Ed Wilson

Pro verzi 3.0 a vyšší

PowerShell

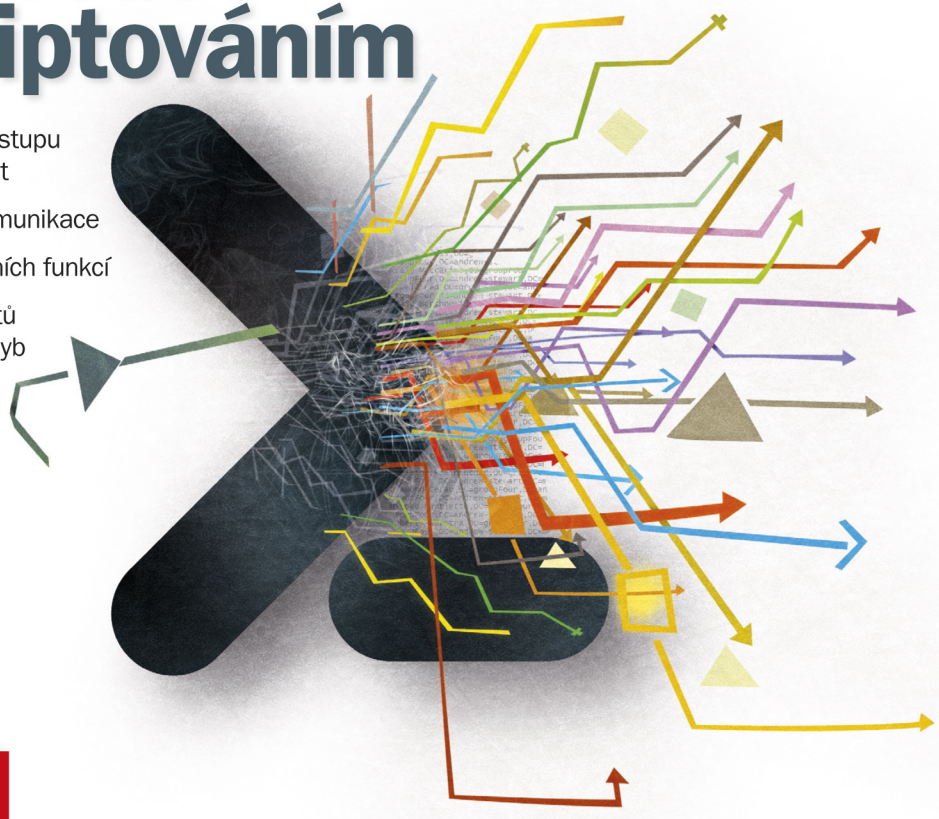
Průvodce skriptováním

Zpracování vstupu a výstupu dat

Vzdálená komunikace

Tvorba vlastních funkcí

Ladění skriptů a obsluha chyb



computer press

Ed Wilson

PowerShell

Computer Press
Brno
2015

PowerShell

Ed Wilson

Překlad: Jakub Goner
Obálka: Martin Sodomka
Odpovědný redaktor: Martin Herodek
Technický redaktor: Jiří Matoušek

Objednávky knih:
http://knihy.cpress.cz
www.albatrosmedia.cz
eshop@albatrosmedia.cz
bezplatná linka 800 555 513
ISBN 978-80-251-4386-5

Vydalo nakladatelství Computer Press v Brně roku 2015 ve společnosti Albatros Media a. s.
se sídlem Na Pankráci 30, Praha 4. Číslo publikace 19 077.

1. vydání

ALBATROS MEDIA a.s.

Obsah

Věnuji Terese, mé spřízněné duši.

Ed Wilson

Předmluva

Jsou různé důvody, proč se pustit do automatizace. Pro mě to byla malá želva z programu s názvem LOGO. V té době jsem samozřejmě vůbec netušil, že se učím programovat. Byl jsem pouhé dítě na základní škole, které se baví tím, že kreslí malé obrázky. O mnoho let později jsem se stal správcem IT a čím dál více jsem nesnášel nudná zadání, jako je třeba ruční kopírování souboru na 100 vzdálených serverů. Začal jsem s automatizací proto, že jsem nedokázal snést představu, že se budu stále dokola zabývat monotónními úkoly. Nějaký čas mi trvalo, než jsem si vše spojil do souvislostí a uvědomil jsem si, že malá želva mi otevřela cestu ke kariéře zaměřené na nasazení a výuku automatizace.

Prostředí Windows PowerShell ideálně vyplňuje prostor pro automatizační nástroje v systémech Windows, protože nabízí výkonné a rozsáhlé možnosti a zároveň zůstává dostatečně jednoduchý, aby s ním mohl začít rychle pracovat i uživatel bez hlubokých technických znalostí. Windows PowerShell se sice může uplatnit jako jednoduché automatizační prostředí, ale vyznačuje se určitými specifiky, kvůli nimž není jeho zvládnutí úplně prosté. Dá se to přirovnat k autu s ručním řazením. Pro člověka, který dosud řídil jen vozidla s automatickou převodovkou, může být ze začátku jízda poněkud náročná. Jakmile se však auto rozjede na první stupeň, zbytek už lze zvládnout docela snadno. Edu Wilsonovi se v této knize podařilo vytvořit dokonalý úvod do prostředí Windows PowerShell, který čtenářům poskytuje stručné a popisné rady, aby mohli rychle zařadit svůj první rychlostní stupeň.

Jako technologický vedoucí projektu Windows PowerShell v divizi Microsoft Services trávím většinu pracovní doby před zákazníky společnosti Microsoft a snažím se je naučit práci s prostředím Windows PowerShell a předat jim své nadšení pro tento produkt. V každém svém kurzu zdůrazňuji vysokou návratnost investic do zvládnutí tohoto prostředí. Nepřestává mě udivovat, že stačí pochopit základní koncepce prostředí Windows PowerShell a poté je můžeme stále znovu aplikovat, což zároveň zvyšuje efektivitu podniku a přináší nám osobní uspokojení.

Při výuce ve svých kurzech vždy učím, že termíny „Windows PowerShell" a „skriptování" se rozhodně mohou vylučovat. Technicky vzato jednořádkové příkazy prostředí Windows PowerShell stále patří mezi „skripty," ale pro mě představují ideální nástroj, který umožňuje vyřešit konkrétní problém bez nutnosti ovládnout vývojářsky orientované dovednosti. Jednořádkové příkazy jsou obvykle orientovány na konkrétní úkol a jsou logicky jednoduché, ale přesto dokážou automatizovat neuvěřitelné množství úkolů. Každý, kdo s prostředím Windows PowerShell právě začíná, se sám přesvědčí, že může tento nástroj účinně využívat i bez psaní skriptů. V celé své knize se autor zaměřil na principy prostředí Windows PowerShell a ukazuje jeho jednoduchost. Skriptováním se přímo zabývá teprve v závěrečných částech knihy. Pokročilí uživatelé nakonec do svého rejstříku řadí i skriptování a tvorbu nástrojů, ale dokážou zvládnout hodně věcí i před tím, než se do této fáze dostanou.

Nezávisle na velmi rozmanité úrovni svých znalostí mají moji studenti něco společného. Prostředí Windows PowerShell bylo vytvořeno takovým způsobem, aby práce s ním byla zábavná a efektivní pro každého: od počítačového začátečníka až po zkušeného vývojáře. Úplní začátečníci například nemusí vědět, že je toto prostředí plně objektově orientované a je založené na architektuře .NET Framework. Mohou se věnovat pouhému spouštění příkazů v prostředí Windows PowerShell a nemusí se nijak zabývat vlastnostmi objektového modelu, ale přitom dokážou svou práci účinně automatizovat. Jakmile se začnou dozvídat o objektech, otevřou se před nimi nové možnosti. Stále mě překvapuje, že prostředí Windows PowerShell může odpovídat potřebám natolik odlišných uživatelů.

Když přemýšlím o hodnotě tohoto prostředí a o tom, proč stojí za to se s ním seznamovat, vybavuje se mi základní protiklad mezi „vytvářením" a „provozováním". Když poněkud zjednodušíme role v IT, vidíme, že existuje dělicí linie mezi vývojáři a správci. Vývojáři tvoří řešení a správci se starají o návrh, nasazení a fungování systémů, které se při tom používají. Prostředí Windows PowerShell může toto rozdělení překlenout a obě role propojit. Správci mohou díky němu vyvíjet automatizační řešení a obejdou se přitom bez skutečného vývojáře. Jazyk prostředí Windows PowerShell zahrnuje mnoho prvků, které skrývají a zjednodušují jeho interní složitost. Profesionálové v IT tak mohou být ve své práci efektivnější a více cenění. Zvládnutí prostředí Windows PowerShell může zvýšit vaši hodnotu na pracovišti a zároveň vám usnadnit život.

