptu

```
PS C:\> where-object -FilterScript <scriptblock>
```

- Cmdlet **Select-Object** (alias **select**)
  - Výběr objektů na začátku, konci, určitých pozicích, …

```
PS C:\> select-object [–First <int>] [-Last <int>]
 [-Skip <int>] [-Index <ints>] [-Unique]
```

Zřetězování příkazů (*pipelining*)

- Přeposílání výstupu (objektů) jednoho příkazu na vstup jiného příkazu přes rouru (*pipe*)

```
PS C:\> <příkaz> | <příkaz> [ | <příkaz> … ]
```

- Příklady

```
# Výpis tabulky názvů služeb seskupených podle stavu
get-service | sort-object -Property Status |
  format-table -Property Name -GroupBy Status
# Výpis 5 procesů, které mají nejvyšší využití paměti
get-process | sort-object -Property WS -Descending |
  select-object -First 5
# Výpis všech služeb, jenž aktuálně běží na počítači
get-service | where-object { $_.Status -eq "running" }
```

Skripty a omezování jejich spouštění

- Soubory s příponou **.ps1**
- Spouštění se řídí nastavením zásad spouštění
  - Zjištění aktuálního nastavení zásad spouštění

```
PS C:\> get-executionpolicy
```

  - Nastavení zásad spouštění pro konkrétní rozsah

```
PS C:\> set-executionpolicy <úroveň> -Scope <rozsah>
```

Rozsah	Popis
Process	Nastavení zásad spouštění ovlivňuje aktuální PowerShell proces
CurrentUser	Nastavení zásad spouštění ovlivňuje aktuálního uživatele
LocalMachine	Nastavení zásad spouštění ovlivňuje všechny uživatele na počítači

Úrovně zásad spouštění skriptů

- Podporováno celkem 7 úrovní zásad spouštění

Úroveň	Verze	Popis
Default	**1.0**	Výchozí nastavení zásad spouštění (aktuálně **Restricted**)
Restricted	**1.0**	Nelze spouštět skripty ani načítat konfigurační soubory
AllSigned	**1.0**	Všechny skripty a konfigurační soubory musí být podepsány (*signed*) důvěryhodnou autoritou
RemoteSigned	**1.0**	Všechny skripty a konfigurační soubory stažené z internetu musí být podepsány (*signed*) důvěryhodnou autoritou
Unrestricted	**1.0**	Lze spouštět skripty a načítat konfigurační soubory, spouštění skriptů stažených z internetu vyžaduje potvrzení od uživatele
Bypass	**2.0**	Lze spouštět skripty a načítat konfigurační soubory, spouštění skriptů stažených z internetu nevyžaduje potvrzení uživatele
Undefined	**2.0**	Odebere aktuální nastavení zásad spouštění z daného rozsahu (nelze použít pro rozsah zásad skupiny (**Group Policy** *scope*))

Proměnné (*Variables*)

- Deklarovány při prvním použití (při přiřazení dat)
- Názvy musí začínat znakem dolar (**$<*název*>**)
- Příklady
  - Deklarace proměnných přiřazením

```
$str = "text"
# Deklarace více proměnných zároveň
$str1, $str2, $str3 = "text1", "text2", "text3"
# Deklarace proměnných konkrétního datového typu
[int]$number = 12
```

  - Deklarace proměnné pomocí cmdletu

```
set-variable -name str -value "text"
```

Speciální proměnné

Proměnná	Význam
$_	Aktuální *pipeline* objekt (aktuální objekt při iterování přes objekty)
$?	Obsahuje výsledek (úspěch / neúspěch) poslední operace
$args	Obsahuje parametry příkazové řádky
$error	Obsahuje poslední chybu
$env:<název>	Reprezentuje proměnnou prostředí daného názvu (**$env:Path**, …)
$foreach, **$switch**	Enumerátory (ve **foreach** smyčce, ve **switch** větvení)
$input	Vstup, jenž je poslán přes rouru (*piped*) do funkce nebo bloku kódu
$match	Hashovací tabulka obsahující položky nalezené **match** operátorem
$myinvocation	Obsahuje informace o vykonávaném skriptu nebo příkazu
$host	Obsahuje informace o hostiteli vykonávajícím daný skript či příkaz
$true, **$false**, **$null**	Speciální hodnoty (pravda, nepravda, nulový objekt)
$stacktrace	Obsahuje zásobník volání (*stack trace*)

