EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

jihomoravský kraj

OPERAČNÍ SYSTÉMY

Tvorba skriptů v PowerShellu

Metodický list

Autor: Ing. Marek Kocan, Metodik: Ing. Roman Koláčný

Recenzent: Mgr. Jiří Činčura

Rok vydání: 2023

Obsah

Cíle

Studenti se seznámí s teoretickými základy v oblasti PowerShellu, tedy rozšířeného shellu společnosti Microsoft používaného primárně v operačních systémech Windows. Dále si osvojí základní dovednosti pro vytváření jednoduchých skriptů, a to na základě konkrétních příkladů předvedených vyučujícím i na základě samostatně vyzkoušených úkolů.

Student bude schopen vlastními slovy vysvětlit co je to PowerShell, dokáže popsat jeho základní charakteristiky a zvládne vytvořit i spustit jednoduché skripty. Student dále dokáže vlastními slovy popsat základní programové konstrukce. V dlouhodobém horizontu bude student schopen samostatně využít nové znalosti a dovednosti pro zjednodušení správy operačního systému, zejména pak s ohledem na možnosti automatizace pomocí skriptů.

Dovednosti

Student bude schopen vytvářet a spouštět skripty v PowerShellu, a to včetně využití základních programových konstrukcí včetně vstupu, výstupu, podmínek a cyklů. Dále student zvládne zapracovat do skriptů či v rámci jejich spouštění využívat i další prvky operačních systémů. V dlouhodobém horizontu bude student schopen aplikovat získané dovednosti pro efektivní práci v operačních systémech společnosti Microsoft.

Pracovní prostředí

Výuku lze realizovat v prostředí:

- Cylab JCEKB, operační systém Windows 10 (dle volby vyučujícího workstation či server)

Pro práci postačí standardní nástroje.

1 PowerShell – teoretický základ

PowerShell je příkazové prostředí se skriptovacím jazykem, které primárně pracuje s objekty. Základem je .NET Common Language Runtime. Počátky spadají do podzimu roku 2006, aktuálně je k dispozici verze 7.3. Vazba na .Net mj. zajišťuje to, že vše dostupné v rámci .NET frameworku lze využívat i v PowerShellu.

Na rozdíl od tradičních textových shellů (CMD, případně unixové shelly) pracuje PowerShell s objekty, a tedy například výstupy nejsou primárně textové, ale objektové – což má vliv i na řetězení příkazů (roury), v rámci kterého jsou předávány místo textů objekty. Všechny objekty v PowerShellu jsou .NET objekty.

Základní programovou jednotkou jsou cmdlety, tedy přeložené příkazy vytvořené v programovacím jazyce kompatibilním s .NET. Pomocí těchto základních příkazů lze vytvářet funkce a skripty (lze volat i další programy a samostatné součásti operačního systému, ty jsou nicméně spuštěné jako samostatný proces).

Názvosloví cmdletu má přesná pravidla – skládá se ze dvou částí oddělených pomlčkou, slovesa a podstatného jména. Povolená (doporučená) slovesa Get-Verb, podstatné jméno je v prvním pádu jednotného čísla. Názvy je vhodné volit vždy podle významu, například tedy Get pro získání apod.

1.1 Přenositelnost

PowerShell je k dispozici i na dalších operačních systémech, v případě Linuxu jsou podporovány různé distribuce (Debian, Fedora, Ubuntu …). Nejde nicméně o 100% přenositelnost, rozdíly se týkají jak samotné funkčnosti, tak i například mechanismu řízení oprávnění. Vstupní informace k problematice jsou k dispozici například https://learn.microsoft.com/cs-cz/powershell/scripting/install/installing-powershell-on-linux?view=powershell-5.1 a https://learn.microsoft.com/cs-cz/powershell/scripting/whats-new/unix-support?view=powershell-5.1. Vyučující dle své volby a zkušeností může se studenty více diskutovat PowerShell core a xplat.

