Aktualizováno: 28. února 2022

Windows PowerShell 5.1 a Powershell 7

PŘÍRUČKA PRO ZVLÁDNUTÍ ZÁKLADŮ WINDOWS POWERSHELL (WPS) DO VERZE 5.1 A NOVĚ PRO POWERSHELL VERZE 7.X (PS7).

Obsah

1. ÚVOD

WPS/PS7 je nadstavba příkazové řádky pro Windows a skriptovací prostředí navržené především pro systémové administrátory. Přináší do příkazové řádky sílu objektů a prostředí .NET Frameworku. Obsahuje řadu nových koncepcí umožňujících využít a rozšířit znalosti získané při vytváření skriptů pro příkazový řádek v prostředí jiných shellů. Odstraňuje dlouhotrvající problémy shellů a přidává spousty nových vlastností. Zlepšuje orientaci ve svých vlastnostech a dá se tak lépe zjistit, který příkaz zajišťuje daný úkol. Je konzistentní v používání příkazů – příkazy mají jednotný styl zápisu. WPS/PS7 není citlivý na velikost písmen (není case-sensitive).

WPS/PS7 byl navržen tak, aby využil uživatelovy historické znalosti CLI. Podporuje interaktivní prostředí (výstup příkazu v konzoli může být směrován do roury, na tiskárnu nebo do souboru) a stejně dobře podporuje skriptování (opakovaně použitelné příkazy jsou uloženy v souboru a na daný pokyn spouštěny) pomocí souborů s příponou PS1. A je samozřejmé, že ve WPS/PS7 můžete spouštět všechny aplikace, jak pro příkazový řádek, tak pro Windows. WPS/PS7 je rozšiřitelný o vlastní cmdlety a moduly nabízené i třetími stranami.

POZNÁMKA 1: GUI používá některé základní koncepty, které jsou dobře známé většině počítačových uživatelů – pomocí klikání na příkazy spouštíte potřebné funkce. V CLI však musíte použít zcela jiný přístup než v GUI, protože toto rozhraní nemá menu nebo GUI. Musíte znát jména příkazů a parametrů, dříve, než je použijete. I když můžete napsat složité příkazy, které jsou rovnocenné s funkcemi v GUI prostředí, je třeba se seznámit s běžně používanými příkazy a jejich parametry. Protože CLI byly první nadstavby operačních systémů, mnoho jmen příkazů a jejich parametrů bylo vybíráno tak, aby byly kompatibilní s dřívějšími povely, takže příkazy jsou stále ve tvaru starém desítky let. I když CLI obsahují systém nápovědy, tak se musíte každý CLI pracně učit, protože používá jinou logiku a jiná jména příkazů a parametrů (např. copy u MSDOS, cp u Unixu atp.).

1.1. Upozornění

V tomto manuálu nepřekládám následující výrazy, u kterých budu používat jejich anglické názvy:

- Activity, activities.
- Aliases.
- Checkpoint, checkpoints, checkpointing.
- Cmdlet, cmdlets.
- Managed node.
- Node, nodes.
- Persistance.
- Provider, providers.
- Runspace, runspaces.
- Runtime.
- Shell.
- Session, sessions.
- Snap-in.
- Snapshot, snapshots.
- Splatting.
- Tab-Completion.
- Workflow, workflows.

1.2. Konvence použité v tomto dokumentu

[]	Hranaté závorky obsahují nepovinný parametr.
<>	Špičaté závorky označují povinný parametr.
()	Kulaté závorky označují výraz nebo metodu.
{}	Složené závorky označují příkazy (spustitelný kód) čili blok kódu.
|	Znak roury.
#	Znak hash (historie příkazů).
WPS	Platí pro Windows Powershell do verze 5.1.
PS7	Platí pro Powershell od verze 7.
WPS/PS7	Platí pro Windows Powershel (do verze 5.1) i pro Powershel (od verze 7)

1.3. Slovníček pojmů

Activity, activities: Aktivita, úloha, konkrétní úkol, který se má provést v rámci workflow.

Alias, Aliases: Přezdívky (zkratky) užívané jako alternativa ke jménu standardního příkazu.

Case-insensitive: Necitlivost k velikosti písmen (Praha je totéž co praha).

Case-sensitive: Citlivost k velikosti písmen (Praha není totéž co praha).

Checkpoint, checkpoints: Kontrolní bod. Checkpoint je snapshot aktuálního stavu workflow (včetně aktuálních hodnot proměnných a jakéhokoli výstupu generovaného do tohoto bodu) uložený na disk.

Checkpointing: Proces ukládání checkpoints. Jednotlivé checkpoints ukládají stav workflow a data na disk v průběhu workflow. Pokud bude workflow přerušený, není potřeba ho restartovat, ale stačí ho obnovit z posledního checkpoint.

CIM: Comon Information Model. Sada standardů, popisujících znázornění informací o správě software. CIM je rozšiřitelný, objektově orientovaný datový model, který obsahuje informace o různých částech podniku. Další informace o modelu CIM najdete v článku **Model CIM (Common Information Model) na MSDN**.

CLI: Command Line Interface. Rozhraní příkazové řádky.

Cmdlet, cmdlets: Zkratka pro command let(s). Nativní binární interní příkazy. Jsou to jednoduché, jednofunkční vestavěné příkazy WPS/PS7. Lze je rozšiřovat pomocí vlastní i pomocí třetích stran.

Command mode: Příkazový režim. Výstupem u WPS/PS7 je objekt (u ostatních shellů text).

Command pane: Okno pro příkazovou řádku WPS v ISE.

Conceptual topic: Pojmová témata. Začínají slovem `about_`.

DCOM: Distributed Component Object Model. Získávání informací pomocí vzdáleného volání procedur.

Dot Sourcing: Potřebujete-li zavést funkci nebo chcete-li po dokončení nějakého skriptu mít přístupné jeho proměnné (proměnné ve skriptu existují jen po dobu jeho běhu), pak použijte dot sourcing: spusťte skript pomocí tečky (následované mezerou) na začátku, např: `. .\mujskript.ps1`.

Expression mode: Výrazový režim. Výstup u WPS/PS7 je hodnota.

GUI: Graphical User Interface. Grafické uživatelské rozhraní.

Hostitel: Ve významu rozhraní WPS/PS7. Každá verze Powershellu má svoje vlastní rozhraní, a navíc každá konzole či ISE také.

ISE: Integrated Scripting Environment. Integrované skriptovací prostředí.

Item: Položka. Zde ve významu souboru, složky, klíče registru.

Managed node: Role, funkce, nastavení nebo fyzické či virtuální zařízení, na kterém se **provádějí** activities daného workflow.

No-terminating error: Neukončující chyby, které znázorní chybovou hlášku, ale pokračují dále ve vykonávání.

Node, nodes: Obecné označení pro stanici či server.

Noun: Podstatné jméno – vždy v jednotném čísle! Určuje objekt, se kterým se má provést akce. Např.: Command, Computer, Error, Event, Help, History, Host, Item, Location, Member, Object, Output, Path, Printer, Process, Service, Variable, Warning.

Persistence: Jiné označení pro checkpointing.

Provider: Zprostředkovatel, poskytovatel.

Příkaz, příkazy: Cmdlets, funkce, alias, příkaz DOSu atp.

PS1: Přípona pro skriptovací soubory WPS/PS7. Platí pro všechny verze WPS. Pokud spouštíte soubor PS1, musíte ho spouštět s uvedenou plnou cestou!

Relace: Viz Runspace.

Remote session: Viz Session.

Remoting: Ovládání vzdáleného počítače.

Runspace, runspaces: Běhový prostor workflow, taky nazýván jako relace či runtime. Samotné workflow je spuštěno ve svém runspace, jehož proměnné jsou dostupné pro všechny activities workflow. Samotné activities, příkazy a skripty běží každý ve svém runspace, jehož proměnné nejsou k dispozici v rámci workflow.

Runtime: Viz Runspace.

Rutina: Jsou to cmdlets, funkce, příkazy, výrazy atp.

Security Police: Bezpečnostní politika.

Session, sessions: Sezení, uzavřené prostředí, ve kterém můžete pracovat se vzdáleným počítačem – vytvoří pracovní prostor oddělený od vašeho pracovního prostředí na místním počítači.

Script pane: Okno pro editaci skriptu v ISE.

Shell: Nadstavba CLI. Například: CMD.EXE, SH, KSH, CSH, BASH. Vykonává příkaz nebo utilitu v novém procesu a zobrazuje uživateli výsledek v textovém režimu. Shelly mají vestavěny i svoje vlastní příkazy.

Snap-in: Primární typ nástroje správy, který poskytuje soubor funkcionality vyžadovaný ke správě technologie nebo aplikace z MMC konzole. Snap-in moduly se vytváří pomocí Visual Studia.

Snapshot, snapshots: Snímek čili „otisk" prostředí v definovaném okamžiku.

Snippet, snippets: Výstřižek kódu. Vkládá do kódu předdefinované šablony kódu, které jen doplníte o dané parametry.

Splatting: Použití hash tabulky místo parametrů (viz `about_splatting`).

Tab-Completion: Dokončení příkazu klávesou Tab. Lze používat u příkazů, parametrů, vlastností, metod, jmen souborů. Používá technologii Intellisense.

Terminating error: Ukončující chyby, které zastaví vykonávání příkazu.

Token: Známka, symbol (prostě pešek).

Typy příkazů: Alias, Application, Cmdlet, Function, Scripts, External Script, Workflow. Pokud je v tomto dokumentu uvedeno slovo příkaz, pak to může být jakýkoli typ příkazu zde uvedený. Jinak uvádím konkrétní typ příkazu (tedy např. Alias).

Uživatel: Kontext přihlášeného uživatele.

Verb: Slovesa. Základní akce, které určují, co s daným objektem chcete provádět:
Add, Clear, Compare, Complete, Connect, ConvertTo, ConvertFrom, Copy, Debug, Disable, Disconnect, Enable, Enter, Exit, Export, ForEach, Format, Get, Group, Help, Import, Invoke, Join, Limit, Measure, Move, New, Out, Pop, Push, Read, Receive, Register, Remove, Rename, Reset, Resolve, Restart, Restore, Resume, Select, Send, Set, Show, Sort, Split, Start, Stop, Tee, Test, Trace, Undo, Unregister, Update, Use, Wait, Where, Write. Doporučená slovesa získáte pomocí cmdlet `Get-Verb`.

WHS: Windows Script Host. Jazykově nezávislý skriptovací stroj („předchůdce" WPS). Ale ve WPS/PS7 ho můžete nadále používat.

Wildcard, wildcards: Divoké znaky (žolíky). Nahrazují výskyt libovolných znaků.

WMI: Windows Management Instrumentation. Hlavní technologie pro administraci Windows – získává široké rozmezí informací o počítači v jednotném formátu.

WMF: Windows Management Framework.

Workflow, workflows: Pracovní postup. Session, během které můžete spouštět paralelně a/nebo sekvenčně úlohy na více počítačích současně. Skládá se z jedné nebo více activities prováděných v uvedeném pořadí. Workflow se dá použít jako activity v jiném workflow.

Workflow client: Node, na kterém správce sleduje stav workflow a stará se o něj. Viz též Manage node.

Workflow server: Node, na kterém **běží** workflow (node, na kterém je relace!). Viz též Manage node.

WSMAN: Web Services Management. Nahrazuje DCOM. Potřebuje, aby na počítači byla spuštěna služba WinRM. Protože používá protokol http, není omezen firewally a směrovači. Avšak na rozdíl od DCOM, musíte pro každý požadavek (vyvolání metody, zjištění vlastností) spustit další požadavek.

WQL: **Dotazovací jazyk pro WMI.**

1.4. Příkazy nejsou textově založené

I když příkazy pro WPS/PS7 jsou textové, jejich výstupy jsou objekty na rozdíl od klasických shellů příkazové řádky. A to je rozdíl, který převrací způsob uvažování a používání!

Na rozdíl od tradičních CLI, které jsou textově založené (jejich výstup je textový řetězec), WPS/PS7 cmdlets jsou navrženy tak, že pracují s objekty založenými na platformě .NET. Tomu, kdo s touto platformou pracuje se otevírají velké možnosti použití WPS/PS7. Pro toho, kdo neví nic o objektech stručně vysvětlím:

Objekty jsou strukturovanými informacemi. Objekty mají metody (akce, které můžete s objektem provádět) a vlastnosti (charakteristiky objektu) a události. Technicky se to dá vyjádřit tak, že .NET objekt je instance (výskyt) .NET třídy, jež sestává z dat a operací sdružených s těmito daty. Všechny objekty stejného typu mají stejné metody, vlastnosti a události, ale vlastnosti se u každého objektu liší svou hodnotou. Pomocí cmdlet `Get-Member` můžete zjistit vlastnosti, metody a události daného objektu.

Příklad 1

Pomocí příkazu:

```
Get-Service | Get-Member
```

získáte metody a vlastnosti společné všem službám. Pomocí příkazu:

```
Get-Service ClipSrv | Get-Member
```

získáte výpis všech metod a vlastností pro reálný a konkrétní objekt (v tomto případě síťovou schránku). Všechny služby mají stejnou vlastnost Name, ale ta má u každého konkrétního objektu jinou hodnotu.

Protože výstup příkazu je objekt, vždy nese další informace, které můžete použít, pokud je budete potřebovat. Pokud používáte WPS/PS7, můžete použít i staré textově založené nástroje (čili můžete používat příkazy, které jste používali dříve). Ve většině případů však zjistíte, že jdou nahradit pomocí příkazů standardního WPS/PS7.

1.5. Rodina příkazů je rozšiřitelná

CLI (jako například Cmd.exe) neposkytují způsob, jak můžete přímo rozšířit jeho příkazovou sadu o další příkazy. Můžete vytvořit externí nástroje příkazové řádky, které fungují v cmd.exe, ale tyto externí nástroje nemají služby jako je integrace nápovědy a Cmd.exe automaticky neví, zda jsou to platné příkazy.

Nativní binární interní příkazy v systému WPS/PS7, které jsou známy jako cmdlets, mohou být rozšířeny o další cmdlets, které si vytvoříte sami a které přidáte do WPSu pomocí snap-in modulů. WPS/PS7 snap-in moduly jsou kompilovány tak, jako binární nástroje v jakémkoli jiném CLI. WPS/PS7 obsahuje velkou sadu cmdlets s konzistentním rozhraním.

Pomocí WPS/PS7 může spouštět i jiné příkazy než cmdlets. WPS/PS7 podporuje skripty, které jsou obdobou skriptů shellu UNIX a dávkových souborů Cmd.exe, ale mají příponu PS1. WPS/PS7 také umožňuje vytvářet interní funkce, které mohou být použity přímo v rozhraní příkazového řádku nebo ve skriptech.

WPS/PS7 je snadno rozšiřitelný – lze jej rozšířit o cmdlets pomocí knihoven a modulů. Pro aplikace typu Exchange, SharePoint, SQL Server, Vmware a další existují samostatné moduly, které pouze naimportujete a pracujete úplně stejně jako se základem WPS/PS7.

1.6. Ovládání vstupu konzole a zobrazování ve WPS/PS7

Když napíšete příkaz, tak WPS/PS7 zpracuje příkaz přímo a také formátuje výstup, který vidíte na obrazovce. To je významné, protože to snižuje práce nutné pro každý cmdlet a zajišťuje, že můžete vždy dělat věci stejným způsobem bez ohledu na to, který cmdlet používáte. Jedním z příkladů tohoto zjednodušení života pro vývojáře a uživatele je nápověda příkazového řádku: Tradiční nástroje příkazového řádku mají svá vlastní schémata pro požadavek o pomoc. Některé používají parametr /? na spuštění nápovědy, jiné používají parametry -?, / H, nebo dokonce / /. Některé zobrazují nápovědu v GUI okně, jiné v obrazovce konzole. Některé složité nástroje, jako jsou aktualizační aplikace, rozbalují interní soubory před zobrazením jejich nápovědy. Pokud použijete nesprávný parametr, nástroj může ignorovat to, co jste napsali a začít plnit úkol něco co nechcete...

Když zadáte příkaz ve WPS/PS7, vždy je automaticky analyzován a předzpracován systémem WPS/PS7. Pokud používáte parametr -?, tak v cmdlet WPS/PS7 to vždycky znamená, "zobraz nápovědu pro tento příkaz". Vývojáři cmdlets tak nemusí analyzovat příkazy, ale pouze obstarat nápovědu.

Je důležité vědět, že vlastnosti nápovědy jsou dostupné i při spuštění nástrojů tradičního příkazového řádku v systému WPS/PS7. WPS/PS7 zpracuje parametry a předá výsledky externím nástrojům.

Ve WPS/PS7 existují dva typy nápověd: ta první je vestavěná přímo v shellu a pak tematická k jednotlivým částem WPS/PS7. Tematická nápověda začíná slovem `about_<tema>`. K oběma se dostanete pomocí příkazu `Get-Help` a tematickou nápovědu si můžete také otevírat rovnou v poznámkovém bloku – je umístěna ve formě textového souboru v podadresáři en-us umístěného v adresáři WPS/PS7.

1.7. WPS/PS7 používá C# syntaxe

Syntaxe funkcí a klíčových slov je podobná programovacímu jazyce C#, protože WPS/PS7 je založen na platformě .NET Framework. Rozvoj nadstandardních znalostí ve WPS/PS7 je mnohem jednodušší, ovládáte-li jazyk C# a .NET Framework.

1.8. Znalosti WPS/PS7 jmen

Znalost jmen příkazů a parametrů je významnou časovou investicí ve většině CLI. Takže jediný způsob, jak se učit, je zapamatování každého příkazu a každého parametru, který budete pravidelně používat. Pracujete-li s novým příkazem nebo parametrem, nelze obecně použít to, co již víte, musíte se naučit novým příkazům či parametrům.

Většina příkazů ve většině CLI je vystavěna tak, aby spravovala prvky operačního systému nebo aplikace, jako jsou služby nebo procesy. V CLI existují příkazy, které dělají podobnou věc, ale jinak se jmenují a používají jinou šablonu parametrů, i když dělají podobnou věc (net start, net stop a proti tomu například příkaz sc). A tomu WPS/PS7 učinil konec svým jednotným a logickým způsobem tvorby příkazů.

1.9. Používání programů Windows

Ve WPS/PS7 můžete spouštět libovolné programy. Ale pokud není program v cestě PATH, tak ho musíte uvést plnou cestou. Pokud budete spouštět grafické aplikace v systému WPS/PS7, tak ten otevře aplikační okno. WPS/PS7 nemá vliv na to, jak aplikace pracuje interně, pouze umožňuje předat aplikaci parametry a umožňuje převzít výstup GUI aplikace do okna konzole.

1.10. Důležitá doporučení

Pokud začínáte pracovat s WPS/PS7, tak, obzvláště, pokud jste pracovali s klasickými shelly, budete chvíli plní zmatků, protože koncepce a způsob uvažováni a vlastní práce je někdy více, někdy méně, odlišná od vašich původních zkušeností. Asi nejhorší to bude pro ty z vás, kteří jste pracovali se starým typem shellu, neumíte programovací jazyk C# a neumíte pracovat s .NET Frameworkem.

Řekl bych, že nejdůležitější je si uvědomit si následující:

- Především musíte vědět, co chcete dělat (vývojové diagramy vám mohou pomoci) a rozdělit si celý problém na elementární akce.
- Naučte se používat nápovědu - je v ní opravdu všechno podstatné!

- Je třeba mít alespoň přehled o základních cmdlets (Get-Command, Get-Verb, Get-Member, Get-ChildItem, Where-Object, Select-Object atp.) a vědět o základních skupinách cmdlets (Format-* a Out-* pro výstup, *-PSDrive pro práci s jednotkami, *-Item pro práci s položkami atp.). Také je třeba si pamatovat několik důležitých parametrů (Verb, Noun, Syntax, ?, atp.). Potřebujete-li najít cmdlet, který dělá určitou akci pomůže vám právě jednotnost schématu používání jmen Verb-Noun (všechna povolená slovesa zjistíte pomocí Get-Verb): je tedy důležité si pamatovat alespoň pár základních sloves (Verb): Add, Clear, Compare, Complete, Connect, ConvertTo, ConvertFrom, Copy, Debug, Disable, Disconnect, Enable, Enter, Exit, Export, ForEach, Format, Get, Group, Help, Import, Invoke, Join, Limit, Measure, Move, New, Out, Pop, Push, Read, Receive, Register, Remove, Rename, Reset, Resolve, Restart, Restore, Resume, Select, Send, Set, Show, Sort, Split, Start, Stop, Tee, Test, Trace, Undo, Unregister, Update, Use, Wait, Where, Write a podstatných jmen (noun): Command, Computer, Error, Event, Help, History, Host, Item, Location, Member, Object, Output, Path, Printer, Process, Service, Variable, Warning, abyste si dokázali vydedukovat potřebný cmdlets. Nejdůležitější příkazy, které potřebujete ke své práci a bez kterých se neobejdete, jsou:
    - `Get-Help` – od verze WPS 3 je k dispozici cmdlet Show-Command, který usnadňuje práci s cmdlety a jejich vyhledávání a nápovědu.
    - `Get-Command` pro přehled příkazů dle různých podmínek.
    - `Get-Verb` pro získání povolených sloves.
    - `Get-Member` nebo `Format-List` pro získání vlastností.
- Používejte při ladění parametr WhatIf a Confirm!
- Uvědomte si, že vše jsou objekty mající vlastnosti a metody! Čili výstup jednoho příkazu v rouře vstupuje do dalšího příkazu jako objekt se svými vlastnostmi a metodami (výjimkou jsou cmdlets Out-*, jejichž výstup je text). Důležité je vědět, jaký typ objektu je akceptován příkazem na druhé straně roury a jaké jsou implicitní Parameter Set – a to si zjistíte právě z nápovědy.
- WPS/PS7 používá PSDrive jednotky pro disky, proměnné, certifikáty, aliasy, funkce, AD atp.
- Každý příkaz (kromě cmdlets Out-*) vrací pole objektů (pouze v expression mode je vrácena hodnota).
- Cmdlet `Tee-Object` posílá výstup zároveň do souboru a do roury.
- Naučte se používat Tab Expansion – velice to usnadňuje práci.
- Ve WPS/PS7 můžete spouštět libovolné programy. Ale pokud není program v cestě PATH, tak ho musíte uvést plnou cestou. Pokud spouštíte soubor PS1, musíte ho vždy spouštět s uvedenou plnou cestou!
- Doporučuji používat aliasy při práci v konzoli, avšak přímo ve skriptech používejte plná jména příkazů – za prvé tím dosáhnete přehlednosti, srozumitelnosti a pochopitelnosti (a to nejen pro sebe), ale zaručíte i plnou kompatibilitu skriptů (vaše vlastní aliasy si můžete definovat v profilu a ten se může snadno ztratit či být přepsán).
- Dávejte si pozor na formáty data, času a oddělovače desetinných míst. Funkce WPS/PS7 pracují s anglickými formáty!
- Pokud chcete zaznamenat vaše příkazy a jejich výstupy do souboru použijte Start-Transcript a Stop-Transcript (ve verzi WPS 5.0 a 5.1 lze použít i v ISE). Nelze použít v PS7!
- Chcete-li po dokončení nějakého skriptu mít přístupné jeho proměnné (proměnné ve skriptu existují jen po dobu jeho běhu), pak použijte dot sourcing: spusťte skript pomocí tečky (následované mezerou) na začátku, např: `. .\mujskript.ps1`.
- Pokud se chcete dotázat na vstup uživatele, tak promptem, např.: `$a = Read-Host -Prompt "Vložte vaše jméno:"`

2. INSTALACE, SPUŠTĚNÍ A OVLÁDÁNÍ WPS A PS7

2.1. Upozornění

- Chcete-li provést bezobslužnou instalaci WPS ze sítě u Windows XP, zadejte: `<PowerShell-exe-file-name> /quiet`. Ve Windows XP SP2 lze nainstalovat pouze WPS 2.0.
- Od Windows Server 2008 R2 a od Windows 7 je již WPS zabudován v systému. Ve Windows Server 2008 je WPS dostupný jako volitelný doplněk a můžete jej doinstalovat přes Server Manager.
- PS7 není implicitně nainstalován na Windows 10 a vyšších a Windows Server 2016 a vyšších.
- U WPS verze 2 až 5.1 (u verze 1 nic takového není třeba) je potřeba spustit následující příkaz v prostředí WPS, abyste mohli provádět Remote Scripting: `Set-ExecutionPolicy RemoteSigned`

2.2. Instalace

2.2.a) WPS

Instalace Windows Powershell ve Windows - uvedený odkaz uvádí jaké verze WPS jsou na kterých OS implicitně nainstalovány.

2.2.b) PS7

Instalace Poweshell ve Windows - uvedený odkaz uvádí všechny způsoby instalace PS7.

2.3. Umístění spustitelného souboru

2.3.a) WPS verze 5.1 a nižší (spouští se příkazem powershell.exe)

Na 32bitových verzích Windows je WPS implicitně instalován do adresáře:
%SystemRoot%\System32\WindowsPowerShell\v1.0.

Na 64bitových verzích Windows je 32-bit verze WPS instalována v adresáři:
%SystemRoot%\SystemWow64\WindowsPowerShell\v1.0.

Na 64bitových verzích Windows je 64-bit verze WPS instalována v adresáři:
%SystemRoot%\System32\WindowsPowerShell\v1.0.