Konstanty (*Constants*)

- Jejich hodnota nemůže být změněna
- Nemohou být smazány
- Příklady
  - Deklarace konstanty

```
set-variable -name CSTR -value "ctext" -option constant
```

  - Použití konstanty

```
$str = $CSTR
```

Pole (*Arrays*)

- Indexované seznamy hodnot (.NET objektů)
- Příklady
  - Definice pole

```
$arr = @("text", 2, 4.6)
```

  - Konkatenace dvou polí

```
$arr2 = $arr + @("text2", 1, 3.5)
```

  - Přístup k prvkům pole (první prvek má index **0**)

```
# První prvek pole
$first = $arr[0]
# Poslední prvek pole
$last = $arr[$arr.count - 1]
```

Výstup (*Output*)

- Cmdlet **Write-Host**
  - Vypisuje objekty v hostiteli (nejčastěji konzole)

```
write-host <objekt> -NoNewline -Separator <řetězec>
  -BackgroundColor <barva> # Nastavení barev nemusí být
  -ForegroundColor <barva> #   podporováno hostitelem
```

- Cmdlet **Write-Output** (aliasy **echo** a **write**)
  - Posílá objekty dalšímu příkazu v zřetězení (*pipeline*)
  - Pokud další příkaz není, vypíší se objekty do konzole

```
write-output <objekt>
write <objekt>
echo <objekt>
```

Práce s datovými typy

- Informace o datovém typu proměnné

```
PS C:\> <proměnná>.GetType()
```

- Informace o prvcích (metodách, …) proměnné

```
PS C:\> get-member -InputObject <proměnná>
```

- Přetypování proměnné

```
<nová-proměnná> = [<typ>]<proměnná>
```

- Příklad

```
$str = "12:00"; write-host $str.GetType()
$date = [datetime]$str; write-host $date.GetType()
get-member -InputObject @($str, $date) # Prvky Object[]
@($str, $date) | get-member # Prvky String a DateTime
```

Foreach

- Alias cmdletu **ForEach-Object**
- Použití

```
# Jako klíčové slovo, přístup k prvku přes <item>
foreach (<item> in <collection>) { <statements> }
# Jako cmdlet, přístup k prvku přes proměnnou $_
<collection> | foreach { <statements> }
```

- Příklad

```
$arr = @("text", 2, 4.6)
# Iterace přes prvky pole
foreach ($item in $arr){ $item.GetType().FullName }
# Iterace přes prvky poslané přes rouru (pipe)
$arr | foreach { $_.GetType().FullName }
```

For a while

- Použití

```
# For
for (<init>; <condition>; <action>) { <statements> }
# While
while (<condition>) { <statements> }
```

- Příklad

```
$arr = @("text", 2, 4.6)
# Iterace přes prvky pole pomocí for
for ($i = 0; $i -lt $arr.Count; $i++) { $arr[$i] }
# Ekvivalentní zápis s využitím foreach
foreach ($i in (0 .. ($arr.Count - 1))) { $arr[$i] }
# Iterace přes prvky pole pomocí while
$i = 0; while ($i -lt $arr.Count) { $arr[$i]; $i++ }
```

Porovnávací operátory

Operátor	C / C++	Popis
-eq	**==**	Je rovno
-ne	**!=**	Není rovno
-gt	**>**	Větší než
-ge	**>=**	Větší než nebo rovno
-lt	**<**	Menší než
-le	**<=**	Menší než nebo rovno
-like		Odpovídá výrazu se zástupnými (*wild card*) znaky
-notlike		Neodpovídá výrazu se zástupnými (*wild card*) znaky
-match		Odpovídá regulárnímu výrazu
-notmatch		Neodpovídá regulárnímu výrazu