Například v případě Debinu je situace k 11/2022 následující:

- Ikona ✅ označuje, že verze operačního systému nebo PowerShellu je stále podporovaná.
- Ikona 🟡 označuje, že verze PowerShellu už není v dané verzi operačního systému podporovaná.
- Ikona 🚧 označuje, že jsme nedokončili testování PowerShellu v daném operačním systému.
- Ikona ❌ označuje, že verze operačního systému nebo PowerShellu není podporovaná.
- Pokud má verze operačního systému i verze PowerShellu ikonu ✅, je tato kombinace podporovaná.

Debian	7.0 (LTS)	7.1	7.2 (LTS-current)	7.3
✅ 11	✅	🟡	✅	✅
✅ 10	✅	🟡	✅	✅
✅ 9	🟡	🟡	🟡	🟡
❌ 8	🟡	❌	❌	❌

2 Tvorba skriptů v PowerShellu – praktické ukázky a cvičení

V této části vyučující představí jednotlivé příklady související s tvorbou skriptů v PowerShellu. Na tyto ukázky bude průběžně navazovat samostatná práce studentů (kontrolní body).

2.1 Pracovní prostředí

Výuku lze realizovat v prostředí Cylab JCEKB, operační systém Windows 10 (dle volby vyučujícího workstation či server)

2.2 Základní informace pro tvorbu skriptů

2.2.1 Seznam cmdletů a funkcí

Vyučující seznámí studenty s teoretickými základy a předvede spuštění pracovního prostředí PowerShell.

Jedním z prvních kroků pro zorientování se v novém prostředí je získání seznamu dostupných cmdletů/funkcí (dále až na výjimky související s probíranou částí budeme hovořit o příkazech, *vyučující nicméně ještě jednou zdůrazní podstatu pojmu cmdlet*). V případě PowerShellu k tomu využijeme příkaz `Get-Command`.

```
PS c:\Users\syadmin> Get-Command
```

Výstup (zkrácený)

```
CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Alias           Add-AppPackage                                     2.0.1.0    Appx
Alias           Add-AppPackageVolume                               2.0.1.0    Appx
Alias           Add-AppProvisionedPackage                          3.0        Dism
Alias           Add-ProvisionedAppPackage                          3.0        Dism
Alias           Add-ProvisionedAppxPackage                         3.0        Dism
Alias           Add-ProvisioningPackage                            3.0        Provisioning
...
```

Podobně jako v případě ostatních shellů lze i v případě PowerShellu omezit rozsah vypsaných příkazů, v následujícím příkladu budou vypsány všechny příkazy začínající na `Get-`.

```
PS c:\Users\syadmin> Get-Command -Name Get-*
```

Výstup (zkrácený):

```
CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Alias           Get-AppPackage                                     2.0.1.0    Appx
Alias           Get-AppPackageDefaultVolume                        2.0.1.0    Appx
Alias           Get-AppPackageLastError                            2.0.1.0    Appx
Alias           Get-AppPackageLog                                  2.0.1.0    Appx
Alias           Get-AppPackageManifest                             2.0.1.0    Appx
Alias           Get-AppPackageVolume                               2.0.1.0    Appx
...
```

Kontrolní bod

Studenti zjistí seznam všech příkazů začínající na Set-.

2.2.2 Identifikace prostředí

Podobně jako v případě linuxového světa je i v případě PowerShellu vědět, s jakou verzí a v jakém prostředí pracujeme. K tomu lze využít například příkaz `Get-Host`, který mj. zobrazí i informace o verzi a základním nastavení.

```
PS c:\Users\syadmin> Get-Host
```

Výstup:

```
Name             : ConsoleHost
Version          : 5.1.19041.1682
InstanceId       : 7f13ab44-5bc6-4684-b964-f47227c7fe57
UI               : System.Management.Automation.Internal.Host.InternalHostUserInterface
CurrentCulture   : cs-CZ
CurrentUICulture : cs-CZ
PrivateData      : Microsoft.PowerShell.ConsoleHost+ConsoleColorProxy
DebuggerEnabled  : True
IsRunspacePushed : False
Runspace         : System.Management.Automation.Runspaces.LocalRunspace
```

Vyučující diskutuje aktuálně používanou verzi a zopakuje studentům základní informace o verzích PowerShellu.