Edu Wilsonovi, zvanému „The Scripting Guy", někteří lidé také říkají „PowerShellebrity". Ve světě prostředí Windows PowerShell představuje superhvězdu, má rozsáhlé zkušenosti se skriptováním a patří mezi nejvíce energické a nadšené lidi, které jsem potkal. Jsem Edovi vděčný za to, že píše své knihy, protože spousta lidí díky nim získává přístup k jeho bohatým zkušenostem a znalostem. Tato kniha je stručným a srozumitelným úvodním průvodcem k prostředí Windows PowerShell. Kdybych s tímto prostředím začínal, asi bych ji nedokázal odložit. Ať už jste se s prostředím Windows PowerShell začali seznamovat, nebo se k tomu teprve chystáte, tato kniha vám nepochybně pomůže s dalšími kroky.

Gary Siepsert
Senior Premier Field Engineer (PFE)
Microsoft Corporation

Úvod

Gary už v předmluvě zmínil téměř vše, co jsem chtěl napsat v úvodu. Tuto knihu jsem vytvořil pro úplné začátečníky, a měli byste ji proto číst od začátku do konce. Potřebujete-li knihu, která více připomíná referenční příručku, podívejte se na některou z mých knih ze série PowerShell Best Practices, nebo dokonce na titul *PowerShell 3.0 Step by Step*. Kniha „Step by Step" nepatří mezi standardní referenční příručky, ale jedná se spíše o praktický průvodce, jímž můžete v ideálním případě navázat na knihu, kterou právě čtete. Máte-li zájem o denní dávku informací o prostředí PowerShell, navštivte můj blog Hey Scripting Guy na adrese *http://www.ScriptingGuys.com/blog*. Nový obsah zveřejňuji dvakrát denně.

Systémové požadavky

Požadavky na hardware

Počítač by měl splňovat následující minimální požadavky na hardware:

- 2,0 GB paměti RAM (doporučuje se více),
- 80 GB dostupného místa na pevném disku,
- připojení k Internetu.

Požadavky na software

Chcete-li dokončit cvičení v této knize, měli byste mít nainstalováno prostředí Windows PowerShell 3.0:

Prostředí Windows PowerShell 3.0 lze získat z webu Stažení softwaru společnosti Microsoft. Stáhněte si balíček Windows Management Framework a nainstalujte jej do systému Windows 7 Service Pack 1, Windows Server 2008 R2 SP1 nebo Windows Server 2008 Service Pack 2.

- V systémech Windows 8 a Windows Server 2012 je prostředí Windows PowerShell 3.0 již obsaženo. Zkušební verze těchto operačních systémů jsou k dispozici na webu TechNet:
  http://technet.microsoft.com/en-US/evalcenter/hh699156.aspx?ocid=wc-tn-wctc
  http://technet.microsoft.com/en-US/evalcenter/hh670538.aspx?wt.mc_id=TEC_108_1_4
- Část o službě Active Directory vyžaduje přístup ke službě AD DS (Active Directory Domain Services). U těchto případů potřebujete přístup k systému Windows Server 2012.
- V kapitole o serveru Exchange budete potřebovat přístup k serveru se systémem Microsoft Exchange Server 2013. Zkušební verze těchto operačních systémů jsou k dispozici na webu TechNet:
  http://technet.microsoft.com/en-us/evalcenter/hh973395.aspx

Poděkování

K úspěchu této knihy přispělo mnoho lidí. Prvním z nich je Teresa Wilson neboli „Skriptovací manželka". Vždy je mým prvním čtenářem a bez jejího souhlasu se nic nedostane mimo náš dům. Na druhém místě musím zmínit svého odborného korektora Briana Wilhiteho, který odvedl skvělou práci při hledání chyb, omylů a zavádějících informací. Chci také poděkovat uživatelské skupině prostředí PowerShell v Charlotte, jejíž otázky, připomínky a názory ke vzniku této knihy značně přispěly. Při psaní jsem na vás vzpomínal. Mé díky si také zaslouží Michael Bolinger a Melanie Yarbrough z nakladatelství O'Reilly, kteří se mimořádně zasloužili o to, aby se tento projekt řádně dostal do cíle.

Zpětná vazba od čtenářů

Nakladatelství a vydavatelství Computer Press, které pro vás tuto knihu přeložilo, stojí o zpětnou vazbu a bude na vaše podněty a dotazy reagovat. Můžete se obrátit na následující adresy:

Computer Press
Albatros Media a.s., pobočka Brno
IBC
Příkop 4
602 00 Brno

nebo

sefredaktor.pc@albatrosmedia.cz

Computer Press neposkytuje rady ani jakýkoli servis pro aplikace třetích stran. Pokud budete mít dotaz k programu, obraťte se prosím na jeho tvůrce.

Errata

Přestože jsme udělali maximum pro to, abychom zajistili přesnost a správnost obsahu, chybám se úplně vyhnout nelze. Pokud v některé z našich knih najdete chybu, ať už chybu v textu nebo v kódu, budeme rádi, pokud nám ji oznámíte. Ostatní uživatele tak můžete ušetřit frustrace a nám můžete pomoci zlepšit následující vydání této knihy.

Veškerá existující errata zobrazíte na adrese *http://knihy.cpress.cz/K2195* po klepnutí na odkaz Soubory ke stažení.

KAPITOLA 1

Přehled prostředí Windows PowerShell 3.0

V této kapitole:
- Seznámení s prostředím Windows PowerShell
- Práce s prostředím Windows PowerShell
- Používání rutin prostředí Windows PowerShell
- Zadávání parametrů rutin
- Práce s možnostmi nápovědy
- Shrnutí

Když poprvé spustíte prostředí Windows PowerShell, ať už se jedná o konzolu prostředí Windows PowerShell, nebo o Integrované skriptovací prostředí (ISE) v prostředí Windows PowerShell, bude na zadání příkazu čekat prázdná obrazovka. Bohužel není zřejmé, jak má takový příkaz vypadat. Prostředí neposkytuje žádné průvodce ani jiné podobné funkce systému Windows.

Prostředí Windows PowerShell dostalo svůj název ze dvou důvodů: Jedná se o prostředí (shell), které je výkonné (powerful). Bylo by chybou se domnívat, že Windows PowerShell je jen skriptovací jazyk, protože se zdaleka neomezuje jen na tento aspekt. Stejně tak bychom prostředí Windows PowerShell podcenili, kdybychom si mysleli, že dokáže pouze spouštět pár rutin. Díky skriptování poskytuje přístup ke kompletní škále technologií pro správu, jaké jsou ve světě Windows k dispozici.

V této kapitole představíme prostředí Windows PowerShell a ukážeme si mimořádné možnosti, které tento pružný a užitečný nástroj pro správu poskytuje.

Seznámení s prostředím Windows PowerShell

Prostředí Windows PowerShell má dvě podoby. První je interaktivní konzola (podobná konzole KORN či BASH ve světě UNIX), která je integrována do příkazového řádku systému Windows. Konzola prostředí Windows PowerShell usnadňuje zadávání krátkých příkazů a poskytuje výstup setříděných, filtrovaných a formátovaných výsledků. Tyto výsledky lze snadno zobrazovat v konzole, ale také přesměrovat do souborů XML, CSV nebo textových souborů. Konzola prostředí Windows PowerShell poskytuje několik výhod, jako je rychlost, nízká paměťová režie a komplexní služba přepisu, která zaznamenává všechny příkazy a jejich výstupy.

Druhou variantu prostředí Windows PowerShell představuje Integrované skriptovací prostředí (ISE). Integrované skriptovací prostředí (ISE) v prostředí Windows PowerShell nemusí sloužit jen k psaní skriptů. Mnoho uživatelů prostředí Windows PowerShell v praxi raději píše svůj kód v Integrovaném skriptovacím prostředí (ISE), aby mohlo využívat funkce barevného zvýrazňování syntaxe, rozevíracích seznamů a automatického nabízení parametrů.

Integrované skriptovací prostředí (ISE) v prostředí Windows PowerShell navíc poskytuje rozšíření „Show Command", které umožňuje vytvářet příkazy prostředí Windows PowerShell v grafickém uživatelském rozhraní pomocí myši. Vytvořený příkaz je buď přímo spuštěn, nebo je přidán do podokna Script. Záleží to na uživateli. Další informace o práci s Integrovaným skriptovacím prostředím (ISE) naleznete v kapitole 10, „Používání Integrovaného skriptovacího prostředí (ISE) v prostředí Windows PowerShell".

Poznámka: Při práci s jednoduchými příkazy budeme pro jednoduchost zobrazovat příkazy i jejich výsledky v rámci konzoly prostředí Windows PowerShell. Přitom si však budeme uvědomovat, že všechny příkazy lze spustit rovněž z Integrovaného skriptovacího prostředí (ISE) v prostředí Windows PowerShell. Nezávisle na tom, zda je příkaz zadán v konzole prostředí Windows PowerShell, v Integrovaném skriptovacím prostředí (ISE) v prostředí Windows PowerShell, jako naplánovaná úloha nebo jako filtr v zásadách skupin, vždy se jedná o prostředí Windows PowerShell. Skript prostředí Windows PowerShell ve své nejzákladnější formě sestává ze sady příkazů tohoto prostředí.

Práce s prostředím Windows PowerShell

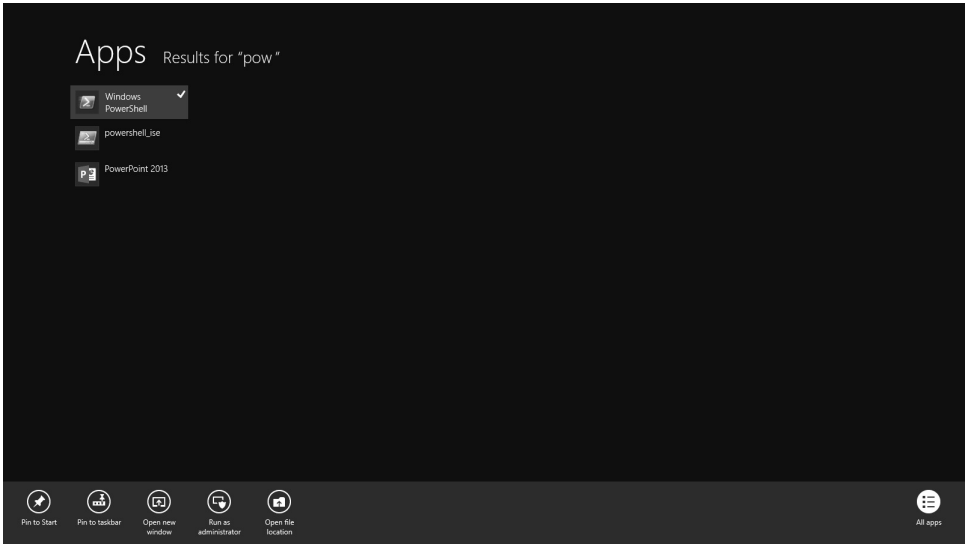
Prostředí Windows PowerShell 3.0 je součástí systémů Windows 8 a Windows Server 2012. V systému Windows 8 stačí v okně Start zadat jen prvních několik písmen slova *PowerShell* a položka Windows PowerShell se objeví mezi dostupnými možnostmi. Tento postup je znázorněn na obrázku 1.1. Zadal jsem do pole Search pouze řetězec **pow** a jednou z nabídnutých možností je také Windows PowerShell.

Kdybychom kvůli spuštění prostředí Windows PowerShell pokaždé museli přejít do okna Start a zadávat písmena **pow**, bylo by to poněkud nepraktické. Osobně proto dávám přednost připnutí zástupců konzoly prostředí Windows PowerShell a Integrovaného skriptovacího prostředí (ISE) v prostředí Windows PowerShell jak do okna Start, tak na hlavní panel systému Windows. Po tomto připnutí zástupců na aplikace (postup je znázorněn na obrázku 1.2) lze odkudkoli získat přístup k prostředí Windows PowerShell jedním klepnutím.

V systému Windows Server 2012 není nutné hledat ikonu pomocí pole Search v okně Start, protože ikona konzoly prostředí Windows PowerShell je již standardně zobrazena na hlavním panelu.

Poznámka: Integrované skriptovací prostředí (ISE) v prostředí Windows PowerShell (editor skriptů) není ve výchozím nastavení v systému Windows Server 2012 k dispozici. Integrované skriptovací prostředí (ISE) v prostředí Windows PowerShell je nutné přidat jako funkci. Postup přidání Integrovaného skriptovacího prostředí si ukážeme v kapitole 10,„Používání Integrovaného skriptovacího prostředí (ISE) v prostředí Windows PowerShell".

Obrázek 1.1: Po zadání textu do okna Start se otevře okno Search se zvýrazněnou položkou konzoly prostředí Windows PowerShell



Obrázek 1.2: Po klepnutí pravým tlačítkem myši na ikonu prostředí Windows PowerShell v poli výsledků hledání se zobrazí možnosti Pin to Start a Pin to taskbar

Bezpečnostní aspekty prostředí Windows PowerShell

Prostředí Windows PowerShell lze spustit dvěma způsoby: jako správce nebo jako normální uživatel bez zvýšených oprávnění. Je vhodné spouštět prostředí Windows PowerShell s minimálními právy. V systémech Windows 7 a Windows 8 k tomu stačí pouze klepnout na ikonu prostředí Windows PowerShell. Prostředí se spustí pro uživatele bez zvýšených oprávnění, i když jste přihlášeni s právy správce. V systému Windows Server 2012 se prostředí Windows PowerShell automaticky spouští s právy aktuálního uživatele. Jste-li tedy přihlášeni jako správce domény, konzola prostředí Windows PowerShell se spustí s právy správce domény.

Spuštění pro uživatele bez zvýšených oprávnění

Vzhledem k tomu, že prostředí Windows PowerShell dodržuje bezpečnostní omezení systému Windows, nemůže uživatel prostředí Windows PowerShell provést žádnou akci, k níž nemá jeho uživatelský účet příslušná oprávnění. Jste-li tedy uživatelem bez zvýšených oprávnění, nemáte právo provádět akce typu instalace ovladačů tiskáren, čtení protokolu zabezpečení ani změny systémového času.

Jestliže spravujete místní počítač se systémem Windows 7 nebo Windows 8 a nespustíte prostředí Windows PowerShell s právy správce, dojde k chybám, když se pokusíte provést některé akce, jako je prohlížení konfigurace diskových jednotek. Následující příklad uvádí příkaz a příslušnou chybu:

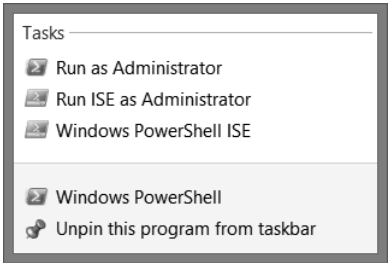
```
PS C:\> get-disk
get-disk : Access to a CIM resource was not available to the client.
At line:1 char:1
+ get-disk
+ ~~~~~~~~
    + CategoryInfo    : PermissionDenied:
                        (MSFT_Disk:ROOT/Microsoft/Windows/Storage/MSFT_Disk)
                        [Get-Disk], CimException
    + FullyQualifiedErrorId : MI RESULT 2,Get-Disk
```

Tip: Pokud se pokusíte spustit rutiny, které vyžadují zvýšená oprávnění, dojde k nekonzistentnímu chování a objeví se chyby. V konzole prostředí Windows PowerShell bez zvýšených oprávnění například rutina *Get-Disk* poskytne chybu „Access To A CIM Resource Was Not Available To The Client". Rutina *Stop-Service* generuje chybu „Cannot Open XXX Service On Computer". Rutina *Get-VM* jednoduše nevrátí žádné informace a nezobrazí žádnou chybu. Při řešení potíží tedy nejdříve zkontrolujte oprávnění konzoly.

Spuštění prostředí Windows PowerShell s právy správce

Chcete-li provést činnosti, které vyžadují práva správce, musíte konzolu prostředí Windows PowerShell spustit s příslušnými právy. Přitom klepněte pravým tlačítkem myši na ikonu prostředí Windows PowerShell (ikonu připnutou k hlavnímu panelu, ikonu v okně Start nebo ikonu nalezenou pomocí pole Search v okně Start) a z nabídky Action zvolte možnost Run as Administrator. Velkou výhodou tohoto postupu je to, že můžete spustit buď konzolu prostředí

Windows PowerShell (první položku nabídky), nebo ze stejné obrazovky také Integrované skriptovací prostředí (ISE) v prostředí Windows PowerShell, v obou případech jako správce. Tyto možnosti jsou patrné na obrázku 1.3.



Obrázek 1.3: Po klepnutí pravým tlačítkem myši na ikonu prostředí Windows PowerShell se objeví možnost Run as Administrator

Po spuštění konzoly prostředí Windows PowerShell s právy správce se zobrazí dialogové okno nástroje User Account Control (UAC), kde je nutné povolit, že může prostředí Windows PowerShell změnit systém. Prostředí Windows PowerShell ve skutečnosti žádné změny systému neprovede, alespoň zatím. Máte-li příslušná práva, rozhodně však pomocí prostředí Windows PowerShell můžete do systému zasahovat. To je smysl zobrazeného dialogového okna.

Poznámka: Tuto výzvu lze potlačit vypnutím nástroje UAC (Řízení uživatelských účtů). Nástroj UAC však plní důležitou bezpečnostní funkci, takže jej vypínat nedoporučuji. V systémech Windows 7 a Windows 8 byl tento nástroj vyladěn. Oproti stavu při svém uvedení v systému Windows Vista již nástroj zobrazuje mnohem méně výzev.

Když nyní pracujete s prostředím Windows PowerShell s právy správce, můžete provést cokoli, k čemu vám účet poskytuje oprávnění. Jestliže například spustíte rutiny *Get-Disk*, zobrazí se informace podobné následující ukázce:

```
PS C:\> get-disk

Number Friendly Name              Operational    Total Size Partition
                                  status                    Style
------ -------------              ------------   ---------- -----------
0      INTEL SSDSA2BW160G3L       Online          149.05 GB MBR
```

Používání rutin prostředí Windows PowerShell

Všechny rutiny prostředí Windows PowerShell fungují podobným způsobem. Díky tomu se snáze používají. Rutiny prostředí Windows PowerShell mají vesměs názvy složené ze dvou částí. První část tvoří sloveso, ačkoli pokaždé není v gramaticky správném tvaru. Sloveso naznačuje akci, kterou příkaz provede. Jako příklady sloves lze uvést *Get*, *Set*, *Add*, *Remove* a *Format*. Podstatné jméno určuje položku, na kterou se akce aplikuje. K podstatným jménům patří *Process*, *Service* (služba), *Disk* a *NetAdapter* (síťový adaptér). Celý název příkazu prostředí Windows

PowerShell se skládá ze slovesa a podstatného jména, mezi něž je vložena pomlčka. Příkazy prostředí Windows PowerShell se označují jako rutiny (v angličtině *cmdlet* neboli „command let"), protože se chovají jako krátké příkazy či programy. Dají se používat samostatně nebo je lze kombinovat mechanismem, který se nazývá *zřetězení* (pipeline). Další informace o zřetězení naleznete v kapitole 2, „Používání rutin prostředí Windows PowerShell".

Nejčastější sloveso: Get

Více než 25 procent z téměř 2 000 rutin (a funkcí) v systému Windows 8 obsahuje sloveso *Get*. Sloveso *Get* znamená načítání informací. Získané informace závisí na podstatném jméně, které je druhou částí názvu rutiny. Chcete-li získat informace o procesech ve svém systému, spusťte konzolu prostředí Windows PowerShell buď klepnutím na ikonu prostředí Windows PowerShell na hlavním panelu, nebo zadáním příkazu **PowerShell** v okně Start systému Windows 8, aby se zobrazily výsledky hledání položky Windows PowerShell, jak jsme popsali v předchozí části „Spuštění prostředí Windows PowerShell s právy správce".

Po zobrazení konzoly prostředí Windows PowerShell spusťte rutinu *Get-Process*. Přitom můžete název rutiny dokončit pomocí funkce Tab Completion prostředí Windows PowerShell. Jakmile se zobrazí název rutiny, spusťte příkaz stisknutím klávesy Enter.

Poznámka: Funkce Tab Completion prostředí Windows PowerShell značně šetří čas. Kromě toho, že není nutné vypisovat celé názvy, zabraňuje také chybám, protože přesně řeší názvy rutin. Můžeme ji přirovnat ke korektoru pravopisu názvů rutin. Bylo by například značně frustrující, kdybychom se pokusili ručně napsat přesný název rutiny jako *Get-NetAdapterEncapsulatedPacketTaskOffload*. Když však použijeme funkci Tab Completion, stačí napsat jen *Get-Net* a stisknout asi šestkrát klávesu Tab, aby se správně zadaný název rutiny objevil v konzole Windows PowerShell. Seznámení s tím, jak rychle a účinně pracovat s funkcí Tab Completion, patří mezi klíčové prvky úspěšné práce s prostředím Windows PowerShell.

Nalezení informací o procesech

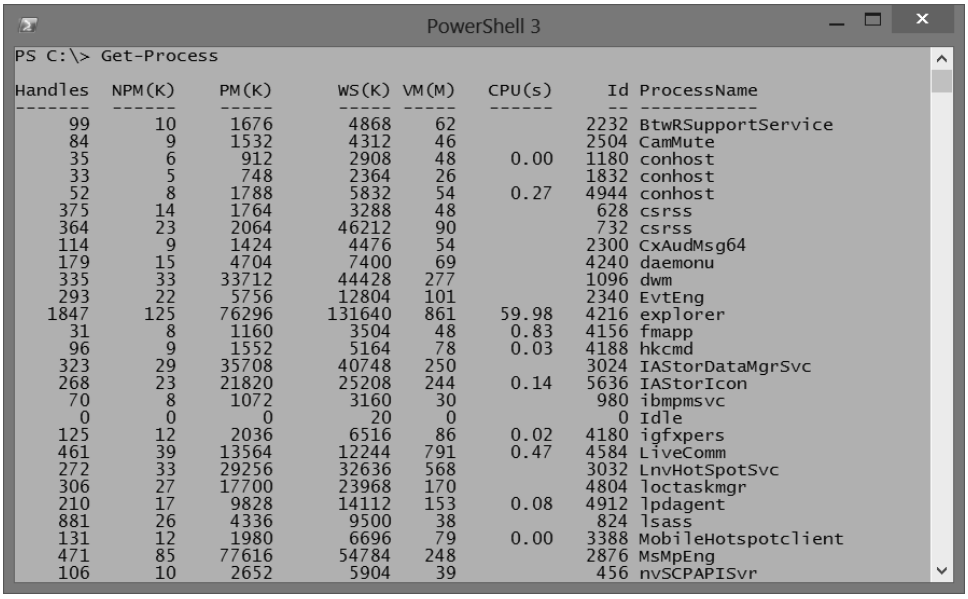
Chcete-li pomocí funkce Tab Completion prostředí Windows PowerShell zadat název rutiny *Get-Process* na příkazový řádek konzoly prostředí Windows PowerShell, zadejte na první řádek konzoly prostředí Windows PowerShell následující řetězec a poté stiskněte klávesu Tab následovanou klávesou Enter:

```
Get-Pro
```

Tato sekvence příkazů – příkaz následovaný klávesami Tab a Enter – se označuje jako *expanze tabulátoru*. Příkaz *Get-Process* a příslušný výstup je znázorněn na obrázku 1.4.

Chcete-li najít informace o službách systému Windows, použijte sloveso *Get* a podstatné jméno *Service*. V konzole prostředí Windows PowerShell zadejte následující řetězec a poté stiskněte klávesu Tab následovanou klávesou Enter:

```
Get-Servi
```



Obrázek 1.4: Rutina Get-Process prostředí Windows PowerShell vrací podrobné informace o procesech v systému Windows

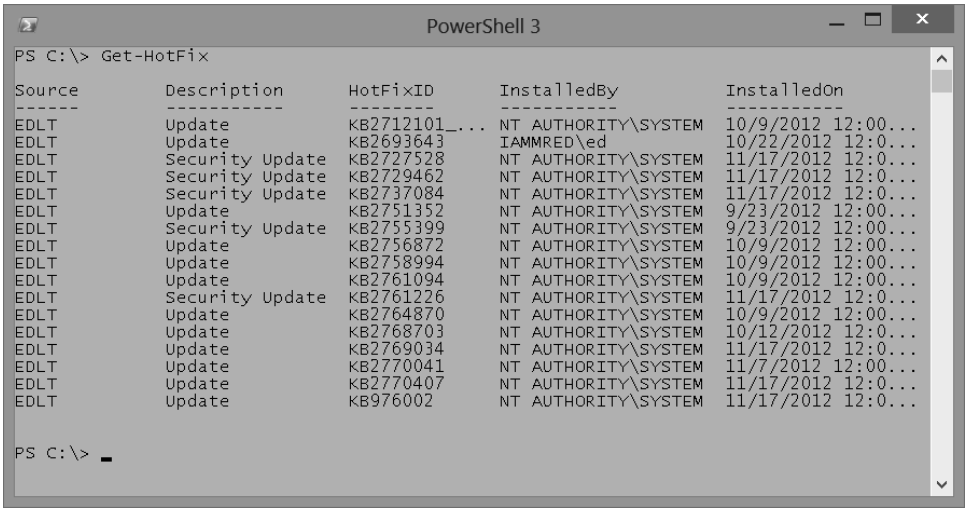
Poznámka: Konvence prostředí Windows PowerShell vyžaduje použití podstatných jmen v jednotném čísle. Ačkoli se neuplatňuje univerzálně (ve svém počítači mám asi 50 podstatných jmen v množném čísle), jedná se o dobré výchozí pravidlo. Pokud si tedy nejste jisti, zda je podstatné jméno (či parametr) v jednotném, nebo množném čísle, zvolte jednotné číslo. Ve většině případů bude správná tato varianta.

Identifikace nainstalovaných oprav hotfix systému Windows

Chcete-li vypsat seznam oprav hotfix, které byly nainstalovány do aktuálního systému Windows, použijte rutinu *Get-Hotfix*. Sloveso *Get* je v tomto případě následováno podstatným jménem *Hotfix*. V konzole prostředí Windows PowerShell zadejte následující řetězec a poté stiskněte klávesu Tab následovanou klávesou Enter:

```
Get-Hotf
```

Příkaz *Get-Hotfix* a příslušný výstup je znázorněn na obrázku 1.5.



Obrázek 1.5: Rutina Get-Hotfix umožňuje zobrazit podrobný seznam všech aplikovaných oprav hotfix systému Windows

Získání podrobných informací o službách

Informace o službách v systému lze zjistit pomocí rutiny *Get-Service*. Opět není potřeba zadávat celý příkaz. Po zadání následujícího řetězce lze pomocí expanze tabulátoru dokončit příkaz *Get-Service* a spustit jej:

```
Get-Servi
```

Poznámka: Efektivita expanze tabulátoru závisí na počtu rutin, funkcí nebo modulů nainstalovaných v počítači. Spolu s rostoucím počtem dostupných příkazů se úměrně snižuje i účinnost expanze tabulátoru.

Po spuštění rutiny *Get-Service* se zobrazí následující (zkrácený) výstup:

```
PS C:\> Get-Service

Status   Name                DisplayName
------   ----                -----------
Running  AdobeActiveFile...  Adobe Active File Monitor V6
Stopped  AeLookupSvc         Application Experience
Stopped  ALG                 Application Layer Gateway Service
Stopped  AllUserInstallA...  Windows All-User Install Agent
<VÝSTUP JE ZKRÁCEN>
```

Identifikace nainstalovaných síťových adaptérů

Potřebujete-li informace o síťových adaptérech v počítači se systémem Windows 8 nebo Windows Server 2012, použijte rutinu *Get-NetAdapter*. Zadejte následující řetězec a poté stiskněte klávesu Tab následovanou klávesou Enter:

```
Get-NetA
```

Následující příklad uvádí příkaz a příslušný výstup:

```
PS C:\> Get-NetAdapter

Name                      InterfaceDescription                ifIndex Status
----                      --------------------                ------- ------
Network Bridge            Microsoft Network Adapter Multi...       29 Up
Ethernet                  Intel(R) 82579LM Gigabit Network...      13 Not Pre...
vEthernet (WirelessSwi... Hyper-V Virtual Ethernet Adapter #4      31 Up
vEthernet (External Sw... Hyper-V Virtual Ethernet Adapter #3      23 Not Pre...
vEthernet (InternalSwi... Hyper-V Virtual Ethernet Adapter #2      19 Up
Bluetooth Network Conn... Bluetooth Device (Personal Area...       15 Disconn...
Wi-Fi                     Intel(R) Centrino(R) Ultimate...         12 Up
```

Načtení detekovaných profilů síťových připojení

Profily síťového připojení, které systém Windows 8 či Windows Server 2012 detekoval pro každé rozhraní, lze zobrazit pomocí rutiny *Get-NetConnectionProfile*. Tento příkaz je možné pomocí expanze tabulátoru spustit následujícím řetězcem:

```
Get-NetC
```

Následující příklad uvádí příkaz a příslušný výstup:

```
PS C:\> Get-NetConnectionProfile


Name               :   Unidentified network
InterfaceAlias     :   vEthernet (InternalSwitch)
InterfaceIndex     :   19
NetworkCategory    :   Public
IPv4Connectivity   :   NoTraffic
IPv6Connectivity   :   NoTraffic

Name               :   Network 10
InterfaceAlias     :   vEthernet (WirelessSwitch)
InterfaceIndex     :   31
NetworkCategory    :   Public
IPv4Connectivity   :   Internet
IPv6Connectivity   :   NoTraffic
```

Poznámka: Prostředí Windows PowerShell nerozlišuje velká a malá písmena. V několika případech na velkých a malých písmenech záleží (například při práci s regulárními výrazy), ale názvy rutin, parametry a hodnoty lze zadávat velkými i malými písmeny. Konvence prostředí Windows PowerShell používá kombinaci velkých a malých písmen, přičemž velká písmena obvykle vyznačují začátek slov ve složenině typu *NetConnectionProfile*. Není to však nutné k tomu, aby prostředí Windows PowerShell příkaz správně interpretovalo. Uvedená kombinace velkých a malých písmen pouze zlepšuje čitelnost. Používáte-li expanzi tabulátoru, prostředí Windows PowerShell příkaz automaticky převede do tohoto formátu.

Získání aktuálního nastavení jazykové verze

Typický počítač se systémem Windows má dvě kategorie nastavení jazykové verze. První kategorie obsahuje nastavení jazykové verze, která se týkají rozložení klávesnice a formátu zobrazení položek typu čísel, měny a dat. Hodnotu těchto nastavení jazykové verze lze zjistit pomocí rutiny *Get-Culture*. Chcete-li spustit rutinu *Get-Culture* pomocí expanze tabulátoru, zadejte na příkazový řádek konzoly prostředí Windows PowerShell následující řetězec a poté stiskněte klávesu Tab následovanou klávesou Enter:

```
Get-Cu
```

Příkaz po svém spuštění vrátí do konzoly prostředí Windows PowerShell kromě zobrazovaného názvu nastavení jazykové verze také základní informace, jako je identifikační kód kódu jazyka (LCID – Language Code ID number) a zkrácený název nastavení jazykové verze.

Následující příklad uvádí příkaz a příslušný výstup:

```
PS C:\> Get-Culture

LCID              Name              DisplayName
----              ----              -----------
1033              en-US             English (United States)
```

Do druhé kategorie patří aktuální nastavení uživatelského rozhraní (UI – user interface) systému Windows. Na nastavení jazykové verze uživatelského rozhraní závisí, jaké textové řetězce se zobrazí v prvcích uživatelského rozhraní, jako jsou nabídky a chybové zprávy. Aktuální nastavení jazykové verze uživatelského rozhraní lze určit pomocí rutiny *Get-UICulture*. Chcete-li zavolat rutinu *Get-UICulture*, zadejte následující řetězec a poté stiskněte klávesu Tab následovanou klávesou Enter:

```
Get-Ui
```

Následující příklad uvádí příkaz a příslušný výstup:

```
PS C:\> Get-UICulture

LCID              Name              DisplayName
----              ----              -----------
1033              en-US             English (United States)
```

Poznámka: V mém počítači je jazyková verze národního prostředí i uživatelského rozhraní aktuálně nastavena na stejnou hodnotu. To však neplatí vždy a občas se vyskytují potíže, když je nastaveno lokalizované uživatelské rozhraní a samotný počítač má nastaveno národní prostředí na americkou angličtinu. Problémy vznikají hlavně tehdy, když se používají virtuální počítače vytvořené v jiných zemích. V tomto případě může být velmi frustrující dokonce i jednoduchý úkol, jako je zadávání hesla. Tyto situace je možné vyřešit pomocí rutiny *Set-Culture*.

Zjištění aktuálního data a času

Aktuální datum a čas místního počítače lze zjistit pomocí rutiny *Get-Date*. U této rutiny expanze tabulátoru příliš nepomáhá, protože existuje 15 rutin (v mém počítači), jejichž název začíná písmeny *Get-Da*. Patří k nim rutiny přímého přístupu (Direct Access) i vzdáleného přístupu (Remote Access). Chcete-li tedy zjistit datum a použít přitom expanzi tabulátoru, je potřeba zadat dále uvedený řetězec a poté stisknout klávesu Tab následovanou klávesou Enter:

```
Get-Dat
```

Předchozí syntaxe vyžaduje stejný počet stisknutí kláves, jako byste zadali celý následující název a poté stiskli klávesu Enter:

```
Get-Date
```

Následující příklad uvádí příkaz a příslušný výstup:

```
PS C:\> Get-Date

Tuesday, November 20, 2012 9:54:21 AM
```

Generování náhodného čísla

V prostředí Windows PowerShell 2.0 se objevila rutina *Get-Random*. Nejdříve mě příliš nezaujala, protože jsem náhodná čísla již uměl generovat. Jak je patrné v následující ukázce, pomocí třídy *System.Random* platformy .NET Framework lze vytvořit novou instanci objektu *System.Random* a zavolat metodu next:

```
PS C:\> (New-Object system.random).next()
225513766
```

Samozřejmě jsem náhodných čísel nevytvářel mnoho. Komu by se chtělo psát tak dlouhý příkaz? Jakmile jsem však mohl pracovat s rutinou *Get-Random*, začal jsem náhodná čísla používat k nejrůznějším účelům. Rutina *Get-Random* poskytuje například následující možnosti:

- výběr výherců cen v skriptovacích soutěžích,
- výběr výherců cen na setkáních skupin uživatelů prostředí Windows PowerShell,
- náhodné připojování ke vzdáleným serverům kvůli vyrovnávání zatížení,
- vytváření složek s náhodnými názvy,
- vytváření dočasných uživatelů služby Active Directory s náhodnými jmény,
- nastavení náhodně dlouhé prodlevy před spuštěním nebo zastavením procesů a služeb (ideální při výkonnostním testování).

Ukázalo se, že rutina *Get-Random* patří k velmi užitečným rutinám. Chcete-li v konzole prostředí PowerShell generovat náhodné číslo, zadejte následující řetězec na první řádek konzoly a poté stiskněte klávesu Tab následovanou klávesou Enter:

```
Get-R
```

Následující příklad uvádí příkaz a příslušný výstup:

```
PS C:\> Get-Random
248797593
```

Zadávání parametrů rutin

Rutiny prostředí Windows PowerShell, které se používají nejsnáze, nevyžadují žádné parametry. Tato skupina bohužel čítá jen zlomek z celkového počtu rutin (a funkcí) dostupných v prostředí Windows PowerShell 3.0, jak je k dispozici v systému Windows 8 nebo Windows Server 2012. Stejný postup expanze tabulátoru, který v konzole prostředí Windows PowerShell umožňuje zadávat názvy rutin, naštěstí funguje také pro parametry.

Používání jednotlivých parametrů

Při práci s rutinami prostředí Windows PowerShell často stačí zadat pouze jediný parametr, který zajistí filtrování výsledků. Pokud se jedná o výchozí parametr, není nutné uvádět jeho název a lze jej použít pozičně. Předpokládá se tedy, že první hodnota uvedená za názvem rutiny je hodnotou výchozího parametru (parametru na první pozici). Na druhou stranu, jestliže se jedná o pojmenovaný parametr, je při jeho použití vždy nutné uvádět název (případně alias názvu či částečný název parametru).

Nalezení určitých typů oprav hotfix

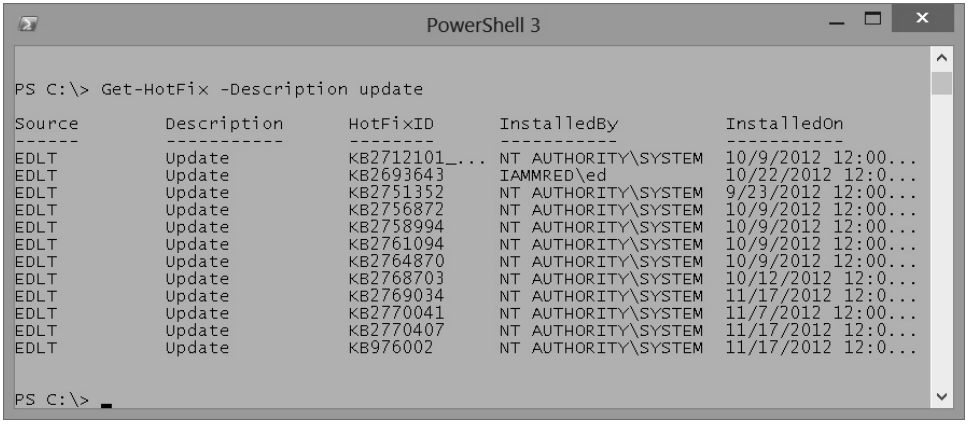
Chcete-li najít všechny opravy hotfix služby Windows Update, použijte rutinu *Get-HotFix* s parametrem -Description a v parametru -Description uveďte hodnotu update. Ve skutečnosti je to snazší, než to vypadá. Jakmile zadáte řetězec **Get-Hot** a stisknete klávesu Tab, objeví se část příkazu *Get-Hotfix*. Potom napište mezeru, parametr *-D* a stiskněte klávesu Tab. Tím dokončíte část příkazu ve tvaru *Get-HotFix -Description*. Nyní stačí zadat **Update** a stisknout klávesu Enter. S trochou praxe vám expanze tabulátoru vejde do krve.

Příkaz *Get-Hotfix* a příslušný výstup je znázorněn na obrázku 1.6.

Pokud se pokusíte najít pouze opravy hotfix typu aktualizací tak, že zadáte hodnotu update na první pozici, dojde k chybě. Následující příklad uvádí problematický příkaz a příslušnou chybu:

```
PS C:\> Get-HotFix update
Get-HotFix : Cannot find the requested hotfix on the 'localhost' computer.
             Verify the input and run the command again.
At line:1 char:1
+ Get-HotFix update
```

```
+ ~~~~~~~~~~~~~~~~~
    + CategoryInfo          :
      ObjectNotFound: (:) [Get-HotFix], ArgumentException
    + FullyQualifiedErrorId :
      GetHotFixNoEntriesFound,Microsoft.PowerShell.Commands.GetHotFixCommand
```



Obrázek 1.6: Přidáte-li k rutině Get-HotFix parametr -Description, zobrazí se ve filtrovaném seznamu konkrétní opravy hotfix typu aktualizací

Tato chyba sice není zcela jasná, ale podle všeho naznačuje, že se rutina *Get-HotFix* pokusila najít opravu hotfix s názvem update. K tomu v tomto případě skutečně došlo. Podle informací o rutině *Get-HotFix* v souboru nápovědy je na první pozici parametr *-ID*, jak je patrné v následujícím příkladu:

```
    -Id <String[]>
        Gets only hotfixes with the specified hotfix IDs.
        The default is all hotfixes on the computer.

          Required?                    false
          Position?                    1
          Default value                All hotfixes
          Accept pipeline input?       false
          Accept wildcard characters?  False
```

Možná se zeptáte: „Jak je to s parametrem -Description?“ Ze souboru nápovědy vyplývá, že -Description je pojmenovaný parametr. Tento parametr tedy můžete použít jen tehdy, jestliže uvedete jeho název, jak jsme ukázali na začátku této části. Následuje odpovídající část souboru nápovědy týkající se parametru -Description:

```
    -Description <String[]>
        Gets only hotfixes with the specified descriptions. Wildcards are
        permitted. The default is all hotfixes on the computer.

          Required?                    false
          Position?                    named
          Default value                All hotfixes
```

```
        Accept pipeline input?         false
        Accept wildcard characters?    True
```

Nalezení konkrétních procesů

Informace o jednotlivém procesu lze zjistit pomocí parametru -Name. Vzhledem k tomu, že parametr -Name je výchozí parametr (na první pozici) rutiny *Get-Process*, není nutné jej při volání rutiny *Get-Process* uvádět. Chcete-li například najít informace o procesu prostředí Windows PowerShell pomocí rutiny *Get-Process*, spusťte pomocí expanze tabulátoru následující příkaz na příkazovém řádku konzoly prostředí Windows PowerShell:

```
Get-Pro + <TAB> + <MEZERA> + Po + <TAB> + <ENTER>
```

Následující příklad uvádí příkaz a příslušný výstup:

```
PS C:\> Get-Process powershell

Handles    NPM(K)    PM(K)       WS(K) VM(M)    CPU(s)     Id ProcessName
-------    ------    -----       ----- -----    ------     -- -----------
    607        39   144552      164652   718      5.58   4860 powershell
```

Jestli rutina *Get-Process* přijímá parametr -Name pozičním způsobem, lze zjistit z toho, že podle souboru nápovědy se tento parametr nachází na prvním místě. Tato pozice je patrná v následující ukázce:

```
    -Name <String[]>
        Specifies one or more processes by process name. You can type
        multiple process names (separated by commas) and use wildcard
        characters. The parameter name ("Name") is optional.

        Required?                      false
        Position?                      1
        Default value
        Accept pipeline input?         true (ByPropertyName)
        Accept wildcard characters?    True
```

Poznámka: Při použití pozičních parametrů dbejte opatrnosti, protože mohou být matoucí. Například prvním parametrem rutiny *Get-Process* je parametr *-Name*, ale u rutiny *Stop-Parameter* je na prvním místě parametr *-ID*. Optimální je vždy zkontrolovat v souboru nápovědy, které parametry jsou skutečně volány a na které pozici jsou očekávány. Ještě důležitější je to při použití rutin s více parametry, k nimž patří rutina *Get-Random* diskutovaná v následující části kapitoly.

Generování náhodných čísel z intervalu

Při použití bez parametrů vrátí rutina *Get-Random* číslo, které se bude nacházet v rozsahu od 0 do 2 147 483 647. Nikdy se nám nestalo, že by se setkání skupiny uživatelů prostředí Windows PowerShell zúčastnilo 0 lidí, a ani jednou se nesešlo 2 147 483 647 lidí. Pokud tedy pomocí rutiny *Get-Random* vybíráte výherce losování o ceny, je důležité, abyste zvolili odlišné minimální a maximální číslo.

Poznámka: Při zadávání parametru -Maximum rutiny *Get-Random* pamatujte na to, že maximální hodnota se nikdy neobjeví. Jestliže se tedy setkání skupiny uživatelů prostředí Windows PowerShell účastní 15 lidí, měli byste nastavit parametr -Maximum na hodnotu 16 (to neplatí, pokud jste si na osobu s číslem 15 zasedli a chcete, aby nikdy nevyhrála).

Výchozím parametrem rutiny *Get-Random* je parametr -Maximum. To znamená, že rutina *Get-Random* umožňuje generovat náhodné číslo v intervalu od 0 do 20 pomocí jednoduché expanze tabulátoru na prvním řádku konzoly prostředí Windows PowerShell. Pamatujte, že rutina *Get-Random* nikdy nedosáhne maximálního čísla, takže vždy volte hodnotu o jednotku vyšší, než je požadovaný horní limit. Postupujte takto:

```
Get-R + <TAB> + <MEZERA> + 21
```

Chcete-li generovat náhodné číslo mezi 1 a 20, možná si myslíte, že byste mohli zadat *Get-Random 1 21*. Tento zápis však generuje chybu. Následující příklad uvádí příkaz a chybu:

```
PS C:\> Get-Random 1 21
Get-Random :
  A positional parameter cannot be found that accepts argument '21'.
At line:1 char:1
+ Get-Random 1 21
+ ~~~~~~~~~~~~~~
    + CategoryInfo          :
      InvalidArgument: (:) [Get-Random], ParameterBindingException
    + FullyQualifiedErrorId :
      PositionalParameterNotFound,Microsoft.PowerShell.
      Commands.GetRandomCommand
```

Tato chyba sděluje, že nelze najít poziční parametr, který by přijímal argument 21. Je to způsobeno tím, že rutina *Get-Random* má pouze jeden poziční parametr: -Maximum. Parametr -Minimum je pojmenovaný parametr. Tento parametr je uveden v souboru nápovědy rutiny *Get-Random*. Používáním souborů nápovědy se budeme zabývat v kapitole 2, „Používání rutin prostředí Windows PowerShell".

Chcete-li generovat náhodné číslo v intervalu od 1 do 20, použijte pojmenované parametry. Pokud potřebujete pomoc při sestavení příkazu, můžete název rutiny i názvy parametrů zadávat pomocí expanze tabulátoru. Chcete-li příkaz vytvořit pomocí expanze tabulátoru, zadejte na příkazový řádek tuto sekvenci:

```
Get-R + <TAB> + -M + <TAB> + <MEZERA> +
  21 + -M + <TAB> + <MEZERA> + 1 + <ENTER>
```

Následující příklad uvádí příkaz a příslušný výstup:

```
PS C:\> Get-Random -Maximum 21 -Minimum 1
19
```

Úvod do sad parametrů

Rutiny prostředí Windows PowerShell mají jednu potenciálně matoucí vlastnost: Jednu rutinu lze často používat různými způsoby. Můžete například zadat parametry -Minimum a -Maximum, ale nemůžete zároveň uvést parametr -Count. To je poněkud nešťastné, protože by se mohlo zdát, že nastavení minimálního a maximálního počtu náhodných čísel pomocí parametrů -Minimum a -Maximum dává smysl. Když skupina uživatelů prostředí Windows PowerShell může svým členům nabídnout pět cen, není efektivní psát skript, který bude generovat pět náhodných čísel, ani spouštět stejný příkaz pětkrát.

Zde se uplatní sady příkazů. Parametry -Minimum a -Maximum určují rozsah, ve kterém se bude vybírat jediné náhodné číslo. Chcete-li generovat více než jedno náhodné číslo, uveďte parametr -Count. Následující příklad znázorňuje dvě sady parametrů:

```
Get-Random [[-Maximum] <Object>] [-Minimum <Object>]
  [-SetSeed <Int32>] [<CommonParameters>]

Get-Random [-InputObject] <Object[]> [-Count <Int32>]
  [-SetSeed <Int32>] [<CommonParameters>]
```

První sada parametrů přijímá parametry -Maximum, -Minimum a -SetSeed. Druhá sada parametrů přijímá parametry -InputObject, -Count a -SetSeed. Nelze proto kombinovat parametr -Count s parametrem -Minimum nebo -Maximum, protože patří do různých skupin parametrů (označovaných jako sady parametrů).

Poznámka: Je poměrně běžné, že rutiny prostředí Windows PowerShell mají více sad parametrů. Expanze tabulátoru nabízí pouze parametry z jedné sady parametrů. Když tedy u rutiny *Get-Random* zvolíte parametr -Count, nekompatibilní parametry se při expanzi tabulátoru nebudou objevovat. Tato funkce slouží jako prevence neplatných příkazů. Přehled sad parametrů rutin poskytuje rutina *Get-Help*.

Generování určitého počtu náhodných čísel

Rutina *Get-Random* v kombinaci s parametrem -Count přijímá také parametr -InputObject. Parametr -InputObject je poměrně silný. Následující citát ze souboru nápovědy uvádí, že parametr přijímá kolekci objektů:

```
-InputObject <Object[]>
    Specifies a collection of objects. Get-Random gets randomly
    selected objects in random order from the collection. Enter
    the objects, a variable that contains the objects, or a command
    or expression that gets the objects. You can also pipe a collection
    of objects to Get-Random.

    Required?                    true
    Position?                    1
    Default value
    Accept pipeline input?       true (ByValue)
    Accept wildcard characters?  False
```

Pole (nebo interval) čísel je současně také kolekcí objektů. Interval (neboli pole) čísel lze nejsnáze generovat pomocí *operátoru rozsahu*. Operátor rozsahu je tvořen dvěma tečkami mezi dvěma čísly. Jak je zřejmé z následujícího příkladu, operátor rozsahu nevyžaduje mezi čísly a tečkami uvádět mezery:

```
PS C:\> 1..5
1
2
3
4
5
```

Chcete-li tedy vybrat pět náhodných čísel z intervalu od 1 do 10, stačí pouze uvést příkaz. Číselný interval 1 až 10 je nutné uzavřít do závorek. Tím je zajištěno, že bude rozsah čísel vytvořen před výběrem pěti čísel z kolekce:

```
Get-Random -InputObject (1..10) -Count 5
```

Následující příklad uvádí příkaz a příslušný výstup:

```
PS C:\> Get-Random -InputObject (1..10) -Count 5
7
5
10
1
8
```

Používání nástrojů pro příkazový řádek

Prostředí Windows PowerShell se sice používá snadno, ale v některých případech je jednodušší vyhledat požadované informace pomocí nástroje pro příkazový řádek. Chcete-li například zjistit informace o konfiguraci IP adres, stačí jen spustit nástroj Ipconfig.exe. Tento příkaz lze zadat přímo do konzoly prostředí Windows PowerShell a výstup se zobrazí v konzole. Následující příklad uvádí příkaz a příslušný výstup ve zkrácené podobě:

```
PS C:\> ipconfig

Windows IP Configuration

Wireless LAN adapter Local Area Connection* 14:

    Media State . . . . . . . . . . . : Media disconnected

    Connection-specific DNS Suffix  . :

Ethernet adapter vEthernet (WirelessSwitch):

    Connection-specific DNS Suffix  . : quadriga.com

    Link-local IPv6 Address . . . . . : fe80::915e:d324:aa0f:a54b%31
```

```
    IPv4 Address. . . . . . . . . . . : 192.168.13.220

    Subnet Mask . . . . . . . . . . . : 255.255.248.0

    Default Gateway . . . . . . . . . : 192.168.15.254

Wireless LAN adapter Local Area Connection* 12:

    Media State . . . . . . . . . . . : Media disconnected

    Connection-specific DNS Suffix  . :

Ethernet adapter vEthernet (InternalSwitch):

    Connection-specific DNS Suffix  . :

    Link-local IPv6 Address . . . . . : fe80::bd2d:5283:5572:5e77%19

    IPv4 Address. . . . . . . . . . . : 192.168.3.228

    Subnet Mask . . . . . . . . . . . : 255.255.255.0

    Default Gateway . . . . . . . . . : 192.168.3.100

<VÝSTUP JE ZKRÁCEN>
```

Chcete-li získat stejné informace pomocí prostředí Windows PowerShell, musíte zadat složitější příkaz. Údaje o IP adresách lze zjistit příkazem *Get-NetIPAddress*. Poskytuje však několik výhod. V prvé řadě výstup příkazu IpConfig.exe je textový, zatímco rutiny prostředí Windows PowerShell poskytují objekty. To znamená, že je možné výstup snadno seskupovat, třídit, filtrovat a formátovat.

Velká výhoda spočívá v tom, že konzola prostředí Windows PowerShell nabízí jednoduchost příkazového řádku, ale zároveň i integrovaný výkonný jazyk prostředí Windows PowerShell. Jestliže tedy potřebujete třikrát aktualizovat zásady skupin a mezi aktualizacemi vyčkat pět minut, můžete zadat příkaz uvedený v následující ukázce (cykly se budeme zabývat v kapitole 11, „Používání skriptů prostředí Windows PowerShell“):

```
1..3 | % {gpupdate ; sleep 300}
```

Práce s možnostmi nápovědy

Chcete-li efektivně používat soubory nápovědy, musíte je nejdříve ve svém systému aktualizovat. Prostředí Windows PowerShell 3.0 totiž zavádí nový model, kde se soubory nápovědy pravidelně aktualizují.

Pokud chcete aktualizovat soubory nápovědy ve svém systému, musíte konzolu prostředí Windows PowerShell spustit s právy správce. Důvod spočívá v tom, že soubory nápovědy prostředí Windows PowerShell jsou umístěny v chráněné složce Windows\System32\

WindowsPowerShell. Po spuštění konzoly prostředí Windows PowerShell s právy správce je nutné zkontrolovat, zda je počítač připojen k Internetu, aby mohl stáhnout a nainstalovat aktualizované soubory. Jestliže počítač nemá internetovou konektivitu, vyprší časový limit příkazu až za několik minut, protože prostředí Windows PowerShell pokus o stažení aktualizovaných souborů několikrát opakuje. Spustíte-li rutinu *Update-Help* bez parametrů, prostředí Windows PowerShell se pokusí stáhnout aktualizované soubory nápovědy pro všechny moduly uložené ve výchozím umístění modulů prostředí Windows PowerShell, které podporují aktualizaci nápovědy. Pokud potřebujete rutinu *Update-Help* spustit vícekrát denně, zadejte parametr -Force jako v následující ukázce:

```
Update-Help -Force
```

Dokonce i bez stažení aktualizované nápovědy prostředí Windows PowerShell zobrazí podsystém nápovědy syntaxi rutiny a další základní informace o ní.

Chcete-li zobrazit informace nápovědy z Internetu, použijte parametr *-Online*. Příkaz v tomto tvaru způsobí, že prostředí Windows PowerShell otevře ve výchozím prohlížeči příslušnou stránku z webu Microsoft TechNet.

V podnikovém prostředí mohou správci sítě pomocí rutiny *Save-Help* stáhnout nápovědu z Internetu. Po stažení může rutina *Update-Help* odkazovat na sdílenou síťovou složku se soubory. Tento úkol lze snadno automatizovat a naplánovat jeho pravidelné spouštění.

Shrnutí

V této kapitole jsme začali přehledem prostředí Windows PowerShell. Konkrétně jsme uvedli některé rozdíly a podobnosti mezi konzolou prostředí Windows PowerShell a Integrovaným skriptovacím prostředím (ISE) v prostředí Windows PowerShell. Vysvětlili jsme si, že příkazy prostředí Windows PowerShell poskytují stejné výsledky bez ohledu na to, odkud jsou spuštěny.

Prostředí Windows PowerShell používá konvenci názvů skládajících se ze slovesa a podstatného jména. Informace lze načítat pomocí rutin, které začínají slovesem *Get*. Typ požadovaných informací určuje příslušné podstatné jméno. Jako příklad této konvence lze uvést rutinu *Get-HotFix*, která vrátí informace o opravách hotfix nainstalovaných v místním systému.

Je nutné znát jeden z hlavních principů prostředí Windows PowerShell: Umožní uživateli provést akci jen v případě, že to povoluje model zabezpečení. Pokud má například uživatel oprávnění zastavit službu pomocí nástroje Services.MSC, může službu zastavit i v rámci prostředí Windows PowerShell. Jestliže však uživatel nemá obecná oprávnění k zastavení služby, prostředí Windows PowerShell mu tuto operaci neumožní provést. Prostředí Windows PowerShell také spolupracuje s nástrojem UAC. V systémech Windows 7 a Windows 8 se prostředí Windows PowerShell ve výchozím nastavení spouští v režimu s nejnižšími systémovými oprávněními. Chcete-li provést akci, která vyžaduje práva správce, musíte prostředí Windows PowerShell spustit jako správce.

Mnoho rutin prostředí Windows PowerShell funguje bez zadání parametrů a vrací přitom platná data. K těmto rutinám patří *Get-Process* nebo *Get-Service*. Většina rutin prostředí Windows PowerShell však ke své činnosti vyžaduje další údaje. Rutina *Get-EventLog* například potřebuje název konkrétního protokolu událostí.

Po prvním přihlášení ke konzole prostředí Windows PowerShell je vhodné spustit rutinu *Update-Help*. Tato operace vyžaduje práva správce a připojení k Internetu.

KAPITOLA 2

Používání rutin prostředí Windows PowerShell

V této kapitole:
- Seznámení se základy rutin
- Vyhledávání v tématech nápovědy
- Nalezení rutin příkazem Get-Command
- Používání rutiny Get-Member
- Používání rutiny Show-Command
- Nastavení pravidel spouštění skriptů
- Vytvoření základního profilu prostředí Windows PowerShell
- Shrnutí

Když jste se seznámili s konvencí názvů rutin prostředí Windows PowerShell a víte, jak se rutiny prostředí Windows PowerShell používají, můžete na těchto znalostech stavět a zaměřit se na použití společných parametrů prostředí Windows PowerShell. Prostředí Windows PowerShell je mimo jiných výhod také intuitivní a srozumitelné. Jakmile si osvojíte způsob, kterým rutiny prostředí Windows PowerShell fungují, dokážete odhadnout, jak by se tyto rutiny měly chovat. Tento intuitivní návrh prostředí Windows PowerShell je jeho charakteristickým rysem a umožňuje využívat znalosti tohoto prostředí na různých platformách. Pokud se například naučíte, jak pomocí prostředí Windows PowerShell spravovat počítač se systémem Windows 8, dokážete zvládnout i práci s rutinami v systému Windows Server 2012. Rutiny prostředí Windows PowerShell by se totiž vždy měly chovat stejným způsobem.

Při seznamování s prostředím Windows PowerShell potřebujete čtyři základní nástroje: *Get-Help*, *Get-Command*, *Get-Member* a *Show-Command*. Tyto čtyři rutiny prostředí Windows PowerShell tvoří základy, na nichž můžete své znalosti tohoto prostředí rozvíjet. Jak budete s prostředím Windows PowerShell pracovat stále více, pokusíte se také konzolu prostředí Windows PowerShell přizpůsobit. Za tímto účelem bude nutné nejdříve upravit pravidla spouštění skriptů prostředí Windows PowerShell. Pravděpodobně to bude vaše první zásadní změna výchozího nastavení prostředí Windows PowerShell, kterou byste měli pečlivě zvážit.

Seznámení se základy rutin

Všechny rutiny prostředí Windows PowerShell se chovají v zásadě stejným způsobem. Rutiny různých dodavatelů nebo jednotlivých týmů ve společnosti Microsoft mají sice svá specifika, ale obecně platí, že rozumíte-li způsobu fungování rutin prostředí Windows PowerShell, dokážete tyto znalosti přenést i na další rutiny, platformy a aplikace. Chcete-li zavolat rutinu prostředí Windows PowerShell, zadejte její název na příkazový řádek v konzole prostředí Windows PowerShell. Jestliže chcete změnit způsob, kterým rutina načítá nebo zobrazuje informace, zadejte hodnoty parametrů, které chování rutiny upraví. Mnohé z těchto parametrů jsou unikátní a vztahují se pouze na určité rutiny. Jiné parametry však platí pro všechny rutiny prostředí Windows PowerShell. Částečně i díky těmto rutinám poskytuje návrh prostředí Windows PowerShell tolik možností. Těmito parametry, zvanými *společné parametry*, které jsou podporované všemi rutinami prostředí Windows PowerShell, se budeme zabývat v následující části kapitoly.

Společné parametry prostředí Windows PowerShell

Všechny rutiny prostředí Windows PowerShell jsou kompatibilní se společnými parametry, které jsou uvedeny v následujícím seznamu. Každý ze společných parametrů má také svůj alias, který lze uvádět místo parametru. Aliasy jednotlivých parametrů jsou zobrazeny v závorkách:

- Verbose (vb)
- Debug (db)
- WarningAction (wa)
- WarningVariable (wv)
- ErrorAction (ea)
- ErrorVariable (ev)
- OutVariable (ov)
- OutBuffer (ob)

Pokud rutina prostředí Windows PowerShell změní stav systému (například zastaví proces nebo změní spouštěcí hodnotu služby), zpřístupní se dva dodatečné parametry:

- WhatIf (wi)
- Confirm (cf)

Poskytování dodatečných informací parametrem Verbose

Jako příklad použití společných parametrů v prostředí Windows PowerShell lze uvést parametr -Verbose, který umožňují získat další údaje o akci, kterou rutina provádí.

Následující příkaz zastaví všechny instance procesu Notepad.exe spuštěné v místním systému (příkaz neposkytuje žádný výstup):

```
PS C:\> Stop-Process -Name notepad
PS C:\>
```

Chcete-li zjistit, které procesy byly zastaveny v důsledku spuštění rutiny *Stop-Process*, uveďte společný parametr -Verbose. V následujícím příkladu jsou kvůli spuštění rutiny *Stop-Process* zastaveny dvě samostatné instance procesu Notepad.exe. Vzhledem k tomu, že rutina přijímá společný parametr -Verbose, zobrazí se na výstupu podrobné informace o každém z procesů:

```
PS C:\> Stop-Process -Name notepad -Verbose
VERBOSE: Performing operation "Stop-Process" on Target "notepad (5564)".
VERBOSE: Performing operation "Stop-Process" on Target "notepad (5924)".
PS C:\>
```

Skrývání chyb parametrem ErrorAction

Když se pokusíte pomocí rutiny *Stop-Process* zastavit určitý proces a přitom není spuštěna žádná jeho instance, zobrazí se v konzole prostředí Windows PowerShell nepříjemná chyba. V následující ukázce se rutina *Stop-Process* pokouší zastavit proces s názvem Notepad.exe, ale v daném okamžiku není spuštěna žádná jeho instance. Zobrazí se proto následující chyba:

```
PS C:\> Get-Process -Name notepad
Get-Process : Cannot find a process with the name "notepad".
Verify the process name and call the cmdlet again.
At line:1 char:1
+ Get-Process -Name notepad
+ ~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          :
      ObjectNotFound: (notepad:String) [Get-Process], ProcessCommandException
    + FullyQualifiedErrorId :
      NoProcessFoundForGivenName,Microsoft.PowerShell.
      Commands.GetProcessCommand

PS C:\>
```

Pokud víte, nebo alespoň předpokládáte, že proces neběží, ale chtěli byste to ověřit, můžete uvést společný parametr -ErrorAction. Chcete-li skrýt chybové zprávy generované rutinou *Get-Process*, uveďte před spuštěním rutiny hodnotu SilentlyContinue parametru -ErrorAction. Tento postup je zřejmý z následující ukázky:

```
PS C:\> Get-Process -Name notepad -ErrorAction SilentlyContinue
PS C:\>
```

Poznámka: Předchozí příkaz je zdánlivě hodně dlouhý, ale nezapomínejte, že díky expanzi tabulátoru jej lze snadno zadat bez chyby. Ve skutečnosti se předchozí příkaz zadává tímto způsobem: *Get-Pro<tab><mezera>-