If … elseif … else

- Použití

```
# If … elseif … else
if (<condition>) { <statements> }
elseif (<condition>) { <statements> }
else { <statements> }
```

- Příklad

```
if ($args[0] -is [datetime]) {
  write-host "Time from datetime:" $args[0]
} elseif ($args[0] -is [string]) {
  write-host "Time from string:" ([datetime]$args[0])
} else {
  write-host "Invalid time."
}
```

Switch

- Použití

```
switch [-wildcard] [-regex] (<variable/collection>) {
  <value> { <statements> }
  default { <statements> }
}
```

- Příklad

```
switch -wildcard ($args) {
  3.14 { write-host "PI" }
  42 { write-host "Answer to The Ultimate Question" }
  "*help*" {
    write-host $myinvocation.MyCommand.Name "<arg>" }
  default { write-host "Unknown parameter" $_ }
}
```

Funkce

- Definice funkce

```
function <name>
{ # Přístup k parametrům přes proměnnou $args
  <statements>
}
```

- Definice funkce s navázáním parametrů

```
function <name> ( <arg> = <default>, … ) {
  <statements>
}
```

```
function <name> {
  param( <arg> = <default>, … )
  <statements>
}
```

Volání funkcí

- Volání funkce

```
<function>( <value>, …) # Parametry odděleny čárkou
<function> <value> … # Parametry odděleny mezerou
<function> -<arg> <value> … # Pro navázané parametry
```

- Příklad

```
function print {
  write-host $args[0] $args[1]
}
function printb($x = "hello", $y = "world") {
  write-host $x $y
}
print("hello", "world") # Výstup: hello world
print "hi"; printb "hi" # Výstup: hi resp. hi world
printb -y "earth"       # Výstup: hello earth
```

Vytváření a přístup k objektům

- Vytvoření nového .NET objektu
  - Cmdlet **New-Object**

```
new-object [-TypeName] <dotnet-class>
```

- Připojení ke COM objektu
  - Cmdlet **New-Object**

```
new-object -ComObject <com-class>
```

- Připojení k WMI objektu
  - Cmdlet **Get-WmiObject** (alias **gwmi**)

```
get-wmiobject [-Class] <wmi-class>
```

Předávání objektů pomocí rour (*pipes*)

- Zápis do roury pomocí cmdletu **Write-Output**
- Zapsané objekty uloženy v proměnné **$input**

```
function producer {
  for ($i = 0; $i -lt 10; $i++) {
    write-output $i
  }
}
function consumer {
  foreach ($object in $input) {
    write-host $object
  }
}
producer | consumer # Generování objektů funkcí
@("text", 2, 4.6) | consumer # Předání objektů přímo
```

Vytvoření uživatelů v Active Directory

```
function Create-UserInGroup($Group) {
  # Heslo musí být datového typu SecureString
  $password = ConvertTo-SecureString "aaa" `
    -AsPlainText –Force
  $adgroup = New-ADGroup $Group Global -PassThru `
    -Path "CN=Users,DC=testing,DC=local"
  foreach ($user in $input) {
    # Znak ` slouží k zalomení příkazu na další řádek
    $aduser = New-ADUser $user -Enabled $true `
      -AccountPassword $password -PassThru `
      -Path "CN=Users,DC=testing,DC=local"
    Add-ADGroupMember $adgroup $aduser
  }
} # Vytvoří 5 uživatelů ve skupině Simpsons
@("homer", "marge", "bart", "lisa", "maggie") |
  Create-UserInGroup -Group "Simpsons"
```

Vytvoření stínové skupiny

```
function Get-UsersInOU($OU) {
  # Vyhledávání probíhá na základě where-like filtru
  $adou = Get-ADOrganizationalUnit `
    -Filter { Name -eq $OU } # Name nemusí být unikátní
  Get-ADUser -Filter * -SearchBase $adou `
    -SearchScope OneLevel # Prohledá pouze zadanou OU,
}                         #   ne podřízené (child) OU
function Create-ShadowGroup {
  param($OU, $Group = "SG_" + $OU)
  $adsgroup = New-ADGroup $Group Global -PassThru `
    -Path "CN=Users,DC=testing,DC=local"
  foreach ($user in Get-UsersInOU($OU)) {
    Add-ADGroupMember $adsgroup $user
  }
} # Vytvoří stínovou skupinou s uživateli z OU Simpsons
Create-ShadowGroup -OU "Simpsons"
```