Dle situace vyučující dále případně diskutuje objektové vlastní PowerShellu s ohledem na „vše je vlastně objekt“ a využije k tomu `Get-Process -Name explorer | Get-Member`.

2.2.3 Vybrané příkazy

Mezi další užitečné základní příkazy patří:

- `Get-ComputerInfo` – vypíše informace o počítači a operačním systému
- `Get-Process` – vypíše informace o spuštěných procesech
- `Get-Alias` – vypíše seznam aktuálních aliasů
- `Get-History` – vypíše historii dosud zadaných příkazů v rámci aktuálního sezení

```
PS c:\Users\syadmin> Get-ComputerInfo
```

Výstup (zkrácený):

```
WindowsBuildLabEx                                           :
19041.1.amd64fre.vb_release.191206-1406
WindowsCurrentVersion                                       : 6.3
WindowsEditionId                                            : Professional
WindowsInstallationType                                     : Client
WindowsInstallDateFromRegistry                              : 03.06.2022 23:26:57
WindowsProductId                                            : 00330-80000-00000-AA573
WindowsProductName                                          : Windows 10 Pro
...
```

Vyučující diskutuje zobrazený výstup a dle možností ukáže i jiné cesty, jak tyto informace získat.

```
PS c:\Users\syadmin> Get-Process
```

Výstup (zkrácený):

```
Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
    181      22    12036      23040       0,13   2228   1 ai
    221      22     2872      12012       0,05  10316   1 AnalyticsSrv
```

```
    444      40    16932      31648       0,23  11900   1 ApplicationFrameHost
    175      19     2852       7892              3400   0 armsvc
    211      21     7824      13824       0,23  14844   0 audiodg
    378      35     7352      19668       6,16   9080   1 AuthManSvr
    658      48    42536       1992       0,52  13280   1 Calculator
    689      45    28372      37704             10256   1 CitrixReceiverUpdater
...
```

Vyučující diskutuje zobrazený výstup a dle svého zvážení porovná s příkazem ps v linuxových prostředích.

```
PS c:\Users\syadmin> Get-Alias
```

Výstup (zkrácený):

```
CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Alias           % -> ForEach-Object
Alias           ? -> Where-Object
Alias           ac -> Add-Content
Alias           asnp -> Add-PSSnapin
Alias           cat -> Get-Content
Alias           cd -> Set-Location
Alias           CFS -> ConvertFrom-String                          3.1.0.0    Microsof...
...
```

Vyučující vysvětlí význam aliasu a s odkazem na výše uvedený Get-Process demonstruje alias ps.

Kontrolní bod

Studenti zobrazí definici jednoho konkrétního aliasu dle zadání vyučujícího (např. `cls`). Vyučující připomene, že lze použít obdobnou konstrukci omezení, jako v případě seznamu příkazů.

```
PS c:\Users\syadmin> Get-History
```

Výstup (zkrácený):

```
Id CommandLine
-- -----------
 1 Get-Command
 2 Get-Command -Name Get-*
...
```

2.2.4 Nápověda

PowerShell má propracovaný systém nápovědy, primárně dostupný pomocí příkazu `Get-Help`. Při prvním spuštění může dojít k časové prodlevě v odezvě v důsledku aktualizace dat.

```
PS c:\Users\syadmin> Get-Help
```

Výstup (zkrácený):

```
TOPIC
    Windows PowerShell Help System

SHORT DESCRIPTION
    Displays help about Windows PowerShell cmdlets and concepts.

LONG DESCRIPTION
    Windows PowerShell Help describes Windows PowerShell cmdlets,
    functions, scripts, and modules, and explains concepts, including
    the elements of the Windows PowerShell language.
```

```
    Windows PowerShell does not include help files, but you can read the
    help topics online, or use the Update-Help cmdlet to download help files
    to your computer and then use the Get-Help cmdlet to display the help
    topics at the command line.
...
```

Nápovědu lze využít k zobrazení informací o konkrétním příkazu, opět pomocí `-Name`.

```
PS c:\Users\syadmin> Get-Help -Name Get-Process
```

Výstup (zkrácený):

```
NAME
    Get-Process

SYNOPSIS
    Gets the processes that are running on the local computer or a remote computer.


SYNTAX
    Get-Process [[-Name] <System.String[]>] [-ComputerName <System.String[]>] [-
FileVersionInfo] [-Module] [<CommonParameters>]

    Get-Process [-ComputerName <System.String[]>] [-FileVersionInfo] -Id <System.Int32[]>
[-Module] [<CommonParameters>]

    Get-Process [-ComputerName <System.String[]>] [-FileVersionInfo] -InputObject
<System.Diagnostics.Process[]> [-Module] [<CommonParameters>]

    Get-Process -Id <System.Int32[]> -IncludeUserName [<CommonParameters>]

    Get-Process [[-Name] <System.String[]>] -IncludeUserName [<CommonParameters>]

    Get-Process -IncludeUserName -InputObject <System.Diagnostics.Process[]>
[<CommonParameters>]
...
```

Vyučující v obecné rovině diskutuje zobrazený výstup v syntaktické části.

2.2.5 Zobrazení definici příkazu

V případě potřeby lze velmi snadno zobrazit definici příkazu – výstup se liší podle toho, zda jde o cmdlet nebo o funkci.

```
PS c:\Users\syadmin> (Get-Command Get-Help).Definition
```

Výstup:

```
Get-Help [[-Name] <string>] [-Path <string>] [-Category <string[]>] [-Component <string[]>]
[-Functionality <string[]>] [-Role <string[]>] [-Full] [<CommonParameters>]

Get-Help [[-Name] <string>] -Detailed [-Path <string>] [-Category <string[]>] [-Component
<string[]>] [-Functionality <string[]>] [-Role <string[]>] [<CommonParameters>]

Get-Help [[-Name] <string>] -Examples [-Path <string>] [-Category <string[]>] [-Component
<string[]>] [-Functionality <string[]>] [-Role <string[]>] [<CommonParameters>]

Get-Help [[-Name] <string>] -Parameter <string> [-Path <string>] [-Category <string[]>] [-
Component <string[]>] [-Functionality <string[]>] [-Role <string[]>] [<CommonParameters>]

Get-Help [[-Name] <string>] -Online [-Path <string>] [-Category <string[]>] [-Component
<string[]>] [-Functionality <string[]>] [-Role <string[]>] [<CommonParameters>]

Get-Help [[-Name] <string>] -ShowWindow [-Path <string>] [-Category <string[]>] [-Component
<string[]>] [-Functionality <string[]>] [-Role <string[]>] [<CommonParameters>]
```

Vyučující vysvětlí, že jde o cmdlet a proto je zobrazena syntaxe použití.

```
PS c:\Users\syadmin> (Get-Command Get-Verb).Definition
```

Výstup:

```
param(
    [Parameter(ValueFromPipeline=$true)]
    [string[]]
    $verb = '*'
)
begin {
    $allVerbs =
[System.Reflection.IntrospectionExtensions]::GetTypeInfo([PSObject]).Assembly.ExportedTypes
|
        Microsoft.PowerShell.Core\Where-Object {$_.Name -match '^Verbs.'} |
        Microsoft.PowerShell.Utility\Get-Member -type Properties -static |
        Microsoft.PowerShell.Utility\Select-Object @{
            Name='Verb'
            Expression = {$_.Name}
        }, @{
            Name='Group'
            Expression = {
                $str = "$($_.TypeName)"
                $str.Substring($str.LastIndexOf('Verbs') + 5)
            }
        }
}
process {
    foreach ($v in $verb) {
        $allVerbs | Microsoft.PowerShell.Core\Where-Object { $_.Verb -like $v }
    }
}
# .Link
# https://go.microsoft.com/fwlink/?LinkID=160712
# .ExternalHelp System.Management.Automation.dll-help.xml
```

Vyučující vysvětlí, že jde o funkci a proto je zobrazen zdrojový kód. Vyučující vysvětlí, že zdrojové kód lze využít nejen pro informace o tom, jak daná funkce funguje, ale také pro případnou inspiraci. Dle situace a průběžných reakcí studentů ukáže výpis cmdletů a funkcí:

```
Get-Command -Name Get* -Type Cmdlet
```

```
Get-Command -Name Get* -Type Function
```

3 Proměnné

PowerShell obdobně jako další shelly umožňuje využívat i proměnné (*vyučující se může dále seznámit například s https://learn.microsoft.com/cs-cz/powershell/module/microsoft.powershell.core/about/about_variables?view=powershell-5.1*). Následující příklady jsou základním představením práce s proměnnými a představují vstup pro tvorbu skriptů.

Vyučující obecně volí míru popisu a vysvětlování dle dosavadních znalostí studentů o programovacích jazycích.

Proměnné jsou v PowerShellu označovány textem začínajícím znakem $. Obor platnosti je v případě interaktivní práce příslušné sezení (spuštěný shell).

3.1 Textové proměnné

Následující příklad vytvoří a iniciuje proměnnou s názvem `b` a obsahem `Text`. Tato proměnná je následně vypsána.

```
PS c:\Users\syadmin> $b="Text"
PS c:\Users\syadmin> Write-Output $b
```

Výstup:

```
Text
```

Další možností je použít pro výstup příkaz Write-Host, který slouží v výpisu přímo na obrazovku terminálu, ale neumožňuje předat výstup k dalšími zpracování do roury (*vyučující využije a vysvětlí dle znalostí studentů s rourami v operačních systémech linuxového typu*).

Výše použitý název je tzv. triviální, nicméně lze využívat i netriviální názvy, například `${Toto je moje promenna}= "Text".` Vyučující zdůrazní, že toto platí pro všechny datové typy, nejen pro textové proměnné.

Kontrolní bod

Studenti si vytvoří dvě textové proměnné, jednu s triviálním a jednu s netriviálním názvem a vyzkouší si jejich výpis.

3.2 Číselné proměnné

Následující příklad vytvoří a iniciuje celočíselnou proměnnou s názvem `cislo` a hodnotou `7`. Tato proměnná je následně vypsána.

```
PS c:\Users\syadmin> $cislo=7
PS c:\Users\syadmin> Write-Output $cislo
```

Výstup:

```
7
```

Dále je vytvořena proměnná s názvem `cislo2` a hodnotou `9`. Následně dojde k vytvoření nové proměnné se součtem obou dosavadních a k jejímu vypsání.

```
PS c:\Users\syadmin> $cislo2=9
PS c:\Users\syadmin> $vysledek=$cislo+$cislo2
PS c:\Users\syadmin> Write-Output $vysledek
```

Výstup:

```
16
```

Vyučující dle možností předvede práci s desetinnou tečkou.

3.3 Logické proměnné

Následující příklady demonstrují práci s logickými proměnnými (pravda / nepravda).

```
PS c:\Users\syadmin> $pravda=$True
PS c:\Users\syadmin> Write-Output $pravda
```

Výstup:

```
True
```

Obdobně lze vytvořit proměnnou s názvem `nepravda`.

```
PS c:\Users\syadmin> $nepravda=$False
PS c:\Users\syadmin> Write-Output $nepravda
```

Výstup:

```
False
```

Vyučující dále demonstruje přiřazení hodnot True a False (bez dolaru) a vysvětlí, že správné použití je právě s dolarem (předdefinované logické hodnoty).

Následující příklady demonstrují porovnání dvou vytvořených proměnných:

```
PS c:\Users\syadmin> $pravda -eq $nepravda
```

Výstup:

```
False
```

```
PS c:\Users\syadmin> $pravda -eq $pravda
```

Výstup:

```
True
```

3.4 Proměnné pro práci s datem

Následující příklady ukazují práci s datem. Nejprve dojde k vytvoření proměnné datového typu `DateTime` a k jejímu vypsání. *Vyučující vysvětlí konstrukci `[DateTime]` (určení datového typu).*

```
PS c:\Users\syadmin> [DateTime] $datum = "02/28/2022"
PS c:\Users\syadmin> Write-Output $datum
```

Výstup:

```
Write-Output $datum
```

Další příklad zobrazí informace o vytvořené proměnné.

```
PS c:\Users\syadmin> $datum.GetType()
```

Výstup:

```
IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     DateTime                                 System.ValueType
```

3.5 Další příklady pro práci s proměnnými (volitelně)

Dle situace a reakce studentů může vyučující využít i další příklady (`C` odpovídá příkazu, `O` výstupu).

```
C:      [int]$cislob=5                  #určení datového typu
C:      Write-Output $cislob
O:      5
C:      $cislob="text"
O:      Cannot convert value "text" to type "System ...
C:      [string]$cislob="text"
C:      Write-Output $cislob
O:      text

C:      [int]$cislob2=5
C:      Write-Output $cislob2
O:      5
C:      $cislob2="6"                    #automatická konverze
C:      Write-Output $cislob2
O:      6

C:      $datum | Get-Member
O:                                              #podrobnosti o objektu
C:      $datum.GetDateTimeFormats()
O:                                              #všechny řetězové reprezentace
C:      $datum.ToString("d")
O:      28.02.2022
C:      $datum.ToString("D")
O:      pondělí 28. února 2022
```

4 Řízení běhu

4.1 Podmínky

Následující příklady demonstrují podmínku `if`.

```
PS c:\Users\syadmin> $promenna=10
PS c:\Users\syadmin> if ($promenna -eq 10) {
                         echo "ANO"
                     }
```

Výstup:

```
ANO
```

V podmínce je využitý operátor `-eq`, který je vyhodnocený jako pravda v případě rovnosti obou operandů.

Následující příklad demonstruje podmínku `if`.

```
PS c:\Users\syadmin> if ($promenna1 -ne 20) {
                         echo "ANO"
                     }
```

Výstup:

```
ANO
```

V podmínce je využitý operátor `-ne`, který je vyhodnocený jako pravda v případě nerovnosti obou operandů.

Vyučující zmíní další operátory:

- `-gt` *větší než*

- `-ge` *větší nebo rovno*
- `-lt` *menší než*
- `-le` *menší nebo rovno*

Kontrolní bod

Studenti vytvoří podmínku na porovnání dvou celočíselných proměnných.

4.2 for cyklus

Následující příklad demonstruje využití cyklu for pro výpis hodnot od 0 do 10.

```
PS c:\Users\syadmin> for ($i=0; $i -le 10; $i++) {
                       Write-Host $i
                     }
```

Výstup:

```
0
1
2
3
4
5
6
7
8
9
10
```

Vyučující vysvětlí konstrukci cyklu a jeho jednotlivé části.

4.3 while cyklus

Následující příklad demonstruje využití cyklu for pro výpis hodnot od 1 do 10.

```
PS c:\Users\syadmin> $i=1
                     while ($i -le 10){
                       $i
                       $i++
                     }
```

Výstup:

```
1
2
3
4
5
6
7
8
9
10
```

Vyučující vysvětlí konstrukci cyklu a jeho jednotlivé části.

Kontrolní bod

Studenti diskutují, proč v případě `for` jde o hodnoty 0 až 10 a v případě `while` o hodnoty 1 až 10.

Studenti vytvoří cyklus typu dle svého výběru a vypíši hodnoty od 10 do 1 (v tomto pořadí).

5 Ukázkový skript

Následující skript na vyvolání uloží obsah textové schránky do výstupního souborů (aktuální a historie).

Skript mj. ukazuje využití:

- přesměrování výstupu textové schránky (`Get-Clipboard |`), jednou s přepsání a jednou s připsáním ke stávajícímu obsahu (`-Append`)
- náhrady řetězců (`replace`)
- odkaz na domovský adresář uživatele (`~`)

Pro vytvoření skriptu lze obecně využít jakýkoli textový editor.

```
$outputFile="~\schranka.out"
$dtout=Get-Date -Format "G"
$dt=$dtout -replace ' |:|\.',''
$dtfile="~\$dt.cpb"
$header="-----$dtout-----"
$footer="-----------------------------"
Write-Output $header | Out-File -Append -FilePath $outputFile
Get-Clipboard | Out-File -Append -FilePath $outputFile
Write-Output $footer | Out-File -Append -FilePath $outputFile
Get-Clipboard | Out-File -FilePath $dtfile
exit
```

Skript je vhodné nazvat například schranka.ps1 a z prostředí PowerShellu jej vyvolat pomocí `./schranka.ps1`. Před samotným spuštěním je nicméně nutné upravit bezpečnostní politiku pro spuštění skriptů, například pomocí `Set-ExecutionPolicy RemoteSigned -Scope Process`.

V případě dostačujícího času vyučující probere jednotlivé zásady, případně na ně odkáže s volbou samostudia (více např. https://docs.microsoft.com/cs-cz/powershell/module/microsoft.powershell.core/about/about_execution_policies?view=powershell-5.1):

- *AllSigned*
- *Bypass (vše povoleno, bez varování)*
- *Default*
- *RemoteSigned (nevyžaduje podpis vytvořených skriptů)*
- *Restricted*
- *Undefined*
- *Unrestricted*

a obor zásad:

- *MachinePolicy*
- *UserPolicy*

- *Process (v paměti, proměnná $env:PSExecutionPolicyPreference)*
- *CurrentUser*
- *LocalMachine*

Kontrolní bod

Studenti ve spolupráci s vyučujícím zkouší různé varianty Ctrl+C a spouštění skriptu.

Vyučující zmíní možnost využít tento skript v rámci plánování úloh a dle svého uvážení zadá realizaci za domácí úkol. Pro ten je možné využít parametry:

```
-ExecutionPolicy Bypass -WindowStyle Hidden -file c:\Users\sysadmin\schranka.ps1
```

Vyučující dle času a možností zmíní možnost nevyužití replace, ale přímo vhodného formátovacího řetězce, například:

```
Get-Date -Format "ddMMyyyyHHmmss"
```

Shrnutí a závěr

Studenti se seznámili se základními programovými konstrukcemi pro tvorbu skriptů v PowerShellu. Dle svého uvážení může vyučující připravit pro studenty například test, a to jak z pohledu teorie, tak i praktického opakování nad zdokumentovanými postupy, případně zadat za domácí úkol tvorbu jednoduchého skriptu například na následující témata:

- vytvořte skript, který demonstruje práci s proměnnými a podmínkami (náročnost 4/10)
- vytvořte skript, který vypíše obsah domovského adresáře (náročnost 2/10)
- vytvořte skript, který bude v nekonečné smyčce vypisovat text napevno zadaný v rámci skriptu (náročnost 3/10)

Seznam použitých zdrojů

Learn. *Dokumentace k prostředí PowerShell*. Dostupné z: https://learn.microsoft.com/cs-cz/powershell/