Umístění modulů:

Powershell moduly	$env:WINDIR\system32\WindowsPowerShell\v1.0\Modules.
Moduly uživatele AllUsers	$env:ProgramFiles\WindowsPowerShell\Modules
Moduly uživatele CurrentUser	$HOME\Documents\WindowsPowerShell\Modules

2.3.b) PS7 (spouští se příkazem pwsh.exe)

Nainstalován do adresáře:

"C:\Program Files\PowerShell\“ a v podadresáři je příslušná verze PS7 (každá verze má svůj podadresář).

Umístění modulů:

Powershell moduly	$PSHOME\Modules.
Moduly uživatele AllUsers	$env:ProgramFiles\PowerShell\Modules.
Moduly uživatele CurrentUser	$HOME\Documents\PowerShell\Modules.

2.4. Zjištění nainstalované verze WPS/PS7

Následující body můžete nahradit příkazem výpisem proměnné $host nebo $PSVersionTable v konzoli WPS/PS7.

2.4.a) Kontrola, zda je vůbec WPS instalován

Key Location: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\PowerShell\1
Value Name: Install
Value Type: REG_DWORD
Value Data: 0x00000001 (1)

2.4.b) Kontrola na instalovanou verzi WPS:

Key Location: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\PowerShell\<1 nebo 2 nebo 3>\PowerShellEngine
Value Name: PowerShellVersion
Value Type: REG_SZ
Value Data: <1.0 | 2.0 | 3.0 | 4.0 | 5.0>

2.5. Spuštění a ovládání WPS/PS7 pomocí konzole

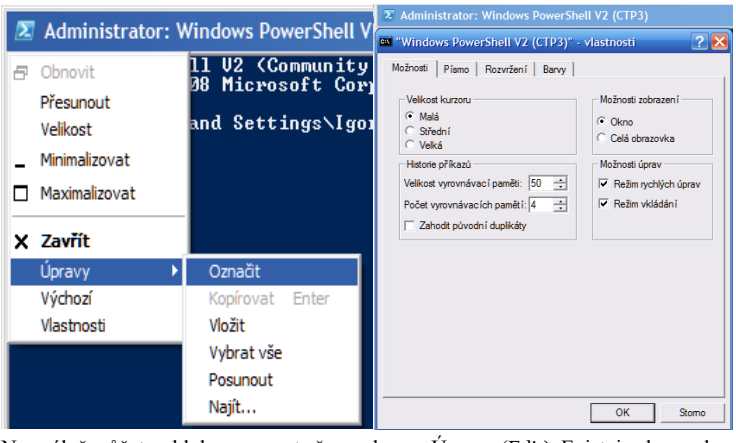
```
about_*powershell*
```

Pokud zadáte do příkazového řádku command konzole `powershell -?` nebo `pwsh -?` znázorní se vám parametry příkazového řádku.

2.5.a) Ovládání

Klávesa	Popis
Šipka nahoru/ Šipka dolů	Vyvolání předchozího/následujícího odeslaného příkazu.
F7	Vyvolání seznamu všech předchozích příkazů – historie. Od Windows 10 nefunguje.
F8	U verze WPS 2.0 pokud označíte v ISE řádek, pak se spustí jen tento řádek. Od verze WPS 3 stačí jen umístit kurzor do řádku.
TAB	Kontextové doplňování příkazů, vlastností, metod a/nebo cest k souborům. Nemusíte tedy zadávat celá jména souborů, stačí začátek. Například CD Prog <TAB> se automaticky doplní na CD 'Program Files'. Každá nadstavba příkazové řádky používá nějaký způsob pro zjednodušení zadávání příkazů. U WPS je to pomocí klávesy Tab (obrácený směr Shift+Tab). Tab Expansion je vnitřní funkcí WPS. Protože tato funkce může být modifikována nebo přepsána je v této příručce uváděno její implicitní chování. Dávejte si však pozor při kopírování do WPS/PS7! Uvědomte si, že klávesa Tab je zde pokládána za funkci rozšíření slov, kdežto v textu je používána jako oddělovač textu! Pokud při vyplňování jména souboru nebo cesty, parametru, metod, vlastnosti a události zadáte část jména a stisknete Tab, pak WPS/PS7 automaticky doplní jméno na první výsledek, který nalezne. Opakovaným použitím klávesy Tab cyklujete přes všechny dostupné výsledky. Při použití klávesy Tab u cmdlets je použití podobné: zadejte jeden či více znaků první části cmdletu (sloveso) a klávesou Tab cyklujte pro správné sloveso; pak zadejte pomlčku; zadejte jeden čí více znaků a cyklujte klávesou Tab. Totéž můžete použít u parametrů cmdlets. Klávesu Tab můžete použít na jednom řádku víckrát například jednou pro výběr cmdletu a podruhé pro vyhledání souboru, který je jeho parametrem a potřetí pro jméno parametru. Tab Expansion usnadňuje a zrychluje psaní, ale také snižuje množství překlepů a chybovost při psaní.
ESC	Vymazání celého aktuálního řádku.
CTRL+B	Přerušení skriptu – od verze WPS 5.
Ctrl+End	Vymazání celého aktuálního řádku od kurzoru doprava (nefunguje v ISE). Jen v konzoli WPS.
Ctrl+Home	Vymazání celého aktuálního řádku před kurzorem (nefunguje v ISE). Jen v konzoli WPS.
Ctrl+C	Stopnutí aktuálně běžícího programu.
CTRL+I	Ve WPS v ISE přepne do script pane.
CTRL+D	Ve WPS v ISE přepne do command pane.
Shift+ENTER	Odřádkování ve WPS v ISE editoru, aniž by to bylo bráno jako nová řádka.
#<znaky>	Hledání v historii příkazů - <znaky> jsou libovolné znaky, které příkaz obsahoval. Listujete pomocí TAB.
CTRL+Space	Ve WPS v ISE otevře intellisense menu.

V konzoli WPS/PS7 (ne v ISE) je možné používat i kopírování bloků z/do schránky Windows. Jsou dvě možnosti, obě využívají menu okna:



Normálně můžete s bloky pracovat přes podmenu Úpravy (Edit). Existuje ale mnohem pohodlnější řešení. Ve volbě menu Vlastnosti (Properties) se dá zaškrtnout položka Režim rychlých úprav (QuickEdit mode), přesně podle obrázku:

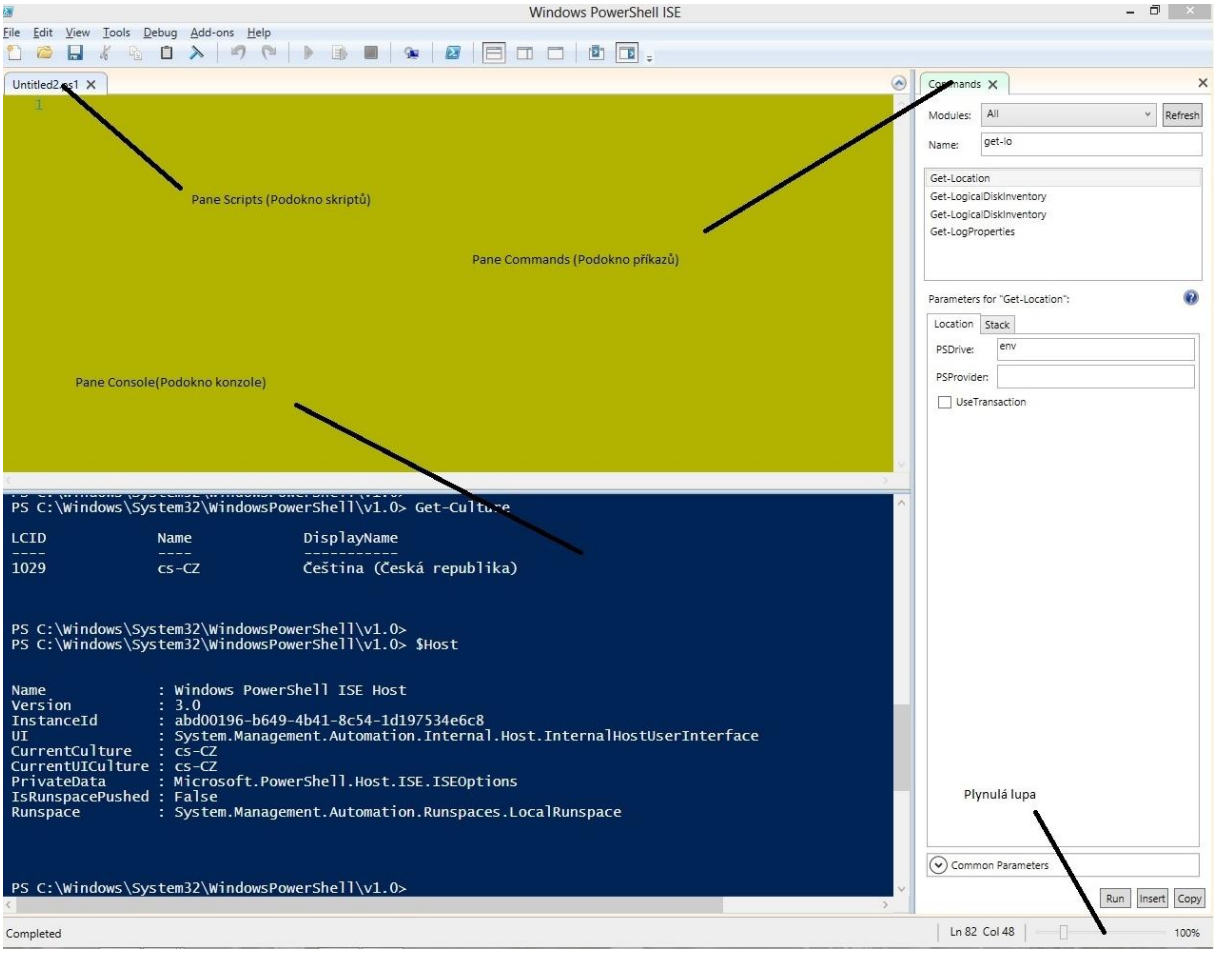
Potom máte k dispozici následující řízení bloků pomocí myši:

Tažení myší	Označuje bloky. Můžete vybrat i více řádků.
Pravé tlačítko	Při označeném bloku ho přenese do schránky. Pokud nic označeného nemáte, vloží aktuální obsah schránky na pozici kurzoru.

2.6. Spuštění a ovládání WPS pomocí ISE

```
about_*ise*
```

Od verze WPS 2 do verze WPS 5.1 je dodáván modul powershell_ise.exe. Je to jednoduchý, ale praktický editor kódu. Od verze WPS 3 je okno příkazového řádku a výstupu sloučeno do jednoho, přibyla intellisense a podokno příkazů. ISE editor se stále více přibližuje Visual Studiu. Je to natolik komfortní prostředí, že jen málokdy je třeba použít konzole. Je možné přidávat add-ons. V PS7 již ISE není a nahradil ho Visual Studio Code editor.



2.7. Bezplatná IDE pro WPS do verze 5.1

Powershell Plus – **https://www.idera.com/productssolutions/freetools/powershellplus** rozšiřuje funkčnost WPS (neplatí pro PS7) a hlavně usnadňuje práci s vytvářením a používáním skriptů.

2.8. Náhrada IDE pro PS7

V PS7 již sice není IDE, ale docela dobře ho nahrazuje Visual Studio Code - **https://code.visualstudio.com/**.

3. PROFILY A BEZPEČNOSTNÍ POLITIKA

about_profiles

Přidáváte-li aliasy, funkce a proměnné do WPS/PS7, pak platí jen pro danou session. Pokud ukončíte WPS/PS7, jsou tyto změny ztraceny. Chcete-li takovéto změny používat při každém spuštění WPS/PS7, pak stačí je umístit do profilu, odkud se automaticky při startu WPS/PS7 natáhnou. Profil je natahován při každém spuštění WPS/PS7. K tomuto musí být nastavena práva (execution policy) jinak nedojde k natažení a zobrazí se chyba.

Profily můžete vytvářet, sdílet a distribuovat. A nevytvářejí se automaticky. Dobře vytvořený profil usnadňuje administraci a používání WPS/PS7.

ISE profily pro WPS se rozlišují pomocí slova ISE ve jménu profilu.

3.1. WPS profily

3.1.a) Konzolové profily

Profil (od nejnižší priority k nejvyšší)	Tento profil je aplikován na…
$PSHOME\profile.ps1	na všechny uživatele a všechny hostitele.
$PSHOME\Microsoft.PowerShell_profile.ps1	všechny uživatele, běžného hostitele.
$HOME\Documents\WindowsPowerShell\profile.ps1	běžného uživatele a na všechny hostitele.
$HOME\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1	běžného uživatele a běžného hostitele.

3.1.b) ISE profily

Profil (od nejnižší priority k nejvyšší)	Tento profil je aplikován na…
$PSHOME\Microsoft.PowerShellISE_profile.ps1	všechny uživatele, běžného hostitele.
$HOME\Documents\WindowsPowerShell\Microsoft.PowerShellISE_profile.ps1	běžného uživatele a běžného hostitele.

3.2. PS7 profily

Profil (od nejnižší priority k nejvyšší)	Tento profil je aplikován na…
$PSHOME\profile.ps1	na všechny uživatele a všechny hostitele.
$PSHOME\Microsoft.PowerShell_profile.ps1	všechny uživatele, běžného hostitele.
$HOME\Documents\PowerShell\profile.ps1	běžného uživatele a na všechny hostitele.
$HOME\Documents\PowerShell\Microsoft.PowerShell_profile.ps1	běžného uživatele a běžného hostitele.

3.3. Vytvoření profilu

Pokud potřebujete zjistit, zda je profil již vytvořen zadejte příkaz:

```
Test-Path $profile (True pokud již existuje, jinak False)
```

Profile můžete vytvořit buď pomocí libovolného textového editoru nebo pomocí příkazu např.:

```
New-Item -Path $env:windir\System32\WindowsPowerShell\v1.0\profile.ps1 -ItemType File -
Force
```

3.4. Alternativa úpravy profilu

about_PowerShell.exe.help

Pokud nechcete nebo nemůžete import nějakých modulů vložit do profilu, pak stačí spustit WPS/PS7 s příslušnými parametry v příkazové řádce, např.:

```
powershell.exe -noexit -command import-module HyperV
```

3.5. Execution Policy

about_signing

Skriptování je užitečná pomůcka, ale také velmi mocný nástroj, kterým se může narušit bezpečnost. Protože spouštění skriptů může zanést do počítače škodlivý kód, je ve WPS/PS7 bezpečnostní politika (security police), která určuje, zda skript může být spuštěn a zda musí obsahovat digitální podpis. Implicitně je bezpečnost nastavena vysoko, takže nemůžete například spustit skript poklepáním na jeho ikonu. WPS/PS7 umožňuje určit bezpečnostní politiku – zda skript lze spustit s podpisem, či bez podpisu, zda konfigurační soubory mohou být nataženy či ne. Bezpečnostní politika se ukládá v registry, kde zůstává i po odinstalaci či reinstalaci WPS/PS7. Po nainstalování je nastavena politika Restricted.

Možné hodnoty politiky jsou (od nejméně bezpečné k nejbezpečnější):

Unrestricted	povoluje spouštění všech skriptů.
RemoteSigned	vyžaduje digitální podpis u skriptů, které nepocházejí z lokálního počítače. Toto je doporučené nastavení.
AllSigned	všechny skripty, i z lokálního počítače, musejí být podepsané.
Restricted	je zakázáno spouštět všechny skripty, kromě “ručně zadaných” do WPS/PS7 konzole. Toto je výchozí nastavení. V tomto módu je umožněno zapisování příkazů přímo v okně PowerShellu, ale pokud chcete spustit skript, PowerShell vám v tom zabrání a zobrazí chybu.

Pro nalezení bezpečnostní politiky na vašem systému:

```
Get-ExecutionPolicy
```

a pro její změnu:

```
Set-ExecutionPolicy <hodnota>
```

Pro více informací:

```
Get-Help about_signing
```

Windows PowerShell – jak digitálně podepsat skript

3.6. Výchozí hodnoty parametrů

Občas se vám mohou hodit vlastní hodnoty implicitních parametrů. Například dosazujete vaše jméno serveru, skupiny apod. Můžete si tyto parametry nastavit přímo ve vaší session nebo také pomocí profilu.

Vzor pro vlastní hodnotu parametru:

```
PSDefaultParameterValue = @{"NameOfCommand:NameOfParameter” = “ValueOfParameter"}
```

Například chcete, aby implicitní hodnota parametru Name pro Get-Process byla "powershell"

```
$PSDefaultParameterValue = @{"Get-Process:Name"="powershell"}
```

Můžete použít i hvězdičkovou konvenci, např. pro všechny cmdlets pro AD:

```
PSDefaultParameterValue = @{"*-AD:Server"="MyDC"}
```

Chcete-li vypnout DefaultParameterValues bez ztráty vašeho nastavení:

```
$PSDefaultParameterValues = @{"Disabled" = $true}
```

4. POWERSHELL VERZE 7.X

- Verze WPS 5.1 je závislá na .NET Framework v4.5 a není tak multiplatformní, takže už pro ni budou vydávány pouze aktualizace zabezpečení.
- Od verze 6 to již není Windows Powershell, ale Powershell. Powershel 7.x nahrazuje verzi Powershell 6 (Powershell Core 2.0), takže o verzi 6 už se nebudu zmiňovat.
- PS7 je postaven na na .NET Core 3.1 (od verze 7.2 bude postaven na .NET 6.0), čímž je multiplatformní a je podporován v následujících operačních systémech Windows: Windows 8.1, 10 a 11 a Windows Server 2012, 2012 R2, 2016 a 2019 a dalších (macOS, Linux). Podrobnosti o podporovaných operačních systémech najdete na **Životní cyklus podpory PowerShellu**.
- PS7 není zatím implicitně instalován do operačních systémů!

4.1. Migrace na PS7

4.1.a) Instalace PS7

Existuje několik způsobů:

- Pomocí balíčku MSI.
- Pomocí balíčku ZIP.
- Přes Windows Store.
- Pomocí WinGet.

Podrobnosti k instalaci na **Install PowerShell on Windows, Linux, and macOS**.

4.2. Kompatibilita PS7 s WPS 5.1

```
about_Windows_PowerShell_Compatibility
```

Rozdíly mezi WPS 5.1 a PS7

4.2.a) Moduly

Většina modulů by měla být kompatibilní. Ale je dobře se o tom přesvědčit. Při importu modulů použijte s příkazem `Import-Module` přepínač UseWindowsPowerShell – usnadňuje to přechod nekompatibilních modulů.

V PS7 nejsou zahrnuty následující moduly:

- ISE.
- Microsoft.PowerShell.LocalAccounts.
- Microsoft.PowerShell.ODataUtils.
- Microsoft.PowerShell.Operation.Validation.
- PSScheduledJob.
- PSWorkflow a PSWorkflowUtility (kvůli nedostatečné podpoře Windows Workflow Foundation v .NET Core byl z PowerShellu odebrán Workflow – měl by být postupně nahrazen paralelismem).
- Vlastní snap-in.

Aktuální informace o kompatiblitě modulů najdete na **Kompatibilita modulů PowerShellu 7**. A podrobné informace o rozdílech (odebrané rutiny apod.) najdete na **Rozdíly mezi WPS 5.1 a PS7**.

4.3. Workflow

Byla odebrána podpora workflow!

4.4. Další podstatné rozdíly PS7 proti WPS 5.1

- Nejsou podporovány vlastní moduly snap-in.
- Není už podporován model DCOM, je podporován pouze model WSMAN.
- Lze spustit souběžně se starší verzí. Je totiž umístěn v jiném adresáři s jiným jménem $env:ProgramFiles\PowerShell\7.
- Má oddělenou PSModulePath (v $Env:PSModulePath).
- Má oddělené profily.
- Powershell verze 7.x již neobsahuje IDE a nebude podporováno – je možno ho v okleštěné formě nahradit **Visual Studio Code (VSCode)** s **PowerShell Extension**.
- Vzdálené komunikace založené na SSH (tam, kde nelze použít WinRM). Cmdlets `New-PSSession, Enter-PSSession a Invoke-Command` mají nový parametr pro SSH.
- Podpora pro kontejnery Dockeru.
- Vylepšená kompatibilita modulů.
- Nové koncové body vzdálené komunikace.
- Podpora Group Policy – podrobně v `about_Group_Policy_Settings`. Šablony a instalační skript pro Group policy je v $PSHOME.
- Oddělené Event logy – podrobně v `about_Logging_Windows`.
- Vrstva kompatibility, která umožňuje uživatelům importovat moduly v implicitní relaci prostředí WPS.
- Automatická oznámení o nové verzi. Podrobně na stránce **about_Update_Notifications**.

4.4.a) Změny v cmdlets a programování

Z různých důvodů kompatibility nebo použití nepodporovaných rozhraní API byly z PS7 odebrány cmdlets, jejichž seznam je na **Rozdíly mezi WPS 5.1 a PS7**. Nejdůležitější je, že byly odebrány `*-Transaction a *-EventLog` cmdlets.

Níže uvedené změny jsou podrobně vysvětleny v kapitole Programování.

- Nové ternární operátory: ? a : .
- Operátory potrubních řetězců || a && .
- Null operátory ?? a ??= .
- Cmdlet `Update-Help` již může být spuštěn v kontextu uživatele (bez administrátorského oprávnění). Nápověda je tak ukládána do složky Help uživatele.
- Cmdlet `ForEach-Object -Parallel`: Ve výchozím nastavení tento cmdlet používá pracovní adresář uživatele, který ho spustil. Parametr Paralell určuje blok skriptu, který je spuštěn paralelně pro každý vstup z roury. Podrobně na stránce o **ForEach-Object**.
- Nový cmdlet `Get-Error` pro snadnější zkoumání chyb poskytuje podrobný přehled o chybě. Podrobně na stránce **Get-Error**. S tímto cmdlet souvisí proměnná $ErrorView, která je implicitně nastavena na hodnotu ConciseView (ve WPS je implicitně nastavena na hodnotu NormalView). Zároveň byla přidána nová vlastnost ErrorAccentColor ($Host.PrivateData), která podporuje změnu barvy zvýraznění chybové zprávy.

5. SCHÉMA JMEN PŘÍKAZŮ

5.1. Úvod

Cmdlets používají následující schéma jmen:

```
verb-noun
```

WPS/PS7 slovesa nemusí být vždy anglická slovesa (verb), ale vyjádřují konkrétní akci v systému WPS/PS7. Seznam povolených sloves (pokud vyvíjíte své vlastní cmdlets, je důležité jej dodržovat) získáte pomocí Get-Verb. Podstatná jména (noun) popisují konkrétní typy objektů ke kterému se sloveso vztahuje. Názvy podstatných jmen můžou obsahovat jenom písmena, číslice, pomlčky (-) a podtržítka (_). Vyhněte se však raději názvům s podtržítky, protože ty můžou být pro uživatele matoucí (nemusí být totiž zřetelná). Podstatná jména se používají v jednotném tvaru.

Například v případě dvou podstatných jmen a dvou sloves se učení moc nezjednoduší, ovšem při sadě 10 sloves a 10 podstatných jmen se musíte naučit pouhých 20 slov, která však mohou vytvořit 100 různých jmen příkazů. Většinou je zřejmé, jaké sloveso a podstatné jméno použít pro požadovanou funkci např: vypnutí počítače příkazem `Stop-Computer`; získání seznamu všech počítačů v síti příkazem `Get-Computer`; zjištění systémového data příkazem `Get-Date`.

Můžete si třeba zobrazit seznam všech příkazů, zahrnujících sloveso Get:

```
PS> Get-Command -Verb Get
CommandType     Name                                Definition
-----------     ----                                ----------
Cmdlet          Get-Acl                             Get-Acl [[-Path] <String[]>]...
Cmdlet          Get-Alias                           Get-Alias [[-Name] <String[]...
Cmdlet          Get-AuthenticodeSignature           Get-AuthenticodeSignature [-...
Cmdlet          Get-ChildItem                       Get-ChildItem [[-Path] <Stri...
...
```

Podstatné jméno je ještě více užitečné, protože vám umožňuje zjistit skupinu příkazů, které se týkají stejného typu objektu. Například, pokud chcete znát příkazy, které jsou k dispozici pro správu služeb, zadejte následující příkaz:

```
PS> Get-Command -Noun Service
CommandType     Name                                Definition
-----------     ----                                ----------
Cmdlet          Get-Service                         Get-Service [[-Name] <String...
Cmdlet          New-Service                         New-Service [-Name] <String>...
Cmdlet          Restart-Service                     Restart-Service [-Name] <Str...
Cmdlet          Resume-Service                      Resume-Service [-Name] <Stri...
Cmdlet          Set-Service                         Set-Service [-Name] <String>...
Cmdlet          Start-Service                       Start-Service [-Name] <Strin...
Cmdlet          Stop-Service                        Stop-Service [-Name] <String...
Cmdlet          Suspend-Service                     Suspend-Service [-Name] <Str...
...
```

Příkaz nemusí být nezbytně cmdlet jen proto, že používá schéma <sloveso-podstatné jméno>. Například nativní WPS/PS7 příkaz Clear-Host není cmdlet, ale používá schéma <sloveso-podstatné jméno>. Clear-Host je vlastně vnitřní funkce, jak uvidíte, spustíte-li příkaz:

```
PS> Get-Command -Name Clear-Host
CommandType     Name                                Definition
-----------     ----                                ----------
Function        Clear-Host                          $spaceType = [System.Managem...
```

5.2. Sumář informací o příkazech

Get-Command získává jména všech dostupných příkazů:

```
PS> Get-Command
CommandType     Name                                Definition
-----------     ----                                ----------
Cmdlet          Add-Content                         Add-Content [-Path] <String[...
Cmdlet          Add-History                         Add-History [[-InputObject] ...
Cmdlet          Add-Member                          Add-Member [-MemberType] <PS...
...
```

Tento výstup vypadá hodně podobně jako výstup nápovědy z Cmd.exe: tabulární shrnutí vnitřních příkazů. Je znázorněn typ příkazu (v tomto případě cmdlet). Ve výstupu `Get-Command` příkazu jsou všechny definice ukončeny třemi tečkami (...), které naznačují, že WPS/PS7 nemůže zobrazit celý obsah využitelného prostoru konzole. Když systém WPS/PS7 zobrazuje výstup, formátuje výstup jako text a pak ho umístí do okna konzole.

5.3. Zobrazení dostupných typů příkazů

Get-Command příkaz neuvádí všechny příkazy, které jsou k dispozici v systému WPS/PS7. Uvádí pouze nativní cmdlets ve WPS/PS7. WPS/PS7 podporuje několik dalších typů příkazů: Aliases, Functions a Scripts, což jsou také WPS/PS7 příkazy. Vnější soubory, které jsou vykonavatelné nebo mají registrovaný typ souboru, jsou rovněž klasifikovány jako příkazy.

Můžete získat seznam všech položek, které lze vyvolat zadáním následujícího příkazu:

```
PS> Get-Command *
```

Protože tento seznam zahrnuje externí soubory nalezené v cestě PATH může obsahovat tisíce položek. Je tedy lepší podívat se na omezenou sadu příkazů. Chcete-li najít nativní příkazy jiných typů než cmdlet, můžete použít parametr CommandType cmdletu `Get-Command`.

Chcete-li zobrazit speciální kategorii příkazů – aliases:

```
PS> Get-Command -CommandType Alias
```

Chcete-li zobrazit speciální kategorii příkazů – funkce:

```
PS> Get-Command -CommandType Function
```

Chcete-li zobrazit speciální kategorii příkazů – externí skripty:

```
PS> Get-Command -CommandType ExternalScript
```

5.4. Help – nápověda

```
Get-Help
Save-Help
Update-Help (nefunguje v ISE)
Help (funguje v ISE až od verze 4)
Man (funguje v ISE až od verze 4)
```

Nápověda existuje k příkazům a také k jednotlivým tématům. Pokud chcete nápovědou stránkovat nebo řádkovat, pak použijte příkaz Help nebo Man, který funguje stejně jako příkaz More v příkazové řádce: mezerníkem stránkujete a enterem řádkujete (nefunguje v ISE, pouze v konzoli).

Ve WPS/PS7 existují dva typy nápověd: ta první je vestavěná přímo v shellu a pak tematická k jednotlivým částem WPS/PS7. Tematická nápověda začíná slovem `about_<tema>`. K oběma se dostanete pomocí příkazu `Get-Help` a tematickou si můžete otevírat rovnou v poznámkovém bloku a je umístěna ve formě textového souboru v podadresáři en-us umístěného v adresáři WPS/PS7.

Pomocí přepínače –Online lze získat aktuální webovou nápovědu s opravou chyb. Cmdlet `Update-Help` si nápovědu zaktualizujete přímo u sebe na počítači. Dokonce můžete pomocí tohoto cmdletu na počítači, který nemá přístup k internetu nahrát nejnovější verzi helpu z adresáře, který jste si vytvořili pomocí cmdletu `Save-Help`.

Vřele doporučuji nápovědu – je v ní skoro všechno a je velice vyčerpávající. Nepodceňujte ji – pokud vám něco nejde, podívejte se nejprve do nápovědy.

5.4.a) Důležité parametry pro nápovědu

–Full

Plná nápověda i s příklady.

–Examples

Pouze příklady.

–Online

Online verze helpu (poslední verze).

5.4.b) Základní části vlastní nápovědy ke cmdletu

Name

Jméno cmdletu.

Synopsis

Stručný popis funkce cmdlet.

Syntax

Syntaxe cmdletu. Cmdlety mohou mít i více variant syntaxe. První varianta je implicitní.

Description

Podrobný popis fungování cmdletu.

Parameters (Parameter Set)

Pokud existuje více variant syntaxí pro cmdlet, pak první bývá implicitní. Vždy musíte použít pouze syntaxi daného parameter setu! Pokud WPS/PS7 nenajde implicitní hodnotu (například si neodpovídá typ objektu), zkouší další parameter set! V této sekci najdete přehled všech parametrů, který má následující formát:

Required?	False nebo True. Určuje, zda tento parametr musí být uveden.
Position?	Named (nebo číselné určení pozice). Určuje, zda tento parametr lze určit pozičně nebo musíte vypsat jméno parametru.
Default value	Implicitní hodnota.
Accept pipeline input?	True nebo False. Určuje, zda tento parametr přijímá vstup z roury. Popřípadě je uvedeno co přebírá (ByValue, ByPropertyName). ByValue předává objekt, ByPropertyName předává jméno vlastnosti.
Accept wildcard characters?	Určuje zda je možno použít wildcard.

Inputs

Udává s jakým typem objektu příkaz pracuje. Je to důležitý údaj v případě, že rourou předáváte objekt jako celek (ByValue) – musíte předat stejný typ objektu! Typ objektu si zjistíte pomocí `Get-Member`, který vám určí TypeName, což je vlastně typ objektu.

Outputs

Platí totéž co u Inputs, pouze se jedná o to, co je daným cmdletem předáváno dál rourou.

Notes

Poznámky nezařaditelné do žádné jiné sekce nápovědy. Například o tom, na kterých systémech a v které verzi WPS/PS7 lze cmdlet použít.

(Examples)

Příklady.

Related Links

Odkazy na online verzi helpu a další související cmdlety.

5.4.c) Příklady použití

```
Get-Help <hledaný výraz=téma>
```

Pokud například chcete vyhledat všechny příkazy, které pracují s logy:

```
Get-Help log
Name                                Category
----                                --------
```

```
Get-EventLog                        Cmdlet
Clear-EventLog                      Cmdlet
Write-EventLog                      Cmdlet
Limit-EventLog                      Cmdlet
Show-EventLog                       Cmdlet
New-EventLog                        Cmdlet
Remove-EventLog                     Cmdlet
about_eventlogs                     HelpFile
about_logical_operators             HelpFile
```

`Get-Help <jméno cmdlet>`

Vypíše základní nápovědu k danému cmdletu.

`Get-Help <about_téma>`

Vypíše obsah textového souboru s daným tématem.

`Get-Help Get-WinEvent -full`

Vypíše kompletní nápovědu k danému cmdletu.

5.5. Historie příkazů

`Get-History`

V příkazové řádce se pro historii používá klávesa F7 (nefunguje v ISE, pouze v konzoli). U WPS/PS7 se pro seznam již zadaných řádků používá příkaz Get-History. Pro vlastní práci a vyvolávání příkazů z historie se používá znak #:

```
PS> Get-History
PS> Id CommandLine
  -- -----------
   1 Get-Verb
PS> #erb
```

Pokud zadáte znak hash následovaný výrazem (bez úvodní mezery!), který chcete vyhledávat v historii, stačí tabelátorem listovat v seznamu řádků, ve kterých je uvedený výraz obsažen.

5.6. Parametry

`about_command_syntax`
`about_Parameters`

5.6.a) Doporučené názvy parametrů

Jádro WPS/PS7 používá standardní jména pro podobné parametry u všech cmdlets. Ačkoli použití standardního jména parametru jména není povinné, jsou doporučeny pokyny pro použití podpory standardizace jmen.

Například, při pojmenování parametru, který se vztahuje k počítači použijte obecně použitelná jména jako například pro jméno počítače použijte ComputerName – nepoužívejte Server, Host, System, Nod nebo jiná společná alternativní slova. Mezi důležité doporučené parametry jsou navržena jména jako jsou Force, Exclude, Include, PassThru, Path a CaseSensitive.

5.6.b) Standardní parametry cmdlets

Jak již bylo řečeno, příkazy používané v tradičním CLI obvykle nemají konzistentní jména parametrů. Některé příkazy nemají parametry vůbec. Pokud ano, tak jsou to často jednoznaková nebo zkrácená slova, která můžete sice psát rychle, ale jsou nesnadno pochopitelná pro nováčky.

WPS/PS7 zpracovává parametry přímo, a podporuje používání konzistentních jmen parametrů.

Jména parametrů mají vždy předponu (prefix) '-', aby je mohl WPS/PS7 jasně vymezit jako parametry. V `Get-Command -Name Clear-Host` je například parametr Name jméno parametru, a je zapsán jako -Name.

Jména parametrů nemusíte uvádět celá – stačí jen dostatečný počet znaků pro odlišení od ostatních jmen parametrů. Například místo parametru –detailed stačí uvést –det.

Některá jména parametrů jsou volitelná (jsou uváděna v hranatých závorkách) a tak pokud je vynecháte, musíte parametry uvádět v pozici, ve které se vyskytují v syntaxi. Například místo Get-Help –Name Get-Command stačí použít Get-Help Get-Command.

Jako hodnotu parametrů můžete uvést i funkci (funkce je uzavřena v kulatých závorkách) - například další cmdlet.

Zde jsou některé z obecných vlastností standardních jmen parametrů a jejich použití:

`Parametr Help (?)`

Upozornění: Ve verzi WPS 1 se znázorňuje úplný help pro každý příkaz a u verze WPS 2 pouze syntaxe příkazu bez jakéhokoliv popisu.

Když specifikujete parametr -? pro jakýkoliv cmdlet, pak cmdlet není vykonán, ale je zobrazena nápověda pro tento cmdlet. Např.:

`Get-Childitem -?`

Pro zobrazení syntaxe cmdletu get-help:

`Get-Help -?`

Pro získání nápovědy k jednotlivému příkazu např. get-childitem:

```
Get-Help Get-Childitem
Get-Help Get-Childitem -Detailed
Get-Help Get-Childitem -Full
Get-Help Get-Childitem -Examples
```

nebo pro zobrazení nápovědy po stránkách:

`man Get-Childitem`

či

`Help Get-Childitem`

Pro zobrazení informací o pojmových tématech (začínají slovem about_):

```
Get-Help about_*
```

5.6.c) Důležité parametry

?	Help, nápověda.
Autosize	Automaticky přizpůsobí velikost sloupců výstupu.
CommandType	Typ příkazu: Alias, Application, Cmdlet, Filter, Function, External Script, Script.
Exclude	Vyjmout.
ExpandProperty	Tento parametr vrátí jen hodnotu vlastnosti.
FilePath	Cesta k souboru. Jsou povoleny zástupné znaky.
Force	Vynucení. Podle kontextu příkazu.
GroupBy	Seskupit podle.
Include	Zahrnout.
ItemType	Typ položky: Directory, File.
LiteralPath	Cesta k souboru. Nelze použít zástupné znaky, používá se přesně tak, jak je zadána.
MemberType	Typ členu: Property, Method, Event.
Name	Jméno.
Noun	Podstatné jméno. Určuje objekt, se kterým pracujete.
Paging	Stránkuje výstup na konzoli. Nefunguje v ISE prostředí.
PassThru	Zobrazí výsledek příkazu, který normálně výsledek nezobrazuje.
Path	Cesta (klasická cesta k souboru používající jednotky PSDrive).
Property	Vlastnost. Pokud vlastnost objektu neexistuje je vrácena prázdná hodnota.
Recurse	Rekurze (bere v potaz i podsložky). Pokud použijete tento parametr v akci remove, nebudete dotázáni, zda chcete opravdu smazat.
SimpleMatch	Nebude použito regulárních výrazů.
Syntax	Syntaxe cmdletu.
Verb	Sloveso. Určuje akci, která se s objektem má provést.
Wrap	Zalomí text přesahující šířku řádku na další řádek.

5.6.d) Společné (obecné) parametry

```
about_commonparameters
```

WPS/PS7 má několik parametrů, známých jako společné parametry. Vzhledem k tomu, že tyto parametry jsou řízeny pomocí WPS/PS7, při jejich použití v jakémkoliv příkazu se vždy chovají stejným způsobem. Společné parametry jsou:

AsJob		Spustí cmdlet na pozadí.
Confirm	Confirm[:{$true | $false}]	Zeptat se na potvrzení operace.
Debug	Debug[:{$true | $false}]	Zobrazí programátorskou úroveň detailů o operaci. Stejné jako Verbose, ale ptá se na potvrzení každého kroku. Nastavuje preferenční proměnnou $DebugPreference. Implicitně True.
ErrorAction	ErrorAction[:{SilentlyContinue | Continue | Inquire | Stop)]	Určuje chování příkazu při non-terminating chybě.
ErrorVariable	ErrorVariable [+]<variable-name>	Ukládá chybu cmdletu do proměnné. Znaménko + určuje, že se chyba přidává do proměnné. Jméno proměnné se uvádí bez znaku $ (při dalším použití už samozřejmě ano), jinak se obsah uloží do obsahu proměnné! Výstup je pole!
Keep		Uschová výsledek cmdletu.
OutVariable	OutVariable [+]<variable-name>	Umisťuje výsledek cmdlets do proměnné. Znaménko + určuje, že se výsledek přidává do proměnné. Jméno proměnné se uvádí bez znaku $ (při dalším použití už samozřejmě ano), jinak se výsledek uloží do obsahu proměnné!. Výstup je pole!
OutBuffer	OutBuffer <Int32>	Určuje počet objektů v bufferu, při jehož dosažení se vyšle výsledek příkazu do roury.
Verbose	Verbose[:{$true | $false}]	Zobrazí/nezobrazí detailní výpis operace. Nastavuje preferenční proměnnou $VerbosePreference na Continue. Implicitně True.
WarningAction	WarningAction[:{SilentlyContinue | Continue | Inquire | Stop}]	Určuje chování příkazu při varování.
WarningVariable	WarningVariable [+]<variable-name>	Ukládá varování do proměnné. Znaménko + určuje, že se varování přidává do proměnné. Jméno proměnné se uvádí bez znaku $. Výstup je pole!
WhatIf	WhatIf[:{$true | $false}]	Zobrazuje zprávu, která popisuje výsledek příkazu, jako kdyby se vykonal. Implicitně True.

5.7. Používání známých jmen příkazů – aliasing

Pomocí mechanismu nazývaného aliasing, WPS/PS7 umožňuje uživatelům odkazovat na příkazy (cmdlets, funkce, exe soubory) jejich alternativními jmény. Aliasing umožňuje uživatelům se zkušenostmi v jiných nadstavbách používat jména příkazů, která již znají pro podobné operace v systému WPS/PS7. Aliasing slouží také k minimalizaci psaní. Aliasing sdružuje jména příkazů s jiným příkazem. Například WPS/PS7 má interní funkci s názvem `Clear-Host`, která vyčistí výstupní okno. Pokud zadáte příkaz cls nebo clear na příkazovém řádku, WPS/PS7 interpretuje tyto příkazy jako alias pro `Clear-Host` funkci a spustí ji. Tato vlastnost pomáhá uživatelům k v případě přechodu z jiného shellu.

Většina uživatelů Cmd.exe a UNIXu má zažitý velký repertoár příkazů, které již znají podle jména, a ačkoli ekvivalentní příkazy WPS/PS7 nemusí produkovat úplně stejné výsledky, jsou si natolik podobné, že uživatelé je mohou využívat k práci, aniž by se museli hned napoprvé učit jména příkazů WPS/PS7.

POZNÁMKA 2: Doporučuji používat aliasy při práci v konzoli, avšak přímo ve skriptech používejte plná jména příkazů – za prvé tím dosáhnete přehlednosti, srozumitelnosti a pochopitelnosti (a to nejen pro sebe), ale zaručíte i plnou kompatibilitu skriptů (vaše vlastní aliasy si můžete definovat v profilu a ten se může snadno ztratit či být přepsán).

Hlavním zdrojem frustrace je učení se nové nadstavbě, přičemž již uživatel je obeznámen s jinou nadstavbou – chyby jsou způsobeny "prstovou paměti". Například, pokud jste používali roky Cmd.exe, tak v okamžiku, kdy máte obrazovku plnou a chcete ji vyčistit, reflexivně zadáte příkaz cls a stisknete Enter ... Bez aliasingu byste dostali chybovou hlášku

```
" 'cls' is not recognized as a cmdlet, function, operable program, or script file.".
```

V následující tabulce je uveden stručný výčet společných Cmd.exe a UNIX příkazů, které můžete používat ve WPS/PS7:

cat	dir	Mount	rm
cd	echo	move	rmdir
chdir	erase	popd	sleep
clr	h	ps	sort
cls	history	pushd	tee
copy	kill	pwd	type
del	lp	r	write
diff	ls	ren	

Pokud se chcete dozvědět skutečná jména příkazů WPS/PS7, skrývajících se za aliasy můžete použít následující příkaz:

```
PS> Get-Alias
CommandType     Name                                Definition
-----------     ----                                ----------
Alias           cls                                 Clear-Host
```

Pro aliasy existuje speciální jednotka (podobná jednotkám souborového systému) alias: S touto jednotkou pracujete stejně jako s klasickými jednotkami souborového systému.

Příklad 2

Nastavení na jednotku alias: a získání aliasů

```
Set-Location alias:
Get-Childitem
```

nebo můžete nahradit jedním příkazem:

```
Get-Childitem alias:
```

5.7.a) Interpretování standardních aliasů

Na rozdíl od výše popsaných aliasů, které byly navrženy pro kompatibilitu jmen příkazů s ostatními rozhraními, aliasy zabudované do WPS/PS7 jsou obecně navrženy pro stručnost. Tyto kratší jména příkazů sice můžete psát rychle, ale bez hlubší znalosti nevíte, na co se odkazují.

WPS/PS7 se snaží o kompromis mezi jasnosti a stručnosti tím, že poskytuje standardní sadu vestavěných aliasů, které jsou založeny na zkrácených jménech pro obecná slovesa a podstatná jména. Například ve standardním aliasu sloveso Get je zkráceno na G, sloveso Set je zkrácen na S, podstatné jméno Item zkráceno na I, podstatné jméno jméno Location je zkráceno na L, a podstatné jméno Command je zkráceno na CM.

Příklad 3

Standardní alias `Get-Item` – G pro Get a I pro Item: `GI`.

Standardní alias `Set-Item` – S pro Set a I pro Item: `SI`.

Standardní alias `Get-Location` – G pro Get a L pro Location: `GL`.

Standardní alias `Set-Location` – S pro Set a L pro Location: `SL`.

Standardní alias `Get-Command` – G pro Get a cm pro Command: `GCM`.

Tyto standardní vestavěné aliasy nelze změnit a v případě, že se pokusíte vytvořit takový alias objeví se chybová hláška.

5.7.b) Vytváření nových aliasů

Můžete vytvořit svoje vlastní aliasy použitím cmdletu Set-Alias:

```
Set-Alias -Name <Váš alias> -Value <příkaz>
```

Příklad 4

```
Set-Alias -Name git -Value Get-Item
```

Pokud však použijete aliasy, které jsou již zabudovány do jádra, jako např:

```
Set-Alias -Name gi -Value Get-Item
Set-Alias -Name si -Value Set-Item
Set-Alias -Name gl -Value Get-Location
Set-Alias -Name sl -Value Set-Location
Set-Alias -Name gcm -Value Get-Command
```

objeví se vám následující chyba:

```
PS> Set-Alias -Name gi -Value Get-Item
```

```
Set-Alias : Alias is not writeable because alias gi is read-only or constant and cannot
be written to.
At line:1 char:10
+ Set-Alias  <<<< -Name gi -Value Get-Item
```

Svoje aliasy však musíte po každém startu konzole WPS/PS7 znovu definovat! Vaše aliasy platí jen pro dané session! Pokud tedy chcete používat vaše aliasy trvale, přidejte je do WPS/PS7 profile souboru.

Příklad 5

Pokud chcete vytvořit alias pro příkaz s parametry, pak musíte vytvořit funkci:

```
Function bootini {notepad c:\boot.ini}
```

5.7.c) Výmaz aliasů

```
Remove-Item alias:<název aliasu>
```

5.8. Objekt roury (pipeline), zřetězení

about_pipelines

Zřetězení se chová jako řada propojených částí potrubí. Prvky pohybující se potrubím projdou každým segmentem. Chcete-li vytvořit zřetězení ve WPS/PS7 propojíte jednotlivé příkazy operátorem roury (pipeline) "|" a pak je výstup jednoho příkazu použit jako vstup pro následující příkaz.

Ve WPS/PS7 rouře se předává kolekce objektů od 0 do x členů.

U klasických CLI tato komunikace vyžaduje často manipulace s textovým výstupem, aby se konvertoval výstup na jiný formát a vyjmuly se titulky a hlavičky sloupců. A velice často se muselo odkazovat na pozici určitých dat ve výstupu.Tím, že WPS/PS7 pracuje s objekty, si každý příkaz může pracovat s těmi vlastnostmi a metodami, které potřebuje.

Správně používané potrubí nejenže snižuje komplikovanost ve vkládání příkazů, ale také umožňuje lépe vidět posloupnost příkazů a jejich tok a samozřejmě také redukuje čas na zpracování, protože výstup jednoho příkazu se stává okamžitě vstupem následujícího.

Příklad 6

V případě, že používáte přímý výstup příkazu na obrazovku zatěžujete procesor. Pokud použijete pipeline s výstupem do příkazu `Out-Host` zatížení procesoru se rapidně sníží (protože se zpracovává stránka po stránce, a ne kompletní výstup), například si zkuste porovnat zatížení procesoru (pomocí správce úloh či podobného nástroje) u těchto příkazů:

```
Get-ChildItem C:\Windows -Recurse
Get-ChildItem C:\Windows -Recurse | Out-Host -Paging
```

nebo ještě lépe:

```
Measure-Command {Get-ChildItem C:\Windows -Recurse}
Measure-Command {Get-ChildItem C:\Windows -Recurse | Out-Host -Paging}
```

Příklad 7

Výstup příkazu může být ve WPS/PS7 rovněž zpracován klasickým příkazem. Chcete-li načíst do schránky posledních 20 událostí z event logu:

```
get-eventlog application -newest 20 | clip
```

V nápovědě ke každému cmdletu je uveden nejen typ vstupu (sekce Inputs), ale typ výstupu (sekce Outputs). Pomocí Get-Member nebo nápovědy si můžete zjistit, jaký typ objektu je určen pro vstup či výstup cmdletu. Je to dobré vědět!

Příklad 8

Příklad na to, jak důležité informace jsou v nápovědě a na typ objektu vystupujícího z jednoho cmdletu a vstupujícího do druhého cmdletu:
Chceme killnout („zabít“) nějaký proces:

```
Get-Process | Get-Member
```

vrátí na první řádce TypeName: System.Diagnostics.Process a následně všechny metody a vlastnosti pro procesy - mezi vlastnostmi je vlastnost **Id** a **Name** (tady vidíte, jak důležitý cmdlet ten Get-Member je!).
Nyní si zjistíme, jaký typ vstupního objektu a jaké sady parametrů má cmdlet Stop-Process, který killne process:

```
get-help -Name Stop-Process -Full
```

V sekci Inputs je System.Diagnostics.Process (čili typ objektu vystupujícího z cmdletu `Get-Process` je stejný jako typ objektu vstupujícího do cmdletu Stop-Process). Ještě si v nápovědě zkontrolujeme, jaké má cmdlet `Stop-Process` sady parametrů. Lze vidět, že proces lze předat jeho vlastností Id (ByPropertyName typu Int32) – což je implicitní sada parametrů, vlastním objektem (ByValue) nebo vlastností Name (ByPropertyName typu String). Čili, když `Stop-Process` na vstupu nenajde u vstupujícího objektu vlastnost Id, ověří si, zda je objekt typu System.Diagnostics.Process a pokud ne, tak u vstupujícího objektu hledá vlastnost Name. Uvědomte si, že vlastnost Id nebo Name má spousta objektů různých typů! A v případě sady parametrů, která pracuje s ByPropertyName, je jedno o jaký typ objektu jde! V tomto případě je cmdletu `Stop-Process` jedno, zda se jedná o objekt typu System.Diagnostics.Process nebo jiný typ objektu! Tady si musíte dát pozor, aby nedošlo k tomu, že do Stop-Process pošlete něco jiného, než jste zamýšleli (a můžete si tak zrušit systémový proces nebo procesy). Proto doporučuji tam, kde jde o hodně používat při testování parametr –WhatIf**,** který nám řekne, co by se stalo, ale vlastní cmdlet nevykoná.

Příklad 9

Vybereme z běžících procesů jen ty se jménem iexplore:

```
Get-Process | ? {$_.Name -eq "iexplore"} | Stop-Process
```

zde si odpovídají typy objektů a není co řešit.

Máme nějaký cmdlet nebo spíše funkci (může jít o váš vlastní cmdlet, u kterého lze měnit vlastnosti), která má vracet Id nebo Name procesu, který chcete zrušit (v následujícím případě použiji cmdlet, který vrací jména souborů z adresáře – je to pitomé, ale snad názorné):

```
Get-ChildItem -Path C:\temp | % {$_.BaseName} | Stop-Process
```

jelikož do Stop-Process nevstupuje v tomto případě typ objektu System.Diagnostics.Process, tak cmdlet Stop -Process nejprve hledá vlastnost Id (tu soubor nemá) a tak hledá vlastnost Name a tu soubor má! Takže můžete killnout úplně jiný proces než jste chtěli, protože v adresáři můžete mít například soubor se jménem lsass.txt (a zrušíte tak proces lsass)!

Takže použijte pro testování:

```
Get-ChildItem -Path C:\temp | % {$_.BaseName} | Stop-Process -WhatIf
```

A ověřte si, zda se vám rourou nemůže dostat nesprávná hodnota vlastnosti Id nebo Name.

5.9. Vstupy, výstupy, formátování výstupu a přesměrování

5.9.a) Vstup údajů

Pro práci s obsahem souboru používejte cmdlets *-Content (pozor nelze však číst a zapisovat do stejného souboru zároveň. Pokud to potřebujete, udělejte ze vstupního souboru výraz):

```
Set-Content # Nastaví hodnotu
Get-Content # Načte hodnoty
Clear-Content # Vynuluje obsah souboru
```

Potřebujete-li vstup údajů od uživatele, použijte cmdlet:

```
Read-Host
```

5.9.b) Formátování výstupu příkazů

about_Format.ps1xml

U klasických CLI, všechny příkazy zároveň formátují svůj výstup, některé příkazy mají navíc i parametry k formátování výstupu. Ve WPS/PS7 existují následující cmdlets k formátování výstupu libovolného příkazu:

Format-List	Více property můžete zobrazit jejich oddělením čárkou. Pomocí wildcard * znázorníte všechny property i s jejich hodnotami (`Get-Member` znázorní jen vlastnosti), implicitně se totiž nezobrazí všechny.
Format-Custom	Uživatelský formát výstupu, který je plně pod vaší kontrolou.
Format-Table	Bez upřesnění property se zobrazí stejný výstup jako bez formátování. Pomocí pořadí property, parametru Wrap a Autosize lze řídit zobrazení.
Format-Wide	Zobrazuje jen jednu vlastnost, pokud ji neupřesníte tak jen default property.
Out-GridView	Interaktivní práce s výstupem. Až ve verzi WPS 3.0 jde jeho výstup zpátky do konzoly, takže můžete dělat vícenásobný výběr.

Výstup příkazu může být zpracován některým z následujících cmdlet, které filtrují výstupy:

`ForEach-Object` nebo `ForEach` nebo `%` – opakování úlohy pro více objektů

`Group-Object` – seskupuje výstupní objekty podle zadané vlastnosti.

`Select-Object` – pomocí výběru vlastností či metod vytváří nový objekt, který je už read/write.

`Sort-Object` – třídí výstup podle vybrané vlastnosti (vlastností). Lze zadat i datový typ, např: `Sort-Object -Property { $_ -as [int]  }`

`Where-Object` nebo `Where` nebo `?` – filtrování objektů v pipelině (můžete používat i více fitrů v pipelině za sebou!)

A tak pomocí znalosti těchto cmdlets můžete formátovat všechny příkazy a nemusíte se učit, jak formátovat každý příkaz zvlášť.

WPS určuje jak implicitně zobrazit objektové typy použitím informací uložených v XML souborech jejichž jméno končí na .format.ps1xml. Způsob formátování dat pro objekt NET System.Diagnostics.Process je uložen v PowerShellCore.format.ps1xml.

Pokud ve WPS spustíte nějaký příkaz, je formátován pomocí implicitního formátování. Chcete-li si formát vystupu přizpůsobit, stačí výstup příkazu přesměrovat do některého z uvedených cmdlets Format.

Výstup cmdlets Format-* není objekt, výstup je v textovém tvaru.

5.9.c) Přesměrování dat pomocí Out-* cmdlets

about_redirection

WPS/PS7 má několik cmdlets pro řízení přímého výstupu dat

`Out-Host` – výstup na konzolu.

`Out-Null` – výstup do černé díry, nezahazuje však chybová hlášení.

`Out-Printer` – výstup na tiskárnu.

`Out-File` - implicitně vytváří Unicode soubor, který odpovídá tomu co je na obrazovce.

`Out-GridView` - interaktivní práce s výstupem. Až od verze WPS 3.0 jde jeho výstup zpátky do konzoly.

Tyto cmdlets mají dvě důležité vlastnosti. Ta první je, že výstup těchto cmdlets je v textovém tvaru (čili ne objekt!), protože většina systémových komponent vyžaduje jako vstup text. A ta druhá vlastnost je, že data se vysílají mimo WPS/PS7, takže se ztrácí. Proto nedáte-li cmdlets Out na konec roury, nebudou vám příkazy následující za ním stejně fungovat. Cmdlets Out musí být tedy vždy posledním příkazem v rouře! Nezapomeňte, že výstup je limitován délkou řádky výstupního zařízení! Zajímavý je též cmdlet Tee-Object, který může poslat výstup zároveň do souboru nebo proměnné, a i dál do roury. Samozřejmě i ve WPS/PS7 fungují klasické přesměrovávací operátory, které znáte z klasických CLI.

6. PROGRAMOVÁNÍ

6.1. Wildcard – žolíky, divoké znaky

```
about_wildcards
```

Na rozdíl od wildcards v cmd.exe, ve WPS/PS7 wildcard zastupují také časová období. Lze použít kdekoliv (třeba ve výčtu vlastností):

*****	Porovnává žádný (tedy i prázdný znak) nebo více znaků.
?	Porovnává přesně jeden znak na dané pozici.
[znak1 – znakx]	Porovnává rozmezí znaků.
[znak]	Porovnává specifikovaný znak.
x..y	Sekvence po sobě jdoucích čísel (nefunguje na znaky abecedy).

6.2. Escape sekvence, speciální znaky a komentáře

```
about_special_characters
about_Escape_Characters
about_Quoting_Rules
```

6.2.a) Speciální escape sekvence

Speciální escape sekvence (escape znaky začínají zpětným (opačným) apostrofem (levý ALT+096). Znakové literály začínají znakem apostrof ' .

`	Back tick	Znak Escape (levý ALT+096). V konzoli dovoluje pokračovat v příkazu na dalším řádku (pozor nesmí za ním následovat mezery!). Pokud ho uvedete před znakem dolaru, nebude interpretován jako proměnná. Pokud ho uvedete v řetězci před uvozovkou, ta nebude interpretována jako konec řetězce.
`0	Null	Není totožný s hodnotou NULL.
`a	Alerts	Pípnutí.
`b	Backspace	Přesun kurzoru o jeden znak vlevo bez jeho výmazu.
`f	Form feed	Odstránkování (pouze u tisku, ne na obrazovce).
`n	New line	Nová řádka.
`r	Carriage return	Návrat na začátek řádky.
`r`n	Odřádkování	Carriage return + new line.
`t	Horizontal Tab	Vodorovný tab.
`v	Vertical Tab	Vertikální tab (pouze u tisku, ne na obrazovce).
`$	$	Znak dolaru.
``		Zpětný (opačný) apostrof.
Speciální znaky (case-sensitive)		
ENTER (Carriage return)	Line break.	Konec řádky. Povolen mezi příkazy, uvnitř řetězců a po těchto oddělovačích: [| , ; = -as] a od verze 3 [. ::]. Je také povolen po otevíracích tokenech: [{ [(" '].
&	Call operator	Řetězec bude použit jako příkaz např. & "Get-Process".
; (Semicolon)	Statement separator	Oddělovač příkazů. Volitelný, pokud používáte ENTER pro oddělení příkazů. Povinný, pokud zadáváte více příkazů na jedné řádce.
{… }		Blok skriptu. Viz about_script_blocks.
|	Pipeline.	Roura pro předávání výstupu jednoho příkazu druhému. Viz about_pipelines.
""	Double quotation marks	Uvozovky (uvnitř řetězce jsou proměnné či výrazy nahrazeny jejich hodnotami). Uvozovku jako literál znázorníte dvěma uvozovkami.
'	Single quotation marks	Apostrof (nenahrazuje uvnitř řetězce proměnné či výrazy jejich hodnotami). Apostrof jako literál znázorníte dvěmi apostrofy.

6.2.b) Komentáře

#		Začátek jednořádkového komentáře, může být na samostatném řádku i v řádku s příkazem: # Toto je komentář protože # je první znak tokenu $a = "#toto není komentář…" $a = "something" # …ale toto je komentář.
<#		Začátek víceřádkového komentáře.
#>		Konec víceřádkového komentáře.

6.3. Operátory

```
about_operators
```

6.3.a) Priorita operátorů

Ve WPS/PS7 jsou operátory vyhodnocovány v následujícím pořadí: () {}, @ $, !, [ ], ., &, ++ --, Unární + -, * / %, Binární + -, porovnávací operátory, -and – or, |, > >>, =

Můžete sčítat čísla, řetězce, pole a hash tabulky. Násobit můžete čísla, řetězce, pole – ne však hash tabulky!

Protože se mohou sčítat a násobit řetězce a čísla neplatí zákon komutativnosti (tj. a+b = b+a neplatí)!

POZNÁMKA 3: Pokud si nejste jisti s prioritou operací používejte závorek, kterými jednoznačně určíte pořadí vykonávání. To použijte např. u sady příkazů, kde čtete a zapisujete do jednoho a toho samého souboru – vstupní soubor vraťte jako výraz.

6.3.b) Aritmetické binární operátory

```
Get-Help about_arithmetic_operators
```

=	Přiřazuje hodnotu do proměnné.
+	Sčítání, spojování.
-	Odčítání nebo záporné číslo.
*	Násobení v případě čísel, opakování řetězce v případě řetězců.
/	Dělení.
%	Modulo (zbytek po dělení)

6.3.c) Boolean hodnoty a operátory

TRUE (Pravdivý výrok)	FALSE (Nepravdivý výrok)
$TRUE	$FALSE.
Jakýkoliv řetězec o délce > 0 vyjma slova “false”	Prázdný řetězec nebo řetězec řetězec “false”.
Jakékoliv nenulové číslo	Nula.
Pole délky > 1	Pole délky 0.
Pole délky 1 jehož prvek je je TRUE	Pole délky 1 jehož prvek je FALSE.
Reference na jakýkoliv objekt	Null.

6.3.d) Logické operátory

```
about_logical_operators
```

```
Syntaxe:
<příkaz> {-AND|-OR|-XOR} <příkaz> nebo {!|NOT} <příkaz>
```

!	Negace
-not	Negace.
-and	Logický součin.
-or	Logický součet.
-xor	Exkluzívní logický součet.

6.3.e) Přiřazovací operátory

```
about_assignment_operators
```

```
Syntaxe:
Výraz1<operátor>=výraz2 (což znamená: výraz1  = výraz1 <operátor> výraz2)
```

Kde operátory jsou: = ; + ; - ; * ; / ; %

Např. `$a = $a + 10` můžeme napsat zkráceně též jako `$a += 10`

6.3.f) Unární operátory

```
about_operators
about_split
about join
```

```
Syntaxe:
-Join <String[]>
<String[]> -Join <Delimiter>

-Split <String>
<String> -Split <Delimiter>[,<Max-substrings>[,"<Options>"]]
<String> -Split {<ScriptBlock>} [,<Max-substrings>]
```

++	Inkrementace.
--	Dekrementace.
,	Pole s jedním členem.
-join	Spojuje řetězce, oddělené čárkou. Jsou-li v závorce spojí je do jednoho řetězce na řádku, jinak každý odřádkuje.
-split	Rozdělí řetězec podle nastaveného delimiteru, pokud delimiter není uveden, odděluje se podle mezer mezi slovy. Lze používat i RegEx.

POZNÁMKA 4: Inkrementační a dekrementační operátory fungují logicky podle pozice na které jsou umístěny, tzn., že ++$a inkrementuje nejprve proměnnou $a a pak se proměnná použije, kdežto u $a++ se proměnná nejprve použije a pak inkrementuje!

6.3.g) Porovnávací operátory a porovnávací operace

about_Comparison_Operators

-eq	Rovná se.
-ne	Nerovná se.
-gt; –ge	Větší než; větší nebo rovno.
-lt; –le	Menší než; menší nebo rovno.
-like	Používá wildcard k vyhledání vzoru.
-notlike	Používá wildcard k vyhledání vzoru, neguje výsledek.
-match	Porovnání použitím regulárního výrazu na vyhovující řetězec (podporuje RegEx). Vrací True nebo False a pouze pro první výsledek. Proměnná typu pole $Matches vrací vlastní výsledek. Chcete-li vrátit všechna porovnání použijte Select-String s volbou pattern.
-notmatch	Porovnání použitím regulárního výrazu na nevyhovující řetězec.
-contains	Obsahuje identickou hodnotu (ne část řetězce!).
-notcontains	Neobsahuje identickou hodnotu.
-replace	Změní (přepíše) daný prvek. Lze použít i RegEx.

POZNÁMKA 5: Můžete použít prefix “i” nebo “c” pro operaci case-insensitive nebo case-sensitive (například –ceq). Implitně case-insensitive.

Příklad 10

Obsahuje pole číslo 3?

```
1,2,3,5,3,2 –contains 3
```

Vrací všechny prvky rovné číslu 3:

```
1,2,3,5,3,2 –eq 3
```

Vrací všechny prvky menší než 3:

```
1,2,3,5,3,2 –lt 3
```

Udělat něco, když pole obsahuje číslo 2?

```
if (1, 3, 5 –contains 2) …
```

6.3.h) Ternární operátory

about_If

Platné jen pro PS7.

Ternární operator vychází ze syntaxe C#. Podmínka je vyhodnocena jako Boolean a podle výsledku se provede příslušná větev příkazu.

```
Syntaxe:
<condition> ? <if-true> : <if-false>
```

Příklad 11

```
$message = (Test-Path $path) ? "Path exists" : "Path not found"
```

6.3.i) Operátory kanálů potrubního řetězce (Pipeline chain operators)

about_Pipeline_Chain_Operators

Platné jen pro PS7. Tyto operatory provádějí podmíněné řetězení kanálů.

Tyto operatory používají proměnné $? A $LASTEXITCODE pro určení úspěšnosti pipeline. To znamená, že je můžete používat nejen v cmdlets a funkcích, ale i v nativních příkazech.

&& operátor vykoná pravou stranu pipeline, pokud je levá strana True.

|| operátor vykoná pravou stranu pipeline, pokud je levá strana False.

Příklad 12

```
#levá strana True, pravá strana se vykoná
Write-Output 'First' && Write-Output 'Second'

#levá strana False, pravá strana se nevykoná
Write-Error 'Bad' && Write-Output 'Second'

#levá strana True, pravá strana se nevykoná
Write-Output 'First' || Write-Output 'Second'

#levá strana False, pravá strana se vykoná
Write-Error 'Bad' || Write-Output 'Second'
```

6.3.j) Null slučovací, Null přiřazovací a Null podmínkové operátory.

about_Operators
Using Experimental Features in PowerShell

Platné jen pro PS7.

??	Operátor slučování Null. Vrátí hodnotu leveho operandu, pokud není null (poté už nevyhodnocuje pravý operand), jinak hodnotu pravého operandu.
??=	Operátor podmíněného přiřazení null. Přiřadí hodnotu svého pravého operandu levému operandu pouze v případě, že levý operand je vyhodnocen jako null. Pokud je levý operand vyhodnocen jako Null, není pravý operand vyhodnocen.
?.	Podmíněný operátor null (zatím experimentální operátor). Povoluje přístup členů nebo k prvku ke svému operandu pouze v případě, že je tento operand vyhodnocen jako nenulový; v opačném případě vrátí hodnotu null.
?[]	Podmíněný operátor null (zatím experimentální operátor). Povoluje přístup členů nebo k prvku ke svému operandu pouze v případě, že je tento operand vyhodnocen jako nenulový; v opačném případě vrátí hodnotu null.

6.3.k) Ostatní operátory

about_type_operators
about_operators

%	Alias pro ForEach-Object v objektu pipeline.
?	Alias pro Where-Object v objektu pipeline.
&	Invoke (Call) Operator & (vyvolávací operátor). Může být použit k vyvolání bloku skriptu nebo příkazu či funkce, jež jsou uvedeny jako řetězec. Příklad: $a = “Get-Process” &$a $a = { Get-Process \| Select -First 2 } &$a
.	Oddělovač v cestě třídy, např. Systém.IO.FileInfo. Odkaz na metodu nebo vlastnost objektu, např. $a.length Dot source operátor. Spouští funkce, skriptovací bloky a skripty v běžném oboru (za tečkou pište mezeru!) - proměnné definované ve skriptu jsou pak přístupné i ve vaší konzoli! Příklad: . MyFunction Jestliže MyFunction nastavuje proměnnou, ta je nastavena v běžném rozsahu. $a = {$x = Get-Process | Select –First 2} . $a #Evaluates the script block in the current scope
::	[třída]::člen - volání statického členu, např. [math]::pi
-f	Formátovací operátor.
$( )	Operátor podvýrazu. Vrací stejné hodnoty jako @().
@( )	Operátor podvýrazu pole. Převádí hodnoty na pole.
,	Konstruktor pole.
..	Operátor rozsahu (vytváří třeba pole) např: 1..10 vytvoří pole
-is	Operátor typu (odhadce typu): $a -is [int]
-isnot	Operátor typu (negace -is): $a -isnot [int]
-as	Operátor typu (konvertor typu): 1 -as [string] #zpracuje 1 jako řetězec
-band	Binární and.

-bor	Binární or.
-bnot	Binární not.
$( )	Vrací null.
$(1,2,3)	Vrací pole obsahující 1,2,3.
$(Get-Alias a*)	Vrací vyhodnocení výrazu.
@(Get-Alias;Get-Process)	Vykonává dva příkazy a vrací výsledek v poli.
#<tab> #string<tab>	V příkazové řádce slouží pro vyvolání historie příkazů a jejich editování. Můžete i pomocí stringu vyhledávat příkazy obsahující určitý řetězec.
$( )	Subexpression. Podvýraz v kulatých závorkách. Např. $($a=1;$b=2;$b;$a).
(…)	Grouping expression . Skupina výrazů v kulatých závorkách. Lze změnit prioritu operátorů např. 8+4/2 vs. (8+4)/2 nebo umožňuje přistoupit k výsledkům operace např. (Get-Date).month nebo předá výsledek operace jako argument např. write-output (1*54).

6.3.l) Redirecting (přesměrovávací) operátory

```
about_redirection
```

```
Syntaxe:
<input> <operator> [<path>\]<file>
```

>	Posílá stdout stream do specifikovaného souboru (pokud soubor existuje, přepíše ho).
>>	Přidává stdout stream na konec obsahu specifikovaného souboru.
n>	Posílá stream číslo n do specifikovaného souboru (pokud soubor existuje, přepíše ho). N je 2 až 5 nebo * pro všechny streamy.
n>>	Přidává stream číslo n na konec obsahu specifikovaného souboru. N je 2 až 5 nebo * pro všechny streamy.
n>&1	Vysílá stream číslo n do stdout streamu. N je 2 až 5.

6.4. Proměnné, konstanty a rozsahy

```
about_variables
about_scopes
Get-Command –Noun Variable
Get-Variable
```

6.4.a) Konstanty

Vytváření konstant

```
Set-Variable
```

Příklad 13

```
Set-Variable -name pi -value 3.14159265358979 -option constant –Scope global
```

6.4.b) Proměnné

$ (Dollar)	Variable prefix	Prefix pro proměnnou následovaný písmeny, číslicemi nebo podtržítky specifikuje jméno proměnné. Písmena a číslice nejsou limitovány ASCII tabulkou, ale obecně se doporučuje používat pouze první část ASCII tabulky. Použije se obsah proměnné!
${…}	Variable prefix	Pro vložení jakýchkoliv jiných znaků do jména proměnné. Např. ${save-items}. Viz about_variables.
${path}	Path accessor	Speciální případ proměnné pro uložení (např. ${c:\tmp\a.txt}=“pokus“) nebo načtení (např. $data=${c:\tmp\a.txt} ze souboru.
[datatype]$<jmenopromenne>		Určuje datový ty proměnné. Datatype je decimal, int, int32, int16, uint16, int64, uint64, long, ulong, single, double, float, char, bool, string, byte, sbyte, datetime, array, object, version atd.

Každá proměnná začíná znakem $, je objektem a má vlastnosti. Proměnné mají vlastní jednotku PSDrive Variable. Lze je vytvářet klasicky nebo cmdletem Set-Variable. Pokud předřadíte jménu proměnné (lze samozřejmě použít u funkce) datový typ v hranatých závorkách, pak PS se bude snažit přetypovávat vkládané hodnoty na deklarovanou.

```
Syntaxe:
$[scope:]name
```

```
[datatype]$name – určuje datový typ proměnné
${anyname}
${anypath}
```

Způsob tvorby proměnných:

Příklad 14

```
$a = 1
$env:path = "d:\windows"
$global:a = 1  # Visible everywhere
$local:a = 1     # defined in this scope and visible to children
$private:a=1    # same as local but invisible to child scopes
$script:a=1     # visible to everything in this script
nebo
${anyname}
${!@#$%^&*()}=3
nebo
${any path}
${C:\TEMP\testfile.txt}="This writes to a file"
```

WPS/PS7 automaticky vybírá správný typ pro proměnnou a případně rozšiřuje rozsah. Typ proměnné zjistíte snadno pomocí metody GetType() . Převod proměnné na určitý datový typ se provádí pomocí prefixu (ve skutečnosti typ proměnné zůstává stejný):

```
[datatype]  <jméno proměnné>
```

kde datatype je decimal, int, int32, int16, uint16, int64, uint64, long, ulong, single, double, float, char, bool, string, byte, sbyte, datetime, array, object, version atd.

Také existuje ukazatel na proměnnou vracející adresu proměnné (ale ve skutečnosti vrací typ .NET):

```
[ref]${variablename}
```

Pokud chcete kontrolovat deklarování proměnných před jejich použitím (ve Visual Basicu totéž dělá příkaz Option Explicit) použijte:

```
Get-PSDebug -strict
```

6.4.c) Rozsahy

Rozsahy platí pro proměnné, funkce, aliasy a PSDrivy. Ve skriptu jsou implicitně proměnné vytvářeny v rozsahu Script, ve funkcích a aliasech v rozsahu local, pokud jsou již definovány ve skriptu! Pozor stejné proměnné v různých rozsazích jsou dvě různé proměnné a musíte se na ně odkazovat modifikátorem rozsahu! Proměnnou můžete změnit v libovolném rozsahu, ale musíte se na ni odkázat modifikátorem rozsahu. Nový rozsah vzniká spuštěním skriptu nebo funkce, nebo vytvořením session, nebo spuštěním nové instance WPS/PS7.

Pokud vyvoláte ve skriptu další skript pomocí zkrácené notace (. \scripts\pokus,ps1), pak všechny funkce, aliasy a proměnné z tohoto skriptu jsou dostupné v běžném rozsahu (local), při normálním nebo asynchronním volání skriptu ne.

```
Syntaxe: $[<modifikátor rozsahu>]:<jméno proměnné> = <hodnoty>
```

nebo použijte cmdlets

```
New-Variable
New-Alias
```

Modifikátory rozsahu:

Global
Proměnné v globálním rozsahu jsou viditelné ve všech oborech, vytvářejí se při startu WPS/PS7. Používejte jen v nejnutnějších případech.

Script
Proměnné v oboru skriptu jsou viditelné ve všech oborech uvnitř skriptu. Vzniká při startu skriptu.

Local
Běžný rozsah. Může to být global, script nebo private. Proměnné v lokálním oboru jsou viditelné jen v běžném oboru a jeho dětech.

Env
Odkaz na PSDrive s proměnnými.

Private
Privátní obor proměnných je viditelný jen v rámci běžného rozsahu. Můžete ho použít pro privátní proměnné v rámci jiného rozsahu.

Option Allscope
Tento parametr nastavuje viditelnost ve všech rozsazích. Týká se některých cmdlets:

Očíslovaný rozsah
Na rozsahy můžete odkazovat i číslem: local=0; 1=přímý nadřazený rozsah atd.

Příklad 15

```
$global:a = 4
$script:a = 5
$local:a  = 3
$private:a = 6
```

Lze zjišťovat a nastavovat proměnné v různých oborech:

```
Get-Variable –scope <scope-modifier>
```

Zjištění, které cmdlets pracují s rozsahy:

```
Get-Help * -parameter scope
```

6.4.d) Automatické (vestavěné) proměnné

about_automatic_variables

$$	Poslední token předešlé command line.
$?	Boolean stav posledního příkazu.

$^	První token předešlé command line.
$_	Běžný pipeline objekt (aktuálně zpracovávaný objekt).
$Args	Argumenty skriptu nebo funkce (pole). Přebírají se pomocí výrazu $args[x], kde x je pořadové číslo argumentu počínaje od nuly (stejně jako v C#).
$ConsoleFileName	Cesta souboru konzole (.ps1) jenž je používán v tomto sezení.
$DebugPreference	
$Error	Pole chyb z předešlého příkazu.
$ErrorActionPreference	Preferovaná implicitní reakce na výskyt chyby (SilentlyContinue, Continue – default, Inquire, Stop).
$ExecutionContext	Obsahuje EngineIntrinsics objekty, jež reprezentují kontext vykonávání WPS/PS7.
$False	Obsahuje FALSE.
$Foreach	Odkaz na enumerator ve foreach cyklu.
$Home	Uživatelský home directory; obyčejně %HOMEDRIVE%\%HOMEPATH%
$Host	Odkaz na hostující aplikaci jazyka POWERSHELL.
$Input	Enumerator objektů piped do skriptu.
$LastExitCode	Exit kód posledního programu nebo skriptu.
$Matches	Hash tabulka nalezených shod s –match operátorem.
$MyInvocation	Obsahuje objekt s informacemi o běžném příkazu jako jsou skript, funkce, nebo blok skriptu.
$NestedPromptLevel	Obsahuje úroveň běžného promptu. Hodnota 0 indikuje originální úroveň promptu.
$NULL	Obsahuje NULL nebo empty hodnotu.
$PID	Obsahuje process identifier (PID) procesu.
$PSBoundParameters	Obsahuje slovník aktivních parametrů a jejich stávajících hodnot.
$PsCulture	Obsahuje jméno kultury nastavené v systému.
$PSDebugContext	Při ladění obsahuje informace o ladícím prostředí. Jinak obsahuje NULL.
$PSHome	Umístění instalace WPS/PS7.
$PSISE	Proměnná pouze pro prostředí ISE, přes tuto proměnnou je ISE prostředí přístupné.
$PSScriptRoot	Obsahuje cestu ke skožce, kde je uložen spouštěný skript.
$PSSenderInfo	Proměnná, pomocí níž můžete zjišťovat vlastnosti remote session.
$PsUICulture	Obsahuje jméno kultury uživatelského rozhraní (UI) jež je nastavena v systému.
$PsVersionTable	Obsahuje hash tabulku jen ke čtení, jež zobrazuje detaily o spuštěné verzi WPS/PS7.
$profile	Standardní profile (nemusí existovat!).
$Pwd	Obsahuje cestu, jež reprezentuje plnou cestu k běžnému adresáři
$ShellID	Obsahuje identifikátor běžného shellu.
$StackTrace	Poslední výjimky zjištěné WPS/PS7.
$Switch	Enumerator ve switch příkazu.
$This	V bloku skriptu definujícího vlastnost nebo metodu skriptu odkazuje na vlastní blok skriptu.
$True	Obsahuje TRUE.

6.4.e) Preferenční proměnné

```
about_preferences_variables
```

Upravují chování WPS/PS7.

6.4.f) Proměnné prostředí operačního systému

```
about_environment_variables
```

Proměnné prostředí operačního systému jsou přístupné přes alias env:

6.5. Řetězce

""	Double quotation marks	Uvozovky (uvnitř řetězce jsou proměnné či výrazy nahrazeny jejich hodnotami). Uvozovku jako literál znázorníte dvěma uvozovkami.
'	Single quotation marks	Apostrof (nenahrazuje uvnitř řetězce proměnné či výrazy jejich hodnotami). Apostrof jako literál znázorníte dvěmi apostrofy.

6.5.a) Here-strings (používají se pro uložení více řádků textu do proměnné – viz ConvertFrom-StringData)

@" Text i o více řádcích "@	Literal here-string	Proměnné v textu jsou nahrazeny jejich skutečnými hodnotami. Na rozdíl od normálního řetězce můžet vložit uvozovky, aniž byste je museli zdvojovat nebo použít escape sekvenci.
@ ' Text i o více řádcích '@	Interpolated here-string	Proměnné v textu nejsou nahrazeny jejich hodnotami. Na rozdíl od normálního řetězce můžet vložit apostrof, aniž byste jej museli zdvojovat nebo použít escape sekvenci.

Řetězce se mohou uvádět v uvozovkách nebo v apostrofech: Proměnná v uvozovkách se nahradí svou hodnotou. Zatímco proměnná použitá v apostrovech se nenahrazuje. Znaky @“, @‘ a „@,‘@ musí být na samostatném řádku a nesmí za nimi následovat mezera.

Řetězcové konstanty:

```
“toto je řetězec,s vyhodnoceno jako $(2+2)”
‘toto je řetězec, tato proměnná $variable není vyhodnocena’
@”
Toto je řetězec “řetězec” který může obsahovat cokoliv se znakem návratu řádku a uvozovky. Výraz $(2+2) je
vyhodnocen.
”@
@’
“řetězec” s apostrofy nevyhodnocuje výrazy.
‘@
```

6.5.b) Řetězcové operátory

+	Zřetězení dvou řetězců
*	Opakování řetězce
-f	Formátování řetězce (.NET format specifiers)
-replace	Nahrazovací operátor "abcd" –replace “bc”, “TEST” (výsledek je ATESTd)
-match	Regular expression match
-like	Wildcard matching

6.5.c) Formátování řetězců

about_operators (operátor -f)

```
Syntaxe:<formátovací řetězec> -f <formátovaná položka>
“{x:Tz}” je to fomátovací řetězec!
kde x je pořadové číslo parametru, T je jednopísmenná zkratka typu proměnné a z počet desetinných míst (v
případě desetinných čísel). Je to stejné jako v C#. V případě textové položky se uvádí pouze počet míst
(pokud řetězec nedosahuje nastavené délky je nahlášena chyba!).
```

Typ proměnné	Popis (viz Help .Net Framework “Standard numeric format strings”)
C nebo c	Currency
D nebo d	Decimal
E nebo e	Scientific (Exponencial)
F nebo f	Fixed-point
G nebo g	General
N nebo n	Number
X nebo x	Hexadecimal (X=velká písmena; x=malá písmena)
P nebo p	Percent
R nebo r	Round-trip

Příklad 16

```
"{0:P1}" -f  45
```

6.6. Pole

```
about_arrays
about_splatting
```

Pokud potřebujete pracovat s polem, v každém případě (i v případech kdy je vrácena jedna hodnota) použijte konstrukci @(), která převede hodnotu na pole a vrací počet prvků pole.

“a”,“b”,”c”	Pole řetězců.
$array='a,b,c' -split ','	Lze i takto
1,2,3	Pole celých čísel.
x..y	Přiřazení rozmezí do pole.
@(<výraz>)	Převede hodnotu na pole a vrací počet prvků pole.
@()	Prázdné pole. Vrací pole i když je nulové nebo neobsahuje objekt. Viz about_arrays.
@(2)	Pole s jedním prvkem.
1,(2,3),4	Pole v poli.
,”hi”	Pole jednoho prvku.
[datatype[]]	Určuje datový typ pole. Datatype je decimal, int, int32, int16, uint16, int64, uint64, long, ulong, single, double, float, char, bool, string, byte, sbyte, datetime, array, object, version atd.

První prvek v poli je indexován od 0. Chcete-li skočit na poslední prvek v poli použijte index -1.

$a[5]	Šestý prvek pole
$a[2][3]	Čtvrtý prvek třetího prvku dvoudimenzionálního pole
$a[2..20]	Vrací prvky 3 až 21

Pokud není určeno explicitně, prvky pole jsou typu object. Explicitně určete typy prvků takto:

```
[type[]]$array
```

kde type je typ prvku (int32, string, apod.). Např:

```
[int32[]]$i = 10,20,30
```

Pole má samozřejmě vlastnosti jako jsou count apod. Další prvek v poli přidáte prostým přičtením: $pole = $pole + "s".

Pokud potřebujete předat do parametru cmdletu hodnoty pole, použijte @namearray

6.7. Asociativní pole (Hashtables) – hash tabulky

```
about_hash_tables
about_splatting
```

Hash table se skládá z klíče (vždy jedinečný řetězec!) a jeho hodnoty oddělené středníkem. Hashtabulka nezajišťuje pořadí jednotlivých položek! To musíte udělat prefixem [ordered].

@{…}	Inicializace hash tabulky ve formátu @{jmenoklice=“hodnota“;…“}
$hash = [ordered]@{ }	Vytvoří prázdnou hashtable a uloží do proměnné hash.
$hash =[ordered]@{key1=1;key2=2}	Vytvoří a inicializuje hashtable v proměnné hash.
$hash.key1 = 1 $hash.Add(„key1“,1)	Přiřadí 1 do klíče “key1”
$hash.key1	Vrací hodnotu key1
$hash["key1"]	Vrací hodnotu key1
$hash["k"]=$hash["k"]+1	Přiřaď do hodnoty klíče k hodnotu o 1 větší

Chcete-li třídit hash tabulku (ať už podle klíče nebo hodnoty musíte použít konstrukci: $h.getenumerator() | sort name).

Nelze sčítat hash tabulky, které mají stejný název klíče. Hash tabulku můžete použít místo parametrů v příkazu:

```
$MyCommandParameters = @{
        ComputerName = $MyDomainController
        Credential = $DomainAdminCreds
        ConfigurationName = 'ActiveDirectoryAdmin'
        JobName = 'GettingWorkDone'
        AsJob = $true
        ScriptBlock = {Get-ADUser -filter *}
}
Invoke-Command @MyCommandParameters
```

Nebo také pro třídění podle více položek (E je zkratka pro Expression a A je zkratka pro Ascending):

```
Get-Service | Sort-Object -Property @{E='Status'; A=$False},@{E='Name';A=$True} | Out-GridView
```

6.8. Parsing (analyzování)

about_parsing

WPS/PS7 analyzuje ve dvou módech: command mode a expression mode. V expression mode WPS/PS7 analyzuje stejně jako high-level programovací jazyky: čísla jako čísla, řetězce musí být v uvozovkách atd. Výrazy jsou například:

```
2+2
4
(2+2)
4
"Hello" + " world"
Hello world
$a = "hi"
$a.length * 13
26
```

Pokud analyzuje v command mode, řetězce nemusí být uzavřeny v uvozovkách a vše je vyhodnocováno jako řetězec vyjma proměnných v závorkách. Příklad:

```
copy users.txt accounts.txt
```

users.txt a accounts.txt jsou zpracovány jako řetězce

```
write-host 2+2
```

2+2 jsou zpracovány jako řetězce, ne jako výraz

```
copy $src $dest
```

$src a $dest jsou proměnné. Není třeba v command shellu používat uvozovky, což samozřejmě zrychluje psaní. Analyzovací mód je určován prvním dosaženým tokenem: je-li první token číslo, proměnná nebo řetězec v uvozovkách nebo tečka následovaná číslem pak se shell přepne do expression mode. Pokud řádka začíná písmenem, & (ampersandem) nebo . (tečkou) následovanou mezerou pak se přepne do command mode.

2+2	Expression mode – začíná číslem.
"text"	Expression mode – začíná uvozovkami.
$<proměnná>	Expression mode – znak dolar označuje proměnnou.
.125	Expression mode –začíná tečkou následovanou číslem.
.text	Command mode – začíná tečkou následovanou textem.
Text	Command mode - začíná písmenem.
& "text"	Command mode – začíná ampersandem.
. "file.ps1"	Command mode – začíná tečkou následovanou mezerou.

Je dobré míchat výrazy a příkazy a používáním závorek jednoznačně určíte, že se jedná expression mode.

```
Write-Host (2+2)
```

2+2 je zpracováno jako výraz.

Get-Date (říká, kolik času uplynulo od půlnoci) je zpracován jako příkaz a jeho výsledek je výraz:

```
(Get-Date).day + 2
22
```

Můžete vnořovat příkazy a výrazy libovolně.

Např.

```
Write-Host ((Get-Date).day + 2)
```

Get-Date je příkaz ((Get-Date).day+2) je výraz a Write-Host ((Get-Date).day + 2) je znovu příkaz.

```
Write-Host ((Get-Date) - (Get-Date).date)
$day = new-timespan -day 1
Get-Date + $day vrátí chybu
(Get-Date) + $day vrátí zítřejší datum.
```

POZNÁMKA 6: Pokud tedy příkaz uzavřete do závorek stává se z něj výraz (jak je to u programovacích jazyků obvyklé). Pokud WPS/PS7 najde otvírací závorku automaticky předpokládá, že bude následovat pokračování, třeba i na dalším řádku (ale pouze v konzoli ne v ISE).

Pokud potřebujete pracovat s proměnnými systému je od verze 3 nabízen stop-parsing parametr, který nahrazuje escape sekvence. Pokud tedy vložíte v příkazu znak --%, nebudou následující znaky interpretovány nijak PWS+PS7.

6.9. Spouštění skriptů

about_Command_Searches
about_jobs
about_job_details

Cmdlets, aliasy, funkce můžete spouštět bez zadání cesty. Spouštět skripty (soubory .ps1, .bat, .cmd, .com, .exe, .vbs, .js, .wsh, .wsf) ve WPS/PS7 můžete jen zadáním plné cesty, i když je skript v běžném adresáři, kde stojíte. Skript se spouští v místě specifikované cesty. Je to z bezpečnostních důvodů. Pokud chcete spustit skript v běžné cestě, tak použijte buď plné cesty (např. c:\scripts\pokus.ps1) nebo zkrácené notace (.\pokus.ps1 – kde .\označuje běžný adresář; lze použít i ./pokus.ps1). Jedinou výjimkou z tohoto pravidla je spouštění programů, které jsou v proměnné PATH systému – tam nemusíte uvádět plnou cestu (např.: ping AKH0010).

Pokud chcete spustit ze souboru ps1 další skript ps1, pak použijte zkrácenou notaci.

WPS/PS7 vyhodnocuje příkazy v následujícím pořadí: aliases, functions, cmdlets, scripts, exe soubory, normální soubory v běžné cestě.

WPS/PS7 příkazy mohou být uloženy a vykonány ze skriptovacího souboru. Přípona souborů pro WPS/PS7 skripty je ".ps1" (tato přípona je automaticky přidružena k notepadu, takže soubor nelze spustit spuštěním "skript.ps1" jako u cmd nebo vbs skriptů). Parametry mohou být předány do skriptu a skript může

vracet hodnotu. Bohužel nefungují pojmenované argumenty. Pokud přidáte jako první znak do příkazového řádku spouštějícího skript znak tečka následovaný mezerou, pak jde o tzv. dot sourcing – proměnné definované ve skriptu jsou přístupné i ve vaší konzoli!

Příklad 17

```
$sum = MyAdder.ps1 1 2 3
powershell.exe –noexit c:\temp\MyAdder.ps1 1 2 3
&”c:\temp\MyAdder.ps1 1 2 3”
```

6.9.a) Asynchronní spouštění skriptů

Skripty lze spouštět asynchronně (tzn. na pozadí) pomocí cmdlets:

```
Invoke-Item s parametrem –AsJob
Start-Job
```

Kontrolu jobů provádíte příkazem:

```
Get-Job
```

Pozor cmdlet Receive-Job po použití smaže data, takže pokud ho použijete podruhé neřekne vám již nic. Toto odstraní parametr –Keep.

6.10. Script Blocks (bloky skriptů)

about_scripts_block

Skupiny příkazů a výrazů mohou být uloženy v bloku skriptu a vykonány později. Jednotlivé příkazy ve složených závorkách musí být odděleny znakem středník.

Příklad v případě použití script blocks na příkazové řádce nebo ve skriptu:

```
$block = {Get-Process; $a=1} # Definice bloku
&$block # spuštění bloku
V případě použití script blocks v pipeline se automaticky vykoná!
```

6.11. Metody volání

Příklady:

```
$a = “This is a string”
$a.ToUpper()
$a.SubString(0,3)
$a.SubString(0,($a.length/2))
$a.Substring(($a.length/2), ($a.length/3))
```

Statické metody a vlastnosti jsou volatelné operátorem “::” a musíte název statické třídy uzavřít v hranatých závorkách. Např.:

```
[DateTime]::IsLeapYear(2005)
[DateTime]::Now
[System.Environment]::CommandLine
```

Samozřejmě můžete používat Tab pro listování nabízenými vlastnostmi a metodami. Pokud potřebujete zjistit nabízené metody a vlastnosti, pak použijte Get-Member –Static:

```
[Math] | Get-Member -Static
```

6.12. Získání informací o objektu

about_properties
about_methods

Jedním z nejdůležitějších úkolů je zjistit typ, vlastnosti a metody určitého objektu (příkazu). K tomu slouží Cmdlet:

```
Get-Member
```

Tento příkaz vrací jméno namespace (uvědomte si, že WPS/PS7 je založen na .NET). Přes Search box v **MSDN** pak můžete zjistit podrobnosti o této .NET třídě.

Chcete-li zjistit hodnotu konkrétní vlastnosti, musíte použít tuto syntaxi:

```
(příkaz <upřesnění>).<jméno-vlastnosti>
```

například:

```
(Get-Service alerter).canpauseandcontinue
```

Chcete-li zavolat určitou metodu daného příkazu (metoda je funkce, a tak má vždy závorky):

```
(příkaz <upřesnění>).<jméno-metody>()
```

například:

```
(Get-Service schedule).stop()
```

Na vlastnosti objektu můžete tedy odkazovat přímo operátorem “.”

```
$a = Get-Date
$a.Date
$a.TimeOfDay.Hours
```

6.13. Příkazy WPS/PS7

Více příkazů na jednom řádku se odděluje pomocí znaku středník;.

6.13.a) Break

about_break

Break příkaz ukončuje Foreach, For, While, Do nebo Switch. Můžete použít navěští (LABEL), což umožňuje přerušit cyklus a pokračovat prvním příkazem za cyklem s daným návěštím.

Pokud je uveden v těle programu, kdekoliv mimo uvedené cykly ukončuje přímo program.

```
Syntaxe:
break <label>
```

Příklad:

```
:label while (1)
{        $a = something
          if ($a -eq 1) break <label>;
}
$a = $b +2
```

6.13.b) Continue

about_continue

Continue příkaz pokračuje v další iteraci cyklu bez jeho přerušení. Použití v příkazech For, Foreach, While.

Příklad:

```
while (1)
{               $a = something
       if ($a -eq 1) {continue}
       # This line is not reached unless $a == 1
}
#  Tato řádka není nikdy vykonána.
```

6.13.c) For

about_for

```
Syntaxe:
[:label] for ([initializer]; [condition]; [iterator]) {commands}
```

Příklad:

```
for ($i = 0; $i -lt 5; $i++) {Write-Output $i}
for (;;) # nekonečný cyklus
```

6.13.d) Foreach

about_foreach

```
Syntaxe:
[:label] foreach ($<item> in $<collection>){<statement list>}
<command> | foreach {<command_block>} # pro příkazovou řádku
<command> | foreach {<beginning command_block>}{<middle command_block>}{<ending
command_block>}
```

Příklad 18

```
$i = 1,2,3
foreach ($z in $i) {Write-Object $z}
Get-Process |foreach {BEGIN{$x=1}
       PROCESS{$X++}
       END{"$X Processes"}}
V objektu pipeline můžete použít znak % jako alias pro příkaz ForEach
```

6.13.e) Functions a Filters

about_functions
about_functions_advanced
about_functions_advanced_methods
about_functions_advanced_parameters
about_functions_cmdletbindingattribute

```
Syntaxe:
function [<scope-modifier:>]<name>
[([type]$parameter1[=DefaultValue][,[type]$parameter2[=DefaultValue]])]
           {
                param([type]$parameter1 [,[type]$parameter2])

                dynamicparam {<statement list>}

                begin {<statement list>}
                process {<statement list>}
                end {<statement list>}
           }
```

Filtry jsou zkrácenou cestou psaní funkcí v bloku skriptu PROCESS.

```
Syntaxe:
filter [<scope:>]<name> {<statement list>}
```

Jako parametr lze předávat i scriptblocs. Pokud předáváte více parametrů použijte pole – místo 1,2,3 použijte @(1,2,3).

Vestavěné funkce jsou také hodnoty v bytech – xkb, xmb, xgb, xtb, IPB, kde x nahradíte příslušným číslem.
Parametry mohou být pojmenované nebo poziční.
Parametr učiníte povinným pokud zadáte: Param([parameter(Mandatory=$true)]…

Lze též použít v jedné funkci rozdílné sady (sety) parametrů. Příklad použití vázání parametrů podle typu:

```
function Test-Binding
{
  #Content
  [CmdletBinding(DefaultParameterSetName='Number')]
  param
  (
    [String]
    [Parameter(Mandatory,ValueFromPipeline,ParameterSetName='Text', Position=0)]
    $Name,

    [int]
    [Parameter(Mandatory,ValueFromPipeline,ParameterSetName='Number',Position=0 )]
    $Length = -1
  )

  $set = $PSCmdlet.ParameterSetName
  "You called the function with the $set parameter set"
}
```

6.13.f) If/elseif/else

about_if

```
Syntaxe:
if (podmínka) {…}
elseif (podmínka) {…}
else {…}
```

Na příkazové řádce musí být složené závorky ve stejné řádce jako elseif and else. Toto omezení neplatí pro skripty.

6.13.g) Return

return příkaz ukončuje běžný skript a vrací hodnotu.

Příklad 19

```
Function foo ($param1, $param2){
       return 1
}
```

6.13.h) Switch

about_switch

Proměnná $_ je dostupná ve skriptech – reprezentuje vyhodnocování běžné hodnoty. Pokud je používáno ve switch poli, každý prvek pole je testován.

```
Syntaxe:
switch [-regex|-wildcard|-exact][-casesensitive] ( pipeline )
switch [-regex|-wildcard|-exact][-casesensitive] -file filename
    followed by
        {
            "string"|number|variable|{ expression } { statementlist }
            default { statementlist }
        }
```

Příklad 20

```
$var = "word1","word2","word3"
switch -regex ($var) {
"word1"  {"Multi-match Exact " + $_ }
"word2"  {"Multi-match Exact " + $_ }
"w.*2"   {"Pattern match Exact " + $_ }
default  {"Multi-match Default " + $_ }
}
```

A výstup:
Multi-match Exact word1
Multi-match Exact word2
Pattern match Exact word2
Multi-match Default word3

6.13.i)Until a While

about_do
about_while

Testování na počátku cyklu (cyklus nemusí proběhnout ani jednou):

```
Syntaxe:
```

```
[:label] while (condition)
{
…
}
do
```

Testování na konci cyklu, cyklus musí proběhnout alespoň jednou.

```
Syntaxe:
do
{
…
} until (condition)

do
{
…
} while (condition)
```

6.14. Řízení chyb a ladění

about_Debuggers
about_try_catch_finally

Ve WPS/PS7 jsou dva typy chyb: Terminating errors a Non-terminating errors (viz slovníček).

Pro zachytávání chyb se používají dva příkazy:

- příkaz Try – Catch – Finally:
  - určen pro vývojáře
  - pouze od verze V2, ve verzi 3 funguje trošku jinak
  - nezavádí nové rozsahy (scope)
  - hlídaný kód je v try bloku, nemá rozsah působnosti na celý příkaz try
  - podporuje finally
  - podporuje rethrowing výjimky
- příkaz Trap:
  - určen pro administrátory
  - ve verzi V1 i V2
  - zavádí nový rozsah
  - je "globální", což znamená, že se vztahuje na všechen kód ve stejném rozsahu, před nebo po.
  - nepodporuje rethrow (prázdný throw hodí speciální RuntimeException se zprávou "ScriptHalted")

A u cmdlets se používají parametry OutVariable a ErrorVariable a WarningVariable, ErrorActiona a WarningAction.

Proměnná parametrů OutVariable, WarningVariable a ErrorVariable fungují pouze v kontextu tokenu.

Debugger pro WPS/PS7 nepracuje v remote režimu, a tak musíte odlaďovat na lokálním počítači!

Číslo chyby na text chyby převedete jednoduše:

```
$error[0].tostring()
```

U Try-Catch-Finally dochází k jednomu problému, který je popsaný v následujícím výpisu, a proto doporučuji v sekci Try používat proměnné nikolivěk přímé hodnoty!

```
# Skript 1 - vyžaduje verzi WPS 2
# nastavíme proměnné

# Zkusíme dělení nulou přímou hodnotou
Try {
 1/0
}
# V tomto případě se vyvolá rovnou zpracování chyb .NET Frameworku
# a jak blok Catch, tak i Finally jsou ignorovány
Catch {
"Chyba při dělení - nelze dělit nulou"
echo "::Cannot handle the error (will rethrow): $_"
throw $_
}
Finally {
"Ohlídejte si dělitele!"
}

# Skript 2 - vyžaduje verzi WPS 2
# Budeme dělit nulou pomocí proměnných
$one = 1
$zero = 0
Try {
 $one/$zero
}
Catch {
"Chyba při dělení - nelze dělit nulou"
# Nyní je vyvoláno zpracování chyb .NET Frameworku, která je vyvolána až po ukončení bloku Try-Catch-
Finally
```

```
echo "::Cannot handle the error (will rethrow): $_"
throw $_
}
Finally {
"Ohlídejte si dělitele!"
}
```

Throw

Throw nabízí stejné funkce pro skripty jako ThrowTerminatingError API pro cmdlets.

```
throw "Danger, Danger"
Danger, Danger
At line:1 char:6
+ throw  <<<< "Danger, Danger"
```

Throw jako argumenty přijímá řetězce, výjimky nebo ErrorRecord.

(i) Traps

```
Trap [ExceptionType] {
    if (…)
{    continue
#  continue at the script statement after the one that cased the
#  trap; $? is updated but no error record is generated
} else  (…)
{      Break
#  rethrow the exception
}
#  Doing nothing will do what is specified in the
#  $ErrorActionPreference setting
}
```

6.15. Transakce

about_transaction

WPS/PS7 umožňuje transakce. Ten, kdo zná SQL, tak ví o co jde, pro ostatní: jedná se o to, že určitá posloupnost příkazů musí být vykonána celá v pořádku, a pokud se vyskytne chyba musí se všechny akce vrátit zpět na začátek transakce.

6.16. DATA sekce

about_data_sections

Skripty pro PWS od verze 2.0 mohou mít jednu či více datových sekcí, čímž oddělíte data od kódu a zlepšíte přehlednost, bezpečnost skriptu. Pokud je ve skriptu sekce DATA nebude spuštěn pod PWS verzí 1.0.

```
DATA <variable> [-supportedCommand <cmdlet-name>]
{
<Permitted content>
}
```

6.17. WMI Objekty (Get-WmiObject)

WMI je jeden z nejužitečnějších objektů pro reálnou práci. Pomocí WMI lze pracovat s místním i vzdáleným počítačem. Ovšem problém je, že má stovky tříd s více než desítkami vlastností…

Používejte parametr –Filter (tím následně do roury omezíte velikost kolekce) a –Query, který používá WQL (WQL je podmnožina jazyka SQL. Na rozdíl od Powershellu se jako divoký znak místo * používá % a místo –ne atd. se používá <> atd.

) příkazy a čímž si selektujete už v první fázi.

POZNÁMKA 7: Ke zjištění tříd a získání hrubého kódu používejte PowerShellScriptOMatic, následně použijte Get-Member

Pokud používáte Get-WMIObject pro připojení na vzdálený počítač, pak vzdálený počítač musí mít spuštěno WMI a v implicitní konfiguraci musí jeho skupina místních administrátorů obsahovat účet pod kterým ve WMI pracujete. Důležité je, že vzdálený systém nemusí mít instalován WPS/PS7! To znamená, že můžete spravovat vzdálené systémy, které nemají nainstalován WPS/PS7, ale mají dostupné WMI.

Jako název počítače můžete používat:

```
.      (tečka - jako lokální počítač)
IP adresa
jméno počítače
```

WPS/PS7 zajišťuje přístup k metodám a vlastnostem objektů WMI, XML, ASDI, ADO, a COM.

Příklad 21

```
$g = Get-WmiObject Win32_Process
$g[0].Name  # zkráceně místo $g[0].Properties["Name"]
```

POZNÁMKA 8: Get-WmiObject používá root/cimv2 namespace jako implicitní namespace.

6.17.a) WQL

WQL je podmnožina jazyka SQL. Na rozdíl od Powershellu se jako divoký znak místo * používá % a místo –ne atd. se používá <> atd.

6.18. .NET a COM Objekty

K dispozici ve WPS/PS7 jsou také komponenty.NET Framework a COM objekty, které rozšiřují možnosti cmdlets. Některé cmdlets neumí pracovat na vzdálených počítačích. To se dá obejít při řízení událostí pomocí. NET System.Diagnostics.EventLog třídy přímo z WPS/PS7. Tímto způsobem lze pracovat s WHS a VBS.

6.18.a) K vytvoření objektu COM slouží cmdlets:

```
new-object –ComObject
```

6.18.b) K vytvoření objektu .NET:

```
[určení třídy]::metoda(parametry)
```

Lze volat pouze statické metody!

Příklad 22

```
[System.DateTime]::IsLeapYear(2008)
```

lze psát i zkráceně (System je předpokládán automaticky)

```
[DateTime]::IsLeapYear(2008)
```

K odkazu na .NET objekt:

```
new-object –type určení třídy –argumentlist argumenty
```

Statické třídy zjistíte příkazem Get-Member –Static např.:

```
[math] | Get-Member -Static
```

7. NAVIGACE VE WPS/PS7 – PSDRIVE

about_providers

Jedna z nejsilnějších vlastností WPS/PS7 je, že pomocí stejných příkazů můžete pracovat se všemi datovými úložišti (pro každý typ jednotky je používán jiný provider):

Provider	Odkaz	Popis
Active Directory	AD:	Active Directory. Musíte mít naimportován modul pro práci s AD.
FileSystem	C:, D:, atd.	FileSystem. Jednotlivé disky. WPS detekuje systémové jednotky přidané nebo vyjmuté ve Windows, stejně tak namapované síťové disky, připojené USB jednotky a jednotky vyjmuté příkazy net use nebo WScript.NetworkMapNetworkDrive and RemoveNetworkDrive z Windows Script Host (WSH).
Registry	HKCU: HKLM:	Registry (nelze ovšem pracovat se všemi větvemi registru). Pokud chceme zjistit požadované informace, musíme použít Get-ItemProperty, protože hodnoty v určitém klíči či podklíči jsou totiž považovány za vlastnosti tohoto klíče. Viz Get-Help registry. Pokud chcete vložit novou hodnotu, pak zase použít New-ItemProperty.
Alias	Alias:	Aliasy
Certificate	Cert:	Certifikáty.
Variable	Variable:	Proměnné implicitní i vámi definované.
Environment	Env:	Proměnné prostředí systému a WPS/PS7.
Function	Function	Funkce.
WSMan	WSMan	Poskytuje přístup ke konfiguračním informacím Web Services for Management (WS-Management).

symbol pro běžný adresář: . (tečka)

obsah adresáře: * (hvězdička)

používají se wildcard tak jak je obvyklé:

***** (žádný nebo více znaků)

? (jeden znak)

[] (sada odpovídajících znaků).

Zjištění providerů (jednoduché příklady):

```
Get-PSdrive
```

Nastavení jednotky:

```
Set-Location WSMan:
```

Zjištění informací o jednotce:

```
Get-Location –PSProvider Variable
```

vytvoření vlastního WPS/PS7 providera pomocí příkazu:

```
New-PSDrive -Root c:\temp\pokus -Name MyProvider -PSProvider FileSystem
```

Vyjmutí vlastního providera:

```
Remove-PSDrive MyProvider
```

sledování historie procházení jednotkami:

```
Push-Location
Pop-Location
```

přímá manipulace s Items (při práci s položkami PSDrive můžete používat Wildcard. Důležité paramatry jsou Include a Exclude. Pokud použijete Include musíte použít i parametr Recurse. Viz:

```
Get-Command -Noun Item
```

Příklad 23

Máme za úkol vytvořit adresářovou strukturu pro nově vzniklá oddělení. Potřebujeme adresáře od písmena A do Z a pro tyto adresáře musíme nastavit předem daná práva (různá dle oddělení). V Exploreru noční můra, ve WPS/PS7 zábava. Všimněte si i použití ostatních cmdletů určených pro práci s PSDrives:

```
# Vytvoření nového adresáře
New-Item -Path C:\temp\pokus -ItemType directory
# Vytvoření nového PSDrive typu filesystem a přechod do něj
New-PSDrive -Root C:\temp\pokus -Name pokus -PSProvider FileSystem
# následující dva cmdlety
# Push-Location
# Set-Location pokus:
# lze nahradit jediným cmdletem
Push-Location -Path pokus:
# Vytvoření adresářů
97..122 | % {New-Item -name ([char]$_) -ItemType directory}
# vytvoření poddadresářů v každém výše vytvořeném adresáři
Get-ChildItem | % {New-Item -ItemType directory -Path (join-path -Path $_ -ChildPath
"Private")}
Get-ChildItem | % {New-Item -ItemType directory -Path (join-path -Path $_ -ChildPath
"Public")}
```

```
# Nyní si vytvoříme objekty s právy podle vzorových adresářů
$aclPrivate = get-acl C:\temp\private
$aclPublic = get-acl C:\temp\public
# Zobrazíme práva
get-acl C:\temp\private | Format-List
get-acl C:\temp\public | Format-List
# Nastavíme práva na adresáře
Get-ChildItem -Recurse | ? {$_.fullname -match "private"} | Set-Acl -aclobject
$aclPrivate
Get-ChildItem -Recurse | ? {$_.fullname -match "public"} | Set-Acl -aclobject
$aclPublic
# přesuneme se zpět do původního adresáře
Pop-Location
# A uklidíme po sobě
Remove-Item -Recurse -Path C:\temp\pokus
Remove-PSDrive -Name pokus
```

Cmdlet pro náhradu dvojkliku spustí implicitní akci nad objektem (chová se stejně jako dvojklik myší v exploreru):

```
Invoke-Item
```

Příklad 24

Otevře soubor myfile.txt v aplikaci sdružené s příponou txt – obvykle notepad (takto můžete otevřít jakýkoliv typ souboru):

```
invoke-item c:\work\myfile.txt
```

Postupně otevře v notepadu všechny soubory txt v běžném adresáři:

```
invoke-item *.txt
dir *.txt | invoke-item
```

Otevře soubory podle seznamu v souboru files.txt:

```
get-content c:\files.txt | invoke-item
```

Spustí kalkulátor:

```
invoke-item "c:\windows\system32\calc.exe"
```

Otevře v okně exploreru adresář temp:

```
Invoke-item $env:temp
```

8. REMOTE SCRIPTING

```
about_Remote.help.txt
about_Remote_Disconnected_Sessions.help.txt
about_Remote_FAQ.help.txt
about_Remote_Jobs.help.txt
about_Remote_Output.help.txt
about_Remote_Requirements.help.txt
about_Remote_Troubleshooting.help.txt
about_Remote_Variables.help.txt
about_Session_Configuration_Files.help.txt
about_Session_Configurations.help.txt
WS-Management Protocol
```

8.1. Úvod

Remotingem se myslí to, že pracujete ze svého místního počítače přímo na vzdáleném počítači nebo počítačích. Pro omezený remoting (bez potřeby konfigurace vzdáleného počítače) můžete použít nativní cmdlets (cmdlets, které mohou běžet přímo na vzdáleném počítači – mají parametr ComputerName). Pro neomezený remoting musíte konfigurovat vzdálený počítač – můžete tak používat všechny cmdlety a svoje skripty.

POZNÁMKA 9: Samozřejmě, jediné, co nejde při remotingu, je spouštění programů s GUI, sice je spustíte, ale nevidíte jejich okno. Pokud potřebujete pracovat s GUI, pak použijte vzdálenou plochu nebo jiný podobný nástroj.

8.2. Konfigurace remotingu

Snažte se mít na všech počítačích stejnou verzi WPS/PS7. Můžete sice vytvářet remote sessions mezi počítači s různými verzemi WPS/PS7, ale vlastnosti dostupné ve vyšších verzích WPS/PS7 nespustíte na nižší verzi (samozřejmě WPS/PS7 je zpětně kompatibilní). Všechna nastavení musíte provádět jako administrátor (Run as administrator)!

8.2.a) Jak zjistíte verzi Powershellu?

`$PSVersionTable.PSVersion` **nebo** `$host.Version`

8.2.b) Systémové požadavky

Windows PowerShell System Requirements

Požadavky jsou zde uvedeny z hlediska Windows 7 SP1/Windows Server 2008 R2 SP1/Windows Server 2008 SP2.

- Ve Windows 8/Windows Server 2012 je instalován implicitně WPS 3.0.
- Ve Windows 8.1/Windows Server 2012 R2 je instalován implicitně WPS 4.0.
- Ve Windows 10/Windows Server 2016 je instalován implicitně WPS 5.0.

WPS 2.0

- Microsoft .NET Framework 2.0 nebo novější
- Windows Management Framework 2.0

WPS 3.0

- Microsoft .NET Framework 4.0 nebo novější
- Windows Management Framework 3.0

WPS 4.0

- Microsoft .NET Framework 4.5 nebo novější
- Windows Management Framework 4.0

8.2.c) Oprávnění

Pro vytváření remote sessions a spouštění vzdálených příkazů musíte být buď členem skupiny Administrators na vzdáleném počítači nebo musíte mít oprávnění administrátora. Pověření jsou šifrována. Pokud chcete zvýšit zabezpečení, nastavte WinRM na HTTPS. Pokud se připojíte ke vzdálenému počítači, pak platí politiky vzdáleného počítače!

8.2.d) Konfigurace vzdáleného počítače pro remoting

Pokud budete používat pouze nativní cmdlety, není potřeba konfigurovat vzdálený počítač. Pokud však chcete používat session a všechny příkazy WPS/PS7, pak vzdálený počítač musíte konfigurovat.

U Windows Server 2012 a vyšší je remoting povolen implicitně. Remoting se dá povolit dvěma způsoby (oba musíte spustit na vzdáleném počítači jako administrátor):

- Pro jeden nebo pár počítačů – spusťte na daném počítači WPS/PS7 jako administrátor a zadejte cmdlet:
  Enable-PSRemoting nebo Enable-PSRemoting -Force (bez dotazů).
- V příkazovém řádku spusťte jako administrátor (a to je podle mého lepší způsob) a potvrďte dvakrát yes:
  WinRM quickconfig

Pro více počítačů v doménovém prostředí pomocí Group Policy povolte:

"Allow automatic configuration of listeners" v Computer Configuration\Administrative Templates\Windows Components\Windows Remote Management (WinRM)\WinRM service

Potom specifikujte IPv4 a IPv6 filtry (wildcards jsou povoleny).

Příkazem `New-PSSession` (viz dále) si ověřte, že je remoting nastaven správně.

8.3. Základní pojmy

8.3.a) Nativní cmdlets pro remoting

Nativní cmdlety pro remoting nepotřebují konfiguraci vzdáleného počítače pro remoting (samozřejmě to nejsou všechny cmdlety WPS/PS7!). Tyto cmdlety získávají informace ze vzdáleného počítače pomocí Microsoft .NET Framework. Nativní cmdlety pro remoting, které umožňují přistupovat přímo na vzdálený počítač zjistíte snadno:

```
# Cmdlety, které umožňují remoting bez speciální konfigurace mají parametr ComputerName
a nemají session parametr
Get-Command | where { $_.Parameters.Keys -contains "ComputerName" -and
$_.Parameters.Keys -NotContains "Session"}
# ale ono stačí zadat jen
Get-Command -ParameterName ComputerName
```

Většinou se v parametru ComputerName zadává jméno jednoho počítače, lze však i více – názvy počítačů oddělujte čárkou. Lze samozřejmě předávat jména počítačů pomocí roury, ale pozor – jen u cmdlets, které mají v nápovědě k danému cmdletu uvedeno True u hodnoty Accept pipeline input! Pokud je tento parametr nastaven na False, pak v případě použití více počítačů použijte proměnnou, do které uložíte názvy počítačů nebo si do proměnné načtěte obsah textového souboru s názvy počítačů.

8.3.b) Session

Session je uzavřené prostředí, ve kterém můžete pracovat se vzdáleným počítačem a využívat všech možností WPS/PS7. Počet session není teoreticky omezen, ale samozřejmě každá session spotřebovává zdroje. Implicitně je nastaveno maximálně 32 současných session, ale to si můžete změnit parametrem ThrottleLimit. Cmdlety, které tento parametr podporují najdete snadno:

```
Get-Command -ParameterName ThrottleLimit
```

Nativní cmdlets nepoužívají session! Session mohou být dočasné (temporary), interaktivní (interactive) a trvalé (persistent).

(i) Temporary session

Vznikají u cmdletu `Invoke-Command` s parametrem Computername. WPS/PS7 zřídí připojení pouze pro tento cmdlet a po ukončení cmdletu toto připojení uzavře. Jakékoliv proměnné nebo funkce, jež byly v rámci tohoto cmdletu definovány se ztratí.

Invoke cmdlet

`Invoke-Command` je cmdlet, který umožňuje spustit příkaz, blok příkazů nebo script na vzdáleném (nebo i místním samozřejmě) počítači či počítačích a vrací výsledek ze vzdáleného počítače do vaší konzole. To znamená, že nepracujete přímo na vzdáleném počítači, jen dostanete zpátky požadovaná data.

Pomocí `Invoke-Command` také můžete spustit příkaz, blok příkazů či skript, který máte na lokálním počítači. Potřebujete-li spustit lokální skript na vzdáleném počítači, použijte parametr FilePath. Výsledek je vrácen do vašeho lokálního počítače.

Tento cmdlet v případě použití parametru ComputerName si otevře temporary session a po ukončení si ji zase sám zavře, aniž byste se museli o nějakou session starat. Znamená to, že všechny proměnné ze vzdáleného počítače či počítačů se ztratí.

ComputerName parametr tedy slouží v situacích, kdy potřebujete spustit jeden nebo více nesouvisejících příkazů na jednom nebo více počítačích a nepotřebujete uchovávat funkce a proměnné. Pokud chcete použít cmdlet Invoke-Command na místním počítači, uveďte do parametru ComputerName hodnotu . (tečka) nebo "localhost".

Session parametr slouží v situacích, kdy potřebujete sdílet data.

(ii) Interactive session

V rámci této interaktivní session se můžete připojit pouze k jednomu vzdálenému počítači. Příkazy, jež zadáváte, jsou spuštěny přímo na vzdáleném počítači, stejně jako byste je zadávali přímo na něm - v rámci interaktivní session zadáváte přímo příkazy do příkazového řádku a pracujete tak přímo na vzdáleném počítači.

Cmdlets pro interaktivní session:

```
# Zahájení interactive session
Enter-PSSession Server1
Server1\PS> … a zde můžete zadávat libovolné příkazy (jste přímo na vzdáleném počítači)
# Ukončení interactive session
Exit-PSSession
```

(iii) Persistent session

Persistent session je trvalá session, na kterou se můžete odkazovat v cmdlets (tato session je umístěna na straně remote počítače). Pomocí persistent session můžete spustit sérii vzdálených příkazů, jež sdílejí data, jako jsou funkce, aliasy, proměnné. Pokud vytvoříte persistent session pro více počítačů, pak se příkazy spouští pro každý uvedený počítač.

Ve WPS 2.0 při odpojení session je tato session odstraněna i ze vzdáleného počítače. Od WPS verze 3.0 a vyšší (musí být stejná verze WPS/PS7 samozřejmě na místním i vzdáleném počítači) je session ukládána na vzdáleném počítači, a tak se můžete od persistent session odpojit a později z libovolného počítače opět připojit. To je výhoda, protože si můžete svůj počítač restartovat a pak se zase na session připojit nebo se připojit na session třeba odjinud. Pro Interactive session to neplatí!

Odpojené relace jsou udržovány v odpojeném stavu, dokud neodstraníte PSSession na vzdáleném počítači např. použitím cmdlet `Remove-PSSession`.

Příkazy v odpojeném persistent session pokračují tedy i po odpojení, do chvíle, až se ukončí sekvence příkazů či skript, nebo až se ukončí session v okamžiku zaplnění výstupního bufferu nebo dojde k vypršení časového limitu:

- Abyste zabránili zaplnění výstupního bufferu v případě odpojení od session použijte parametr OutputBufferingMode v cmdlets Disconnect-PSSession, New-PSSessionOption, nebo New-PSTransportOption cmdlets.
- Můžete nastavit časové prodlevy PSSession pomocí IdleTimeoutSec nebo idleTimeout parametrů u cmdlets Disconnect-PSSession, New-PSSessionOption, nebo New-PSTransportOption.

Základní cmdlets pro persistent sessions

`Disconnect-PSSession`	Odpojí PSSession.
`Connect-PSSession`	Připojí PSSession.
`Receive-PSSession`	Získá výsledky výsledky příkazů, které byly spuštěny v odpojené relaci.
`Get-PSSession`	Získá seznam PSSessions na místním počítači.
`Get-PSSession -ComputerName <nazev>`	Získá seznam PSSessions na vzdáleném počítači.

Invoke-Command InDisconnectedSession parametr vytváří PSSession a session přímo odpojí.

```
Zahájení persistent session
# Zahájení persistent session
$pss = New-PSSession –ComputerName Server01
Odpojení od persistent session
# Odpojení od session
Get-PSSession -ComputerName Server01 | Disconnect-PSSession
Připojení k persistent session
# Připojení k session na vzdáleném počítači
Get-PSSession -ComputerName Server1 | Connect-PSSession
# Na místním počítači získáme jména session
Get-PSSession
# A následně se připojíme k dané místní session
Connect-PSSession -Name <jméno session>
Ukončení persistent session
# Ukončení a odstranění persistent session
Remove-PSSession $pss
```

POZNÁMKA 10:

- Cmdlet New-PSSession musí mít uveden parametr ComputerName (pro local stačí .), protože jinak vám bude hlášena chyba, že příslušná funkce nebo cmdlet neexistuje!
- Pokud vytvoříte persistent session z klienta (např. MYPC) na vzdáleném počítači (např. MYSRV1), pak přímo ve WPS/PS7 na MYSRV1 tuto session příkazem Get-Sessions nevidíte! Musíte na MYSRV1 použít: Get-Sessions -Computername MYSRV1.
- Pokud se připojujete k session z jiného počítače, musíte si ji nejdříve odpojit (pokud je ve stavu open), abyste se k ní mohli připojit.
- Pokud se na jiném počítači připojíte ke vzdálené session, bude mít jiné ID – to si přiřazuje client!
- Ve výpisu o stavu session je důležitý sloupec Availability – pokud je Busy, znamená to, že je session připojená na jiném počítači, pokud je None je odpojená, pokud je Availability je připojená k danému počítači.

8.3.c) Remote proměnné

Proměnné použité v příkazech pro remoting jsou definované v dané session. Nelze sdílet proměnné v různých session.

Pokud chcete v příkazech pro remoting použít místní proměnné, tak u WPS verze 2.0 musíte použít parametr ArgumentList a u verze 3.0 a vyšší můžete jednodušeji použít modifikátor rozsahu Using:

```
C:\PS>$ps = "Windows PowerShell"

# Windows Powershell 2.0
C:\PS>Invoke-Command -ComputerName S1 -ScriptBlock {param($log) Get-WinEvent -logname
$log} -ArgumentList $ps

# Windows Powershell 3.0 a vyšší
# Syntaxe: $Using:<VariableName>
# Použití: $Using:$ps

C:\PS>Invoke-Command -ComputerName S1 -ScriptBlock {param($log) Get-WinEvent -logname
$log} -ArgumentList $Using:$ps
```

8.4. Shrnutí

8.4.a) Stručný postup pro zprovoznění remotingu

(podrobně v kapitole Konfigurace remotingu):

- Nainstalujte si na všech počítačích, ke kterým budete přistupovat stejnou verzi Windows Management Framework (obsahuje WPS) a k tomu příslušný .NET Framework - (obojí ke stažení na Microsoft Download Center). Doporučuji vám, pokud chcete přistupovat na vzdálený počítač pomocí WPS, aby na všech počítačích byla stejná verze WPS. Přece jen mezi jednotlivými verzemi Powershellu jsou rozdíly a vyšší verze umožňují lepší "fičury".
- Na všech počítačích povolte remoting, ať už ručně nebo pomocí Group Policy.
- Udělte práva uživatelům, kteří budou moci používat remoting.

(i) Co se k čemu hodí:

- Potřebujete spustit cmdlet na vzdáleném počítači a nemáte provedenou konfiguraci pro remoting – použijte nativní cmdlet pro remoting.
- Potřebujete spustit příkaz nebo skript na jednom či více vzdálených počítačích nakonfigurovaných pro remoting, aniž byste potřebovali s výstupem dále pracovat – použijte Invoke-Command.
- Potřebujete přímo pracovat na daném vzdáleném počítači nakonfigurovaném pro remoting – použijte interactive session.
- Potřebujete pracovat současně na více vzdálených počítačích nakonfigurovaných pro remoting – použijte persistent session.

Pro přerušení příkazu použijte CTRL+C.

9. WORKFLOW (PRACOVNÍ POSTUP)

```
About_Jobs
about_Workflow_Common_Parameters
about_WorkflowCommonParameters
about_Workflows
about_Functions_CmdletBindingAttribute
```

Getting Started with Windows PowerShell Workflow
Writing a Script Workflow
Workflow Authoring Reference Topics
Syntactic Differences Between Script Workflows and Scripts

Upozornění:

- Soubory about_Workflow* jsou v adresáři C:\Windows\System32\WindowsPowerShell\v1.0\Modules\PSWorkflow\en-US.
- Doporučuji si zopakovat kapitolu Remote scripting a osvěžit si informace o session.
- Vysvětlivky pojmů jsou uvedeny na začátku tohoto dokumentu.

9.1. Úvod

Od verze WPS 3.0 do verze WPS 5.1 (v PS7 není funkční) je umožněno vytvořit funkci workflow, která umožňuje provádět paralelní běh různých úloh (například potřebujete nakopírovat soubory na klienty, pak tyto soubory spustit a instalovat) – těmto úlohám se říká activities. Activity je vlastně konkrétní úkol, který má workflow provést. Workflow se může skládat z jedné nebo více activities, může být vyvoláván v jiném skriptu anebo může vyvolávat další workflow.

POZNÁMKA 11: V rámci activities se dá určit pořadí a priorita příkazů. Počítačům, na kterých se má workflow provést se zde říká managed node. Tím, že workflow je navržen pro spuštění na více manage nodes současně, není třeba vytvářet remote session nebo používat remote cmdlets. Jednoduše jde o to, že administrátoři potřebují často provést stejný úkol na více počítačích (nodes). Pokud ho provedete klasickým skriptem, který postupuje sekvenčně po jednotlivých node, trvá to neúměrně dlouho, obzvlášť, když je některý node vypnutý. Workflow vám umožňuje stejnou sekvenci příkazů vykonat paralelně (současně, souběžně) na více nodes. Protože se příkazy vykonávají na všech nodes prakticky současně, doba běhu skriptu odpovídá „přibližně“ běhu skriptu na nejpomalejším node.

9.1.a) Workflow je vhodný

- Při provedení dlouhotrvajícího úkolu, sestaveného z více kroků na jednom node.
- Při provedení stejného či dlouhotrvajícího úkolu na více nodes (mohou to být stovky nodes) nebo v prostředí vysoké dostupnosti.
- Při potřebě sledovat a kontrolovat průběh úkolu (workflow).
- Při provedení asynchronní dlouhotrvající activity, která může být plánovaně i neplánovaně přerušena (restartem node, vypnutím proudu, odpojením od sítě atp.).

9.1.b) Výhody WPS Workflow

- Použití syntaxe WPS – je to vlastně funkce (s drobnými rozdíly oproti standardní funkci WPS).
- Správa více nodes současně paralelně (souběžně).
- V rámci jednoho workflow můžete kombinovat jiné workflow, skripty či funkce.
- Stav a průběh activities je viditelný kdykoliv a z kteréhokoliv node.
- Automatické zotavení – workflow přežije plánované i neplánované přerušení (třeba restart počítače). Lze pokračovat od bodu, kdy byla activity pozastavena nebo od checkpointu.
- Při výskytu síťové chyby se workflow může pokoušet o opětné připojení node. V rámci workflow můžete určit activities, které se mají spustit znovu po selhání workflow.
- Sledování workflow můžete na jednom počítači ukončit a pokračovat na jiném. Workflow zůstává spuštěný, i když se managed node restartuje (viz persistent session v kapitole Remote scripting).
- Workflow lze spouštět pomocí scheduleru (plánovače úloh), stejně jako ostatní WPS skripty.

9.2. Plánování workflow

- Vytvořte si základní představu o syntaktických rozdílech mezi WPS a workflow a také o možnostech workflow.
- Vyhodnoťte activities, jež bude workflow provádět. Označte části, které můžou běžet souběžně nebo nemusí běžet v předurčeném pořadí. Uvnitř těchto částí označte části, které se musí provádět sekvenčně – tedy v přesně daném pořadí.
- Uspořádejte activities do funkcí v rámci workflow. Použijte pro každou activity již existující cmdlets nebo vytvořte nové příkazy.
- Přidejte checkpointy po každém kritickém kroku, nejlépe po každém příkazu či funkci. Po dokončení definice workflow pomocí procesu testování zpřesněte checkpointy – přidejte nebo odeberte checkpointy (nezapomeňte na komentáře!) k dosažení přiměřené provozní doby a přiměřené doby obnovení.
- Zvažte prostředí, ve kterém bude workflow běžet (operační systém, verze WPS atp.).
- Vytvářejte nápovědu pro workflow – komentujte svoje příkazy (až se ke skriptu vrátíte za půl roku, rok, pak bez nápovědy budete muset hodně přemýšlet, jak jste to v tom skriptu sakra vymysleli).

9.3. Požadavky na HW a SW pro workflow server

WPS Workflow běží na všech systémech, na kterých je dostupné prostředí WPS 3.0 nebo WPS 4.0. Úplný seznam požadavků na WPS najdete v článku **Požadavky na systém pro Windows PowerShell**.

Pokud vaše workflow obsahují activities modelu CIM, stáhněte a nainstalujte na managed nodes **Windows Management Framework 3.0** nebo **Windows Management Framework 4.0**. WMF můžete taky nainstalovat na managed nodes se systémem Windows Server 2008 R2 s aktualizací

Service Pack 1, Windows 7 s aktualizací Service Pack 1, případně Windows Management Framework 3.0 na uzly se systémem Windows Server 2008 s aktualizací Service Pack 2, pokud chcete workflow spouštět na těchto počítačích. Není to ale potřeba, pokud vaše workflow neobsahují activities modelu CIM.

POZNÁMKA 12: Na managed nodes se tedy nevyžaduje WPS ani WMF, pokud workflow neobsahuje activities modelu CIM.

9.4. Možné konfigurace workflow při vlastním provozu

- Workflow server, workflow klient a managed nodes se provozují každý na samostatných počítačích: Příkazy procházejí od workflow klienta přes workflow server (kde je relace) až na managed nodes. V tomto scénáři musí workflow klient používat alespoň WPS 2.0, aby mohl komunikovat s workflow serverem (počítačem, který spouští workflow).
- Workflow server a workflow klient se provozují na stejném počítači: Verze prostředí WPS a rozhraní .NET Framework musí splňovat nejvyšší požadavky pro workflow server, na kterém běží workflow. Managed nodes se provozují na jednom nebo několika vzdálených zařízeních, která nepotřebují WPS ani WMF, pokud workflow neobsahuje activities modelu CIM.
- Workflow server, workflow klient a managed nodes se provozují na stejném počítači nebo zařízení: Verze prostředí WPS a rozhraní .NET Framework na tomto počítači musí splňovat nejvyšší požadavky pro workflow server.

9.5. Příprava počítačů ke spouštění workflow

9.5.a) Vlastní příprava

Příprava workflow server (spusťte WPS jako správce nebo cmdletem `Start-Process Powershell -verb RunAs`):

- Na Windows Server 2012 R2 nebo Windows Server 2012 je vzdálená správa WPS povolena implicitně.
- Na Windows 8 a vyšší zadejte:
  Enable-PSRemoting.
- Na Windows Serveru 2008 R2 nebo Windows 7 (musí být nainstalován WMF 3.0 nebo 4.0) zadejte:
  Enable-PSRemoting.
- Na Windows Serveru 2008 (musí být nainstalován WMF 3.0) zadejte:
- Enable-PSRemoting.

9.5.b) Konfigurace relace workflow

`about_Session_Configurations`

Konfigurace relace workflow je skupina nastavení na workflow serveru, která definuje přístupová práva a příkazy dostupné pro relace prostředí WPS, které se vytvářejí, když se vzdálení nebo místní uživatelé připojují k prostředí WPS na nějakém počítači. Příkaz Enable-PSRemoting vytváří implicitní konfiguraci relace, která poskytuje tyto výhody:

- Optimalizaci výkonu: Výchozí nastavení rutiny New-PSWorkflowExecutionOption (například časové limity activities a maximální počet povolených procesů na activity) jsou navržená tak, aby omezila využití prostředků a vylepšila výkon workflow.
- Režim SharedHost: Režim SharedHost umožňuje uživatelům spustit workflow na jednom počítači, odhlásit se, přihlásit se k jinému počítači a znovu se k tomuto workflow připojit. Konfigurace relace workflow používá stejný proces pro uživatele na různých počítačích, pokud se uživatel připojuje ke stejné konfiguraci relace a k novému počítači je přihlášený se stejnými přístupovými právy, jaká měl na prvním počítači.
- Automatická podpora několika zařízení: Konfigurace relace workflow je optimalizovaná pro spouštění workflows na několika zařízeních a nabízí následující výhody:
- Úlohy workflow se vytvářejí automaticky pro každé cílové zařízení.
- WPS workflow automaticky omezuje počet souběžných workflow a vzdálených operací, které můžou běžet na cílovém zařízení.
- Spravuje životní cyklus vzdálených připojení, do kterého patří vytvoření, opakované použití a odstranění.

Můžete si následujícím způsobem vytvořit vlastní konfiguraci relace (**není to Microsoftem doporučováno**):

- Na workflow serveru spusťte relaci prostředí WPS jako správce.
- U relací, ve kterých se používá konfigurace relace pro workflow, doporučujeme zvýšit výchozí kvótu paměti WinRM (Windows Remote Management) na 1 GB. To uděláte tak, že spuštěním následujícího příkazu vytvoříte objekt parametru konfigurace relace, který pak uložíte do proměnné $MoreMemory (tato proměnná se použije v dalším kroku): $MoreMemory `= New-PSSessionConfigurationOption -MaxMemoryPerShellMB 1000`
- Spuštěním rutiny `Register-PSSessionConfiguration` přidejte novou konfiguraci relace:
- Hodnotu parametru `SessionConfigurationType` nastavte na Workflow.
- Hodnotu parametru `ModulePath` nastavte na umístění modulů, které obsahují vaše workflow.
- Hodnotu parametru `SessionConfigurationOption` nastavte na proměnnou $MoreMemory, kterou jste vytvořili v předchozím kroku.

9.6. Spuštění WPS Workflow

Vzor workflow funkce používané v následujícím textu:

```
Import-Module PSWorkflow # můžete spustit i samostatně
workflow test-Workflow
{
    Get-Process -Name iexplore
}
```

Workflow se spouští stejně, jako kterýkoli jiná funkce prostředí WPS. Protože je to v podstatě funkce, musíte workflow nejprve zavést stejně jako funkci v profilu při spuštění WPS nebo přes dot sourcing (nezapomeňte na mezeru mezi tečkami!):

```
. .\test-workflow.ps1
```

Pokud chcete najít dostupná workflows, spusťte:

```
Get-Command -CommandType workflow
```

Protokoly chyb pro workflow se ukládají na workflow serveru - událost s ID 45079 zobrazuje každou spuštěnou activity. Můžete je najít v následujícím kanálu: Trasování událostí pro Windows (Event Tracing for Windows - ETW) nebo cestě modulu snap-in Prohlížeče událostí.

POZNÁMKA 13: WPS Workflow je persistent session (ale s nějakými rozdíly, jak uvidíte). Pokud si prostudujete kapitolu Remote scripting ulehčí vám to pochopení toho, jak WPS workflow funguje.

9.6.a) Jednoduché spuštění workflow bez relace

Pokud potřebujete spustit jednoduchý workflow, u kterého nechcete nic sledovat a nepotřebujete odpojovat se a připojovat z jiných počítačů.

Když chcete spustit workflow (workflow funkce musí být zavedena), stačí k tomu jenom zadat jeho název ve tvaru:

```
sloveso-podstatné jméno
```

nebo

```
sloveso-podstatné jméno -PsComputerName MN1,MN2
```

Případný výpis workflow je znázorněn na obrazovce přímo. Nezakládá se session ani job.

```
Příklad: test-workflow -PSComputerName MN1,MN2
```

9.6.b) Spuštění workflow v relaci

(i) Důležité upozornění pro workflow v relaci

Při studiu workflow z materiálů Microsoftu jsem narazil na pár nesrovnalostí, které jsem nevyřešil ani pomocí strýčka Google.

Workflow session zřejmě pracuje trochu jinak než než persistent session! Alespoň u mne ve WPS verze 4.0. Uvádím posloupnost příkazů podle jejich příruček a následně problém a jeho vyřešení:

```
1) . .\test-workflow.ps1
2) $workflowsession = New-PSWorkflowSession -Computername MN1
3) Invoke-Command $workflowsession {test-workflow}
```

1. Zavede se funkce test-workflow (můžete ji použít jako běžnou funkci, ne však ve workflow relaci jak je psáno v manuálech!
2. Spustí workflow session, která je fyzicky umístěna na počítači MN1 (to je workflow server).
3. Je nahlášeno, že test-workflow process je neznámý.

Pokud místo workflow session použijete normální persistent session, pak je vše v pořádku a uvedená posloupnost příkazů funguje, tak jak má! A řešení? Musíte po řádku 2 (spuštění workflow relace) spustit ještě následující příkaz (proč – nevím, ale funguje to):

```
Invoke-Command $workflowsession -FilePath .\test-workflow.ps1
```

Tím se v rámci session zavede funkce test-workflow… Takže funkční proces bude vypadat takto:

```
$workflowsession = New-PSWorkflowSession -Computername .
. .\test-workflow.ps1
Invoke-Command $workflowsession -FilePath .\test-workflow.ps1
Invoke-Command $workflowsession {test-workflow}
```

Je ještě jedno řešení, a to, že ve ScriptBlock uvedete celou funkci Workflow – tím ji nemusíte zavádět a funguje to podle MS návodu. Hodí se to však pouze pro jednoduché workflow funkce. Nebo pomocí jednoduchého workflow tak můžete vyvolat nějakou složitější funkci:

```
Invoke-Command $workflowsession {workflow testworkflow {Get-Process -Name iexplore}}
```

POZNÁMKA 14: Session a job(s) se vytváří fyzicky na každém managed node(s)!

(ii) Spuštění workflow v relaci bez jobs

Vytvoření nové relace workflow:

- Spusťte WPS jako správce. Pokud se připojujete ze vzdáleného počítače, nemusíte WPS spouštět se zvýšenými oprávněními.
- Spusťte vzdálenou relaci prostředí WPS, která je připojená k počítači (workflow serveru), na kterém chcete spustit svůj workflow, a uložte si tuto relaci do proměnné. Tato relace může být
  na vašem místním počítači (localhost):
  Relace workflow na místním počítači (uloží danou relaci do proměnné $workflowsession - relace je nakonfigurovaná tak, aby používala výchozí konfiguraci relace workflow - pomocí rutiny New-PSWorkflowSession se relace nakonfiguruje tak, aby používala výchozí konfiguraci relace workflow):
  $workflowsession = New-PSWorkflowSession -ComputerName .
  nebo na vzdáleném počítači:
  Relace workflow na počítači Server01 (uloží danou relaci do proměnné $workflowsession):
  $workflowsession = New-PSWorkflowSession -ComputerName Server01 -credential DomainName\UserName
  Vzhledem k tomu, že relace serveru workflowu běží na vzdáleném počítači, musí uživatel zadat explicitní přihlašovací údaje. Explicitní přihlašovací údaje se nevyžadují, pokud konfigurace relace zahrnuje funkci RunAs, kde všichni uživatelé sdílí sadu přihlašovacích údajů definovaných v konfiguraci relace.

Kromě toho můžete vytvořit novou relaci prostředí WPS tak, že spustíte rutinu `New-PSSession`. To se popisuje v tématu New-PSSession. Když místo `New-PSWorkflowSession` spustíte `New-PSSession`, můžete přidat parametr ConfigurationName a zadat tak konfiguraci relace jinou než výchozí konfiguraci relace WPS Workflow. Pokud chcete zadat výchozí konfiguraci relace WPS Workflow, přidejte k příkazu parametr `-ConfigurationName Microsoft.PowerShell.Workflow`. Teď můžete v této relaci spouštět workflows, a to buď z modulu, nebo spuštěním samostatného workflowu XAML nebo workflow založeného na skriptu prostředí WPS.

Parametr PSComputerName určuje počítače, na kterých budou role nainstalovány. V následujícím příkladu se workflow spustí jako úloha.

```
Invoke-Command $workflowsession {Install-Role -PSComputerName Server01, Server02 -
AsJob}
```

Připojení nebo opětovné spojení s běžícím workflow

Pokud se znovu připojujete k workflow, který jste už spustili, můžete zadat ID úlohy nebo název workflow, ke kterému chcete znovu navázat spojení. Můžete taky použít rutinu `Connect-PSSession`, se kterou se můžete připojit k odpojené relaci, která byla připojená k počítači, na kterém běží váš workflow (pokud není relace odpojená, musíte ji odpojit).

- Spusťte novou relaci připojenou k počítači, na kterém běží workflow. Uložte připojení relace do proměnné tak, jak ukazuje následující příklad:
  $workflowsession =New-PSWorkflowSession -ComputerName Server01
- Načtěte seznam úloh, které běží na počítači zadaném v kroku 1:
  Invoke-Command $workflowsession {Get-Job}

- Ve výsledcích vyhledejte workflow, ke kterému se chcete znovu připojit.
- Pokud není workflow pozastavený, přejděte na další krok. Abyste obnovili pozastavený workflow, spusťte následující příkaz, kde $workflowsession představuje proměnnou, do které se uložilo připojení relace:
  Invoke-Command $workflowsession {Resume-Job <Název nebo ID úlohy>}
- Další informace o průběhu a stavu workflowu získáte tak, že spustíte Receive-Job, jak ukazuje následující příklad:
  Invoke-Command $workflowsession {Receive-Job <Název nebo ID úlohy>}

(iii) Spuštění workflow v relaci s jobs

`About_Jobs`

Pomocí společného parametru workflowu AsJob, který se přidává ke všem workflowům v prostředí WPS, můžete workflow spustit jako úlohu prostředí WPS. Další informace o úlohách najdete v tématu about_Jobs. Pomocí rutin Job můžete workflowy spouštět, zastavovat, pozastavovat a obnovovat. Pokud chcete workflow spustit jako úlohu, zadejte příkaz v následujícím formátu:
`Sloveso-PodstatnéJméno -PSComputerName managednode01, managednode02 -AsJob`.

POZNÁMKA 15: Invoke-Command a proměnná, ve které je uložená relace workflowu (v tomto případě je to $WFServer), jsou v následujících případech nutné, jenom pokud jste ještě nevytvořili relaci připojenou k počítači, na kterém workflow běží. Pokud už v této relaci jste, stačí spustit jenom příkazy ve složených závorkách.
(Zjistil jsem, že není potřeba ani vytvářet relaci – funguje to i tak…)

Spuštění workflow jako úlohy

- Použití parametru AsJob ke spuštění workflowu Install-Role jako úlohy. Parametr PSComputerName určuje počítače, na kterých budou role nainstalovány. V následujícím příkladu workflow bude mít PSJobTypeName = PSWORKFLOWJOB:
  Invoke-Command $workflowsession {Install-Role -PSComputerName Server01, Server02 -AsJob}
  Pokud spustíte příkaz takto, workflow bude mít PSJobTypeName = REMOTEJOB:
  Invoke-Command $workflowsession {Install-Role -PSComputerName Server01, Server02} -AsJob
- Získání všech úloh spuštěných v relaci serveru workflowu. Každá instance workflowu běží jako samostatná úloha. Úlohy se zobrazují, jako by běžely na místním počítači (localhost), protože objekt úlohy se v relaci serveru workflowu vytvoří místně, a to i přesto, že úloha běží na dvou různých vzdálených počítačích.
  Invoke-Command $workflowsession {Get-Job}
- Získání úloh workflowů, které ovlivňují konkrétní spravovanou roli, pomocí parametru Filter
  Invoke-Command $workflowsession {Get-Job -Filter {PSComputerName = "Server01"}
- Získání stavu úlohy
  Invoke-Command $workflowsession { (get-job -name Job1).state}
- Vyčkání na dokončení úlohy
  Invoke-Command $workflowsession {wait-job -name Job1}
- Získání výsledků úlohy
  Invoke-Command $workflowsession {receive-job -name Job1}
- Smazání nebo odstranění úlohy
  Invoke-Command $workflowsession {remove-job -name Job1}

Pozastavení a obnovení workflow jako úlohy

- Pozastavení workflowu jako úlohy
  Invoke-Command $workflowsession {suspend-job -name Job1}
- Obnovení pozastaveného workflowu jako úlohy
  Invoke-Command $workflowsession {resume-job -name Job1}

9.6.c) Spuštění workflow zabaleného v modulu

(i) Import a spuštění workflow v modulu

Následující postup ukazuje, jak spustit workflow zabalený do modulu. Tento krok je nutný pouze v případě, že jste v manifestu modulu nezadali klíč kořenového nebo vnořeného modulu.

- Pokud na počítači, na kterém chcete spustit workflow, nemáte existující aktivní relaci, vytvořte novou relaci tak, jak se popisuje v části Vytvoření nové relace workflow.
- Zkopírujte adresář modulu, který jste vytvořili na počítač, na kterém běží workflow. Pro větší pohodlí si ho zkopírujte do adresáře Modules (třeba $env:C:\Users\<user_name>\Documents\WindowsPowerShell\Modules) na daném počítači.
- Volitelně můžete do relace připojené k počítači, na kterém běží workflow, modul importovat jedním z následujících dvou příkazů. V prostředí WPS 3.0 se moduly importují do relace automaticky ve chvíli, kdy uživatel poprvé spustí rutinu, která je součástí daného modulu. Přidejte Invoke-Command, aby se v relaci serveru workflowu spustil příkaz Import-Module. Nepovinný parametr Verbose zobrazuje funkce, které modul přidává do relace.
  Invoke-Command -Session $<název proměnné relace> -ScriptBlock {Import-Module <NázevModulu> -Verbose}
  Například následující příkaz importuje modul RoleManager do relace serveru workflowu.
  Invoke-Command $workflowsession {Import-Module RoleManager -Verbose}
  Rutina Import-Module importuje workflowy do relace jako funkce. Teď můžete spouštět workflowy v relaci, která je připojená k počítači, na kterém chcete workflow spouštět.
- Zadejte název workflowu a stisknutím klávesy Enter workflow spusťte. Pokud chcete třeba spustit workflow, který se uložil do modulu v části Ukládání pracovního postupu v modulu, zadejte Install-Role -PSComputerName <názvy_spravovaných_uzlů> a stiskněte Enter.

(ii) Import a spuštění workflow ze souboru XAML

Následující postup vysvětluje, jak spustit workflow založený na jazyce XAML, který není v modulu v relaci připojené k počítači, na kterém chcete workflow spustit. Až importujete soubory XAML workflow, můžete workflows použít ve svojí relaci serveru workflowu. Další informace o workflows založených na jazyce XAML, které můžete používat s WPS Workflow, najdete v části Vytváření a import workflows pomocí Návrháře postupu provádění sady Visual Studio tématu Začínáme s pracovním postupem prostředí WPS.

- Spusťte relaci serveru workflowu podle postupu v části Vytvoření nové relace workflow.
- Pomocí následující příkazu naimportujte soubor XAML workflowu do relace serveru workflowu.
  Invoke-Command -Session $<název proměnné relace> -Scriptblock {Import-Module <Cesta_k_souboru_Xaml>}

Třeba následující příkaz do relace serveru workflowu naimportuje workflowy Install-Role a Set-Role.
Invoke-Command $workflowsession {Import-Module -Path D:\Temp\Install-Role.xaml, D:\Temp\Set-Role.xaml}

9.7. Zobrazení dat o workflow

Workflowy se můžou do relace přidávat jako funkce. K prozkoumání workflowů můžete použít stejné příkazy, které byste použili k prozkoumání funkcí. Pokud už jste v relaci připojené k počítači, na kterém běží workflow, nemusíte přidávat Invoke-Command $variable. Spusťte jenom příkazy ve složených závorkách.

- Nalezení funkcí, které modul workflowu přidal do relace
  Invoke-Command $workflowsession {get-command -module RoleManager}
- Prozkoumání workflowu nebo funkce
  Invoke-Command $workflowsession {ls function:\Install-Role}
- Zobrazení vlastností funkce workflowu pomocí rutiny Get-Command
  Invoke-Command $workflowsession {get-command Install-Role }
- Získání syntaxe funkce workflowu pomocí rutiny Get-Command
  Invoke-Command $workflowsession {get-command install-role -syntax}
- Prozkoumání objektů, které vrací funkce workflowu
  Invoke-Command $workflowsession {install-role | get-member}

9.8. Hlavní rozdíly mezi WPS a workflow

V této kapitole jsou popsány hlavní rozdíly workflow proti WPS. Ostatní rozdíly, které se týkají přímo parametrů, activities atp. jsou popsány přímo v nich.

- Ve workflow nejde volat metody. Activities ve workflow se převádějí do jazyka XAML a vrací serializované reprezentace objektů (ve formátu XML). Tyto objekty mají vlastnosti a hodnoty vlastností, ale metody dostupné nejsou. Pokud chcete zavolat metodu, umístěte příkaz do activity InlineScript.
- Při volání rutiny nebo activity ve workflow nejde vynechat název parametru – vyžadují se explicitní názvy parametrů, dokonce i pro poziční parametry. Můžete použít alias parametru nebo prvních pár znaků názvu parametru, které postačí k rozlišení tohoto názvu od názvů ostatních parametrů v sadě parametrů.
- Splatting se u activities workflow nebo ve voláních workflow nepovoluje.
- Dynamické parametry, tj. parametry, které k příkazu přidá provider, modul nebo jiná rutina, nejsou ve workflow platné. Pokud chcete používat dynamické parametry, spusťte příkaz InlineScript.
- Každý příkaz ve workflow běží ve svém runspace.WPS Workflow se postará o zpracování; tyto runspaces nemusíte spravovat.
- Příkazy, které běží v různých runspaces obvykle nemůžou sdílet data, jako jsou třeba hodnoty proměnných vytvořené v runspace. WPS Workflow ale přidává všechny proměnné workflow nejvyšší úrovně do každého runspace.
- Pokud použijete ve workflow příkazy, které mění aktuální runspace (např. příkaz, který do aktuálního runspace importuje modul), nebudou mít tyto příkazy vliv na runspaces, ve kterých běží následné příkazy workflow. Pokud chcete importovat modul vyžadovaný příkazem (nebo modul, který se neimportuje automaticky), použijte parametr activity s názvem PSRequiredModules.
- Příkazy v bloku skriptu InlineScript jsou výjimkou z pravidla nezávislého runspace. Ve výchozím nastavení běží tyto příkazy v jednom runspace ve svém vlastním procesu WPS.

9.8.a) Příkazy fungující ve workflow jinak než ve WPS

- Příkazy Begin, Process a End nejsou ve workflow platné.
- Dot-sourcing není ve workflow platný.
- Operátor vyvolání (nebo volání) znak „&“ není ve workflow platný.
- Příkaz Switch musí používat parametr CaseSensitive. Klauzule přepínače musí rozlišovat velká a malá písmena a obsahovat jenom konstanty (nemohou obsahovat porovnávací příkazy, regulární výrazy, odkazy na soubory ani bloky skriptu). Všechny klauzule přepínače musí být stejného typu nebo musí být silného typu.
- Příkazy trap nejsou ve workflow platné. Místo nich používejte platné příkazy Try-catch-finally.
- Příkaz #Requires má parametr Assembly, který do workflow přidává activities definované v sestavení. Parametr Assembly je platný jenom ve workflow.
- Pokud chcete určit sestavení, zadejte popis plně kvalifikovaného modulu nebo absolutní nebo relativní cestu. Pokud je sestavení v globální mezipaměti sestavení (GAC), zadejte jenom název sestavení.
- V podmínkách smyčky nejde měnit hodnoty proměnných. Pokud chcete změnit hodnotu proměnné, umístěte příkaz změny do těla smyčky. U smyčky nelze použít label!
- Příkazy Break a Continue nejsou ve workflow platné. Místo toho použijte k řízení provádění smyčky příkaz If.

9.9. Activities Workflow – příkazy, syntaxe

Activities workflow jsou příkazy v rámci vlastního workflow. A to jak příkazy pro samotné workflow, tak příkazy WPS.

- Všechny jednotky přidané providers, které zahrnuje WPS, jsou ve workflows platné. Hlavními providers jsou Certificate, Environment, FileSystem, Function, Registry, Variable a WS-Management. Pokud chcete používat jednotku přidanou providerem v nainstalovaném modulu, použijte parametr activity PSRequiredModules nebo příkaz Import-Module v activity InlineScript.

Všechny rutiny v následujících modulech zahrnutých ve WPS jsou dostupné jako activities, s výjimkou rutin uvedených v části Vyloučené rutiny. Všechny ostatní rutiny a funkce v modulech Microsoft.PowerShell.* jsou implementované jako activities pro použití ve workflow:

- Microsoft.PowerShell.Core
- Microsoft.PowerShell.Host
- Microsoft.PowerShell.Diagnostics
- Microsoft.PowerShell.Management
- Microsoft.PowerShell.Security
- Microsoft.PowerShell.Utility
- Microsoft.Wsman.Management

Pokud nějaká rutina není v modulu WPS Core, není ale výslovně vyloučená, WPS Workflow automaticky spustí tuto rutinu v activity InlineScript a vrátí výstup do workflow.

9.9.a) Vyloučené rutiny

Následující rutiny WPS nejsou implementované jako activities workflow. Protože jsou tyto rutiny vyloučené, WPS Workflow je nespustí automaticky v activity `InlineScript`. Tyto rutiny se obvykle příliš nehodí pro úkoly workflow. Pokud chcete tyto rutiny spustit, zahrňte je do activity `InlineScript`.

Add-History	Enter-PSSession	Get-PSSnapin	Remove-PSBreakpoint	Set-Variable
Add-PSSnapin	Exit-PSSession	Get-Transaction	Remove-PSSnapin	Start-Transaction
Clear-History	Export-Alias	Get-Variable	Remove-Variable	Start-Transcript
Clear-Variable	Export-Console	Import-Alias	Set-Alias	Stop-Transcript
Complete-Transaction	Get-Alias	Invoke-History	Set-PSBreakpoint	Stop-Transcript
Debug-Process	Get-History	New-Alias	Set-PSDebug	Trace-Command
Disable-PSBreakpoint	Get-PSBreakpoint	New-Variable	Set-StrictMode	Undo-Transaction
Enable-PSBreakpoint	Get-PSCallStack	Out-GridView	Set-TraceMode	Use-Transaction

9.9.b) Rezervovaná slova

Rezervovaná slova ve workflows jsou slova, která nejde použít jako název workflow ani jako název proměnné nebo parametru workflow, protože je používá WPS nebo WPS Workflow.

- Keywords - klíčová slova workflows: Workflow, CheckPoint-Workflow, ForEach-Parallel, Parallel, Sequence, Suspend-Workflow.
- Názvy proměnných modulu runtime workflow.
- Názvy společných parametrů workflows (about_WorkflowCommonParameters).
- Názvy společných parametrů activities.
- Názvy společných parametrů pro WPS (about_CommonParameters).

Pokud mají dva příkazy v relaci stejný název, ale jsou jiného typu příkazu, určí pravidla priorit příkazů, který příkaz se při zavolání názvu příkazu spustí. Pokud mají dva příkazy v relaci stejný název a jsou stejného typu příkazu, spustí se po zavolání názvu příkazu ten, který se do relace přidal jako poslední. Pokud chcete rozlišit mezi příkazy se stejným názvem nebo spustit příkaz, který je skrytý příkazem s vyšší prioritou, použijte název příkazu s kvalifikací modulu (`<NázevModulu\NázevPříkazu>`, například `PSScheduledJob\New-JobTrigger`).

9.9.c) Workflow – keyword (klíčové slovo)

Jména workflow skriptů používejte ve standardu cmdlets – `verb-noun` (sloveso-podstatné jméno). Používejte schválená slovesa (`Get-Verb`) a v podstatných jménech používejte jen písmena, číslice a pomlčky (podtržítka se nedoporučují, protože mohou mást nejen uživatele).

```
workflow <Verb-Noun>
{
[CmdletBinding(ConfirmImpact=<String>,
       DefaultParameterSetName=<String>,
              HelpURI=<URI>,
              PositionalBinding=<Boolean>)]
       Param
              (
                     [Parameter(Mandatory=<$True | $False>)]
                     [<Type>]
                     $<ParameterName>
              )
}
```

9.9.d) Param (parametry workflow) - keyword (klíčové slovo)

```
about_Functions_CmdletBindingAttribute
about_Functions_Advanced_Parameters
about_CommonParameters
about_WorkflowCommonParameters
```

Jde o stejné techniky, které se používají při přidávání parametrů do funkcí. Přidává parametr (atribut Parameter je nepovinný!) do workflow. Názvy parametrů ve workflow mohou obsahovat jenom písmena, číslice, pomlčky (-) a podtržítka (_). Vyhněte se názvům s pomlčkami, protože ty je potřeba uzavřít do složených závorek, aby se daly správně interpretovat. Můžete používat více sad parametrů, ať už pomocí příkazu Param nebo volitelného atributu Parameter (nejsou podporovány ve vnořených workflow!).

WPS Workflow přidává do všech activities ve workflow společné parametry WPS (i do workflow, jež nepoužívají CmdletBinding a atribut Parameter) a také společné parametry workflow.

Hodnoty společných parametrů WPS a společných parametrů workflow jsou ve workflow dostupné jako proměnné.

POZNÁMKA 16: *V názvech parametrů u workflow je rozlišována velikost písmen!*

9.9.e) CheckPoint-Workflow (alias = PSPersist) – keyword (klíčové slovo)

```
about_Checkpoint-Workflow
```

Vytvoří checkpoint. Uloží stav a data probíhajícího workflow do snapshotu na disku managed node. Pokud se workflow přeruší nebo znova spustí, může být spuštěný z libovolného checkpoint. Používejte activity `Checkpoint-Workflow` se společným parametrem workflow PSPersist a proměnnou PSPersistPreference, aby byl váš workflow robustní a obnovitelný.

9.9.f) ForEach-Parallel – keyword (klíčové slovo)

```
about_Foreach-Parallel
```

Souběžné spouštění příkazů v neurčitém pořadí - provede příkazy ve skriptovém bloku jednou pro každou položku v kolekci. Tuto activity používejte na kolekci podobných položek. Většina příkazů ForEach jsou vhodnými kandidáty na využití `ForEach -Parallel`.

9.9.g) Parallel – keyword (klíčové slovo)

```
about_Parallel
```

Souběžné spouštění příkazů v neurčitém pořadí. Pořadí provádění není definováno. Tuto activity používejte na příkazy, které nesdílí data, například `Get-Process` a `Get-Service`. Nelze vnořovat.

9.9.h) Sequence – keyword (klíčové slovo)

```
about_Sequence
```

Vytvoří blok sekvenčních příkazů v rámci skriptového bloku `Parallel`. Skriptový blok `Sequence` běží paralelně s ostatními activities ve skriptovém bloku `Parallel`. Příkazy ve skriptovém bloku `Sequence` ale běží v pořadí, ve kterém jsou uvedené. Tato activity je platná jenom ve skriptovém bloku `Parallel`. Nelze vnořovat.

9.9.i) Suspend-Workflow – keyword (klíčové slovo)

```
about_Suspend-Workflow
```

Dočasně zastaví workflow. Pokud budete chtít, aby workflow pokračoval, použijte rutinu `Resume-Job`.

9.9.j) Invoke-Expression

Pomocí rutiny `Invoke-Expression` můžete do workflow zahrnout inline kód XAML. Ve workflow má rutina `Invoke-Expression` parametr Language, který přebírá hodnotu „XAML“. Cestu k souboru XAML zadejte pomocí parametru Command.

9.9.k) New-Object

Ve workflow jsou platné jenom parametry Typename a Argument.

9.9.l) Restart-Computer

Ve workflow je parametr Wait platný při restartování místních i vzdálených počítačů.

9.9.m) InlineScript

9.9.n) Atribut CmdletBinding

```
about_Functions_CmdletBindingAttribute
```

Je to volitelný atribut. Podporuje následující parametry:

- ConfirmImpact
- DeafultParameterSetName
- HelpUri
- PositionalBinding
- SupportsShouldProcess

9.9.o) Proměnné ve skriptech workflow

```
about_Preference_Variables
about_Automatic_Variables
about_WorkflowCommonParameters
PSWorkflowRuntimeVariable Enumeration
```

- Proměnné vytvořené na nejvyšší úrovni workflow jsou dostupné v rámci celého workflow. Proměnné vytvořené na úrovni skriptu nebo příkazu jsou dostupné jen pro tento skript či příkaz.
- Klíčová slova, které jsou vyhrazená v jazyce Visual Basic .NET, nelze použít jako názvy proměnných ve workflow.
- Výsledky podvýrazů nejde uložit do proměnné, protože podvýrazy se nedají zachovat.
- Proměnné prostředí se nedá přiřadit hodnota.

- Ke změně hodnot vlastností se nedají použít odkazy na proměnné.

(i) Pravidla pro proměnné v cyklech, vložených skriptech a příkazech Parallel

Pravidla pro vnořené příkazy ve workflow	Zobrazení hodnot proměnných workflow	Změna hodnot proměnných workflow	Interní proměnné jsou viditelné v oboru workflow
Cykly/řídicí příkazy	Ano	Ano	Ano
Parallel/Sequence	Ano	Ne. Použijte $Workflow.	Ne, vraťte proměnnou.
InlineScript	Ne. Použijte $Using.	Ne. Vraťte proměnnou. $Workflow je neplatný.	Ne, vraťte proměnnou.

(ii) Automatické proměnné WPS

- $Args
- $Error
- $MyInvocation
- $PID
- $PSBoundParameters
- $PsCmdlet
- $PSCommandPath
- $PSScriptRoot
- $StackTrace

(iii) Preferenční proměnné workflow

Preferenční proměnné workflow jsou platné jen ve workflow a ovlivňují jen dané workflow.

$PSDisableSerializationPreference <Boolean>

- $True: Vztahuje se na celý workflow (ne jen na danou activity)! Nedoporučuje se nastavovat na $True, protože workflow, jež neserializuje objekty, vrací objekty s jejich metodami a vlastnostmi, takže pokud se dosáhne checkpointu nelze tyto objekty uložit.
- $False: Objekt workflow je serializován. Default.

$PSParentActivityId <string>

Vytváří identifikátor pro activities v běžném rozsahu a všech jeho podřízených rozsazích. Tento identifikátor se objeví v záznamech pro activities v ovlivněných rozsazích.

$PSPersistPreference <logická hodnota>

Přidá ke skupině activities checkpoint. Hodnota $True udělá checkpoint bod po všech activities, které následují za definicí proměnné. Tato hodnota je ekvivalentní nastavení hodnoty parametru activity PSPersist na $true pro všechny příslušné activities. Pokud chcete toto chování vypnout, nastavte hodnotu této proměnné na $False.

- $True: Kromě checkpoints určených ve workflow přidá za každou activity checkpoint. Ovlivňuje jenom activities, které následují za definicí této proměnné.
- $False: Ukončí přidávání checkpoints. Checkpoints se vytvoří, když jsou ve workflow zadané.

$PSRunInProcessPreference

Proměnná workflow, která určuje, jestli se activities spouštějí v procesu. Pokud je tato proměnná zadaná, spouštějí se všechny activities ve vymezeném oboru v procesu workflow.

(iv) Proměnné runtime workkflow

WPS workflow přidává ke každému workflow proměnné runtime workflow. Do těchto proměnných patří společné parametry workflow a jejich hodnoty, takže ve svém workflow nemusíte tyto proměnné znovu definovat. Ke změně hodnot proměnných runtime workflow uvnitř něj použijte activity `Set-PSWorkflowData`.

9.9.p) Activity parametry

Na parametry se odkazujete stejně Jako u WPS – pomocí znaku "$" před názvem parametru.

- Poziční parametry v activities nejsou platné – názvy parametrů jsou povinné! Jsou ovšem povoleny alias a zkratky parametrů.
- Splatting se nepovoluje.
- Dynamické parametry nejsou platné, pokud ho potřebujete použít musíte použít activity InlineScript.
- Při volání rutiny nebo activity ve workflow název parametru nejde vynechat – vyžadují se explicitní názvy parametrů, dokonce i pro poziční parametry. Můžete použít alias parametru nebo prvních pár znaků názvu parametru, které postačí k rozlišení tohoto názvu od názvů ostatních parametrů v sadě parametrů.

(i) Konkrétní activity parametry

PSRequiredModules - pokud chcete importovat modul vyžadovaný příkazem (nebo modul, který se neimportuje automaticky). Moduly importované v jednom runspace nemají vliv na ostatní runspace.

(ii) Společné parametry activities

```
about_WorkflowCommonParameters
about_CommonParameters
```

Některé společné parametry workflow jsou zároveň společnými parametry activities. Společné parametry activities nejsou platné například v activities `Suspend-Workflow` a `Checkpoint-Workflow`. Nejsou dostupné ani v rutinách nebo výrazech, které WPS Workflow automaticky spouští v nějaké activity. Jsou dostupné v activity `InlineScript`, ale ne v příkazech v bloku skriptu `InlineScript`.

Ke změně hodnot společných parametrů a parametrů runtime ve workflow použijte activity `Set-PSWorkflowData`. Společné parametry pro workflow se ve workflow znázorní jen tehdy, pokud jsou uvedeny v příkazovém řádku.

Workflow Common Parameters	Windows PowerShell Common Parameters
PSAllowRedirection	Debug
PSApplicationName	ErrorAction
PSAuthentication	Verbose
PSCertificateThumbprint	WarningAction
PSComputerName	
PSConfigurationName	
PSConnectionRetryCount	
PSConnectionRetryIntervalSec	
PSConnectionURI	
PSCredential	
PSPersist	
PSPort	
PSSessionOption	
PSUseSSL	

(iii) Společné parametry specifické pro activities

Name	Definition
AppendOutput <SwitchParameter>	Přidá výstup activity k hodnotě proměnné. Standardně se při přiřazení hodnoty proměnné nahradí hodnota proměnné. Například následující příkaz přidá objekt procesu k objektu služby v proměnné $x: workflow Test-Workflow { $x = Get-Service; $x = Get-Process -AppendOutput }
DisplayName <String>	Určuje popisný název activity. Hodnota parametru DisplayName se objeví v indikátoru průběhu, když workflow probíhá, a v hodnotě vlastnosti Progress úlohy workflow. Pokud příkaz obsahuje taky parametr PSProgressMessage, zobrazí se obsah indikátoru průběhu ve formátu <DisplayName>:<PSProgressMessage>.
Input <Object[]>	Předá activity kolekci objektů. Je to alternativa k předávání jednotlivých objektů activity pomocí zřetězení příkazů.
MergeErrorToOutput <SwitchParameter>	Přidá chyby do výstupního datového proudu. Když tento parametr použijete s klíčovými slovy Parallel a ForEach -Parallel, spojíte chyby a výstup více paralelních příkazů do jedné kolekce.
PSActionRetryCount <Int32>	Pokud první pokus o spuštění activity selže, pokusí se activity opakovaně spustit. Výchozí hodnota je 0 – neopakuje.
PSActionRetryIntervalSec <Int32>	Určuje interval v sekundách mezi opakování pokusů o akci. Výchozí hodnota 0 – okamžitě opakuje pokus o akci. Tento parametr je platný jenom v případě, že v příkazu je taky použit parametr PSActionRetryCount.
PSActionRunningTimeoutSec <Int32>	Určuje, jak dlouho má activity běžet na každém cílovém počítači. Pokud activity neskončí dřív, než vyprší časový limit, WPS Workflow vygeneruje ukončující chybu a zastaví zpracování workflow na příslušném cílovém počítači.
PSDebug <PSDataCollection[DebugRecord]>	Přidá zprávy ladění z activity do zadané kolekce záznamů ladění, místo aby se tyto zprávy vypisovaly na konzole nebo zapisovaly do hodnoty vlastnosti Debug úlohy workflow. Můžete taky přidat zprávy ladění z více activities do stejného objektu kolekce záznamů ladění. Když chcete tento společný parametr activities použít, použijte rutinu New-Object k vytvoření objektu PSDataCollection, který bude mít typ DebugRecord, a uložte tento objekt v proměnné. Potom tuto proměnnou použijte jako hodnotu parametru PSDebug jedné nebo více activities, jak ukazuje následující příklad.
PSDisableSerialization <Boolean>	Nařizuje activity, aby workflow vrátila „živé“(neserializované) objekty. Výsledné objekty mají metody i vlastnosti, ale při vytvoření kontrolního bodu se nedají uložit.
PSDisableSerializationPreference <Boolean>	Je to obdoba parametru PSDisableSerialization, ale platí pro všechny activities ve workflow, nejen pro jednu activity. Přidání tohoto parametru se obecně nedoporučuje, protože workflow který svoje objekty neserializuje, se nedá obnovit nebo zachovat. Platné hodnoty jsou True nebo False.
PSError <PSDataCollection[ErrorRecord]>	Přidá chybové zprávy z activity do zadané kolekce záznamů chyb, místo aby se tyto zprávy vypisovaly na konzole nebo zapisovaly do hodnoty

	vlastnosti Error úlohy workflow. Můžete taky přidat chybové zprávy z více activities do stejného objektu kolekce záznamů chyb. Když chcete tento společný parametr activities použít, použijte rutinu New-Object k vytvoření objektu PSDataCollection, který bude mít typ ErrorRecord, a uložte tento objekt v proměnné. Potom tuto proměnnou použijte jako hodnotu parametru PSError jedné nebo více activities.
PSProgress <PSDataCollection[ProgressRecord]>	Přidá zprávy o průběhu z activity do zadané kolekce záznamů průběhu, místo aby se tyto zprávy vypisovaly na konzole nebo zapisovaly do hodnoty vlastnosti Progress úlohy workflow. Můžete taky přidat zprávy o průběhu z více activities do stejného objektu kolekce záznamů průběhu.
PSProgressMessage <String>	Určuje stručný popis activity. Hodnota parametru PSProgressMessage se zobrazí v indikátoru průběhu, když workflow probíhá. Pokud příkaz obsahuje taky parametr DisplayName, zobrazí se obsah indikátoru průběhu ve formátu <DisplayName>:<PSProgressMessage>. Tento parametr je zvlášť užitečný k identifikaci activities ve skriptovém bloku ForEach -Parallel. Bez této zprávy jsou activities ve všech paralelních větvích označené stejným názvem.
PSRemotingBehavior <RemotingBehavior>	Určuje, jak se spravuje vzdálená komunikace, když se activity spouští na cílových počítačích. Výchozí hodnota je PowerShell. Platné hodnoty jsou: None: activity se nespouští na vzdálených počítačích. PowerShell: Ke spuštění activity na cílových počítačích se použije vzdálená komunikace WPS. Custom: activity podporuje svůj vlastní typ vzdálené komunikace. Tato hodnota je platná, když rutina, která se implementuje jako activity, nastaví hodnotu atributu RemotingCapability na SupportedByCommand a příkaz obsahuje parametr ComputerName.
PSRequiredModules <String[]>	Před spuštěním příkazu importuje zadané moduly. Zadejte názvy modulů. Moduly musí být nainstalované na počítači, na kterém se provádí workflow. Moduly, které jsou nainstalované v cestě určené proměnnou prostředí PSModulePath, se automaticky importují při prvním použití libovolného příkazu v modulu. Tento parametr slouží k importu modulů, které nejsou v umístění určeném proměnnou PSModulePath. Protože každá activity ve workflow běží ve vlastní relaci, příkaz Import-Module importuje modul jenom do relace, ve které activity běží. Neimportuje modul do relací, ve kterých běží jiné activities.
PSVerbose <PSDataCollection[VerboseRecord]>	Přidá podrobné zprávy z activity do zadané kolekce podrobných záznamů, místo aby se tyto zprávy vypisovaly na konzole nebo zapisovaly do hodnoty vlastnosti Verbose úlohy workflow. Můžete taky přidat podrobné zprávy z více activities do stejného objektu kolekce podrobných záznamů.
PSWarning <PSDataCollection[WarningRecord]>	Přidá zprávy upozornění z activity do zadané kolekce záznamů upozornění, místo aby se tyto zprávy vypisovaly na konzole nebo zapisovaly do hodnoty vlastnosti Warning úlohy workflow. Můžete taky přidat zprávy upozornění z více activities do stejného objektu kolekce záznamů upozornění.
UseDefaultInput <Boolean>	Akceptuje veškerý vstup workflow jako vstup activity pomocí hodnoty. Například activity Get-Service v následujícím příkladu používá společný parametr activity UseDefaultInput k získání vstupu, který se předává workflow. Když workflow spustíte se vstupem, bude tento vstup použitý activity.

9.9.q) Přidání activities do workflow

Všechny příkazy a výrazy, které ve workflow použijete se spouští jako activities. Většinu rutin WPS převádí workflow na activities. Ale malá část rutin je z převodu vyloučená - **https://technet.microsoft.com/cs-cz/library/jj574194(v=ws.11).aspx**. Tyto vyloučené rutiny lze spustit v activity `InlineScript`.

Pokud rutina nemá odpovídající activity a není explicitně vyloučená, WPS Workflow automaticky spustí rutinu v activity inlineScript a vrátí její výstup do workflow.

9.9.r) Jak spustit v rámci skriptu workflow další skript

Pokud potřebujete spustit ve workflow skript (soubor s příponou .ps1), uzavřete volání skriptu do activity `InlineScript`. Protože workflow může běžet na několika nodes, místo relativní cesty **používejte absolutní cestu** k souboru skriptu – vzhledem k tomu, že workflow může být spuštěno na více nodes.

9.10. Přidání checkpoints do workflow

Workflow je určený pro praxi, kdy jsou součástí běhu přerušení sítě, restartování počítače, zpoždění a výpadky napájení. Všechny časově náročné úkoly, které ovlivňuje víc heterogenních počítačů v distribuované architektuře, musí předpokládat záměrné i neúmyslné přerušení, reagovat na ně, obnovit se, pokračovat a skončit. Checkpoints jsou důležitou součástí této strategie.

Do workflow můžete přidat několik checkpoints s využitím různých technik.WPS automaticky používá data v nejnovějším checkpoint k obnovení workflow, pokud dojde k jeho úmyslnému nebo náhodnému přerušení.

Data workflow (neboli trvalá data) pro workflow se uloží do profilu uživatele na pevném disku počítače, který je hostitelem relace workflow.

Při spuštění workflow jako úlohy, například pomocí společného parametru workflow AsJob, zůstanou checkpoints zachovány, dokud neodstraníte úlohu, třeba pomocí rutiny `Remove-Job`. Jinak se checkpoints po dokončení workflow odstraní.

Když jsou checkpoints nastavené u activities ve zřetězení příkazů, checkpoint se nepořídí, dokud příkazy neskončí. Když jsou nastavené u activities v bloku skriptu `Parallel`, checkpoint se nepořídí, dokud se nespustí blok skriptů `Parallel` na všech cílových počítačích. Když jsou ale checkpoint nastavené u activities v bloku skriptu `Sequence`, checkpoint se pořídí po dokončení každé activity v každém cílovém počítači.

9.10.a) Kam umístit checkpoints?

Checkpoints můžete umístit kdekoli ve workflow, včetně před a za jednotlivé příkazy nebo výrazy nebo za každou activity. Protože vytvoření checkpointu vyžaduje režii (shromáždění dat a jejich uložení na disk), je potřeba používat checkpointing s rozumem.

Při plánování checkpoints berte v úvahu proces obnovení workflow. Doba, po kterou bude trvat opětovné spuštění oddílu workflow po přerušení, je delší než doba potřebná k zápisu stavu checkpoint na disk. Mějte na paměti, že checkpoint se pořídí pokaždé, když se workflow spustí na každém cílovém počítači (jakoby byl celý workflow uzavřený ve výrazu ForEach).

Checkpoint přidejte po každém dokončení důležité součásti workflow, kterou nebudete chtít v případě přerušení opakovat. To může být součást workflow, která trvá moc dlouho; oddíl vyžadující prostředky, které nejsou vždycky k dispozici; koordinace více počítačů, připojení a prostředků.

Po dokončení definice workflow pomocí procesu testování zpřesněte checkpoint. Přidejte nebo odeberte checkpoint (nezapomeňte na komentáře!) k dosažení přiměřené provozní doby a přiměřené doby obnovení.

9.10.b) Jak přidat checkpoints?

(i) Společný parametr workflow PSPersist

PSPersist přidá checkpoint do workflow. Používá se při spuštění příkazu workflow. PSPersist nemůže ve workflow zrušit, ignorovat nebo potlačit explicitní checkpoints. To platí pro všechny workflow.

Ve výchozím nastavení přidá workflow checkpoint na začátek a konec workflow, kromě všech checkpoints, které jsou určené ve workflow.

- $True: Přidá checkpoint na začátek a konec workflow za každou activity určenou ve workflow.
- $False: Žádné checkpoint na začátku nebo na konci workflow. Pořídí checkpoint, jenom pokud je zadaný ve workflow.

(ii) Společný parametr activity PSPersist

PSPersist s hodnotou $True pořídí checkpoint po dokončení activity. Je platný pro všechny activities ve workflow. Není platný ve výrazech nebo příkazech v bloku skriptu InlineScript.

- $True: Pořídí checkpoint po dokončení activity.
- $False: Nepřidá žádné checkpoints. Tato hodnota nemá žádný vliv.

(iii) Activity CheckPoint-Workflow

Activity `Checkpoint-Workflow` vytváří checkpoint okamžitě. Activity `Checkpoint-Workflow` můžete ve workflow použít víckrát a umístit ji za libovolný příkaz nebo výraz ve workflow, s tím rozdílem, že se nemůže objevit v bloku skriptu `InlineScript`. Activity `Checkpoint-Workflow` nemá žádné parametry, ani běžné, ani společné parametry workflow.

(iv) Preferenční proměnná $PSPersistPreference

Hodnota předvolby $PSPersistPreference společně s hodnotou $True pořídí checkpoint po každé activity, která následuje za definicí proměnné. Je ekvivalentní k nastavení hodnoty parametru activity PSPersist na $true pro všechny příslušné activities. Pokud chcete zastavit přidávání checkpoints, nastavte hodnotu proměnné předvolby $PSPersistPreference na $False.

Na rozdíl od jiných proměnných předvoleb je tato proměnná účinná jenom ve workflow, ne v relaci WPS. Můžete ji definovat v relaci nebo v profilu WPS, ale nemá žádný vliv na workflow spuštěné pod touto relací WPS.

- $True: K checkpoints určeným ve workflow přidá checkpoint za každou activity. Tato hodnota ovlivňuje všechny následné activities ve workflow.
- $False: Ukončí přidávání dalších checkpoints. Pořídí checkpoints, jenom pokud je zadaný ve workflow.