Zálohování GPO objektů

```
function Backup-GPOsToFolder($Path) {
  $Path += "\" + (Get-Date -Format "yyyy-MM-dd@HH-mm")
  New-Item $Path -Type Directory # Vytvoření adresáře
  Backup-GPO -All -Path $Path
}
function Backup-Folder($Path, $Target) {
  $policy = New-WBPolicy # Nastavení zálohování
  $backupdir = New-WBFileSpec -FileSpec $Path
  Add-WBFileSpec -Policy $policy -FileSpec $backupdir
  $targetvol = New-WBBackupTarget -VolumePath $Target
  Add-WBBackupTarget -Policy $policy -Target $targetvol
  Start-WBBackup -Policy $policy
}
# Zálohuje GPO objekty do adresáře, pak tento adresář
Backup-GPOsToFolder -Path "C:\Backup"
Backup-Folder -Path "C:\Backup" -Target "E:"
```

Vytvoření domovských adresářů (1)

```
function Allow-FullControl($Identity) {
  # Vytvoření účtu reprezentujícího identitu
  $account = New-Object `
    System.Security.Principal.NTAccount($Identity)
  # Vytvoření zástupné konstanty pro .NET třídu
  Set-Variable FileSystemAccessRule -Option Constant `
    System.Security.AccessControl.FileSystemAccessRule
  # Vytvoření ACE položky ACL seznamu
  New-Object $FileSystemAccessRule ( `
    $account, # Specifikace identity
    "FullControl", # Specifikace oprávnění
    "ContainerInherit, ObjectInherit", # Dědit dále
    "None", # Aplikovat oprávnění a propagovat dále
    "Allow" # Typ položky (povolit / odepřít)
  ) # Zalamování příkazů je možné i pomocí komentářů
}
```

Vytvoření domovských adresářů (2)

```
function Create-HomeFolder($User, $HomeRoot) {
  $homedir = New-Item ($HomeRoot + "\" + $User) `
    -Type Directory
  # Získání ACL seznamu domovského adresáře uživatele
  $acl = Get-Acl $homedir.FullName
  # Zrušení dědičnosti a odebrání zděděných oprávnění
  $acl.SetAccessRuleProtection($true, $false)
  foreach ($identity in `
    @("SYSTEM", "Administrators", $User)
  ) { # Parametr musí být proměnná => užití $(<funkce>)
    $acl.AddAccessRule($(Allow-FullControl($identity)))
  } # Nastavení nového ACL seznamu domovskému adresáři
  Set-Acl $homedir.FullName $acl
  # Odeslání cesty k domovskému adresáři do roury
  Write-Output $homedir
}
```

Vytvoření domovských adresářů (3)

```
function Set-Quota($Path, $Limit = 500MB) {
  # Připojení k správci prostředků přes COM rozhraní
  $qmgr = New-Object -ComObject Fsrm.FsrmQuotaManager
  if ($Path) { # Test, zda je proměnná definována
    $input += $Path # Přidání prvku do pole (kolekce)
  } # Vytvoření kvóty pro každý předaný home adresář
  foreach ($path in $input) {
    $quota = $qmgr.CreateQuota($path)
    $quota.QuotaLimit = $Limit # Nastavení limitu
    $quota.Commit() # Potvrzení (zapsání) kvóty
  }
} # Vytvoření home adresářů pro uživatele v OU Simpsons
foreach ($user in Get-UsersInOU("Simpsons")) {
  Create-HomeFolder $user.Name -HomeRoot "C:\Homes" |
    Set-Quota -Limit 1GB # Adresáře předány přes rouru
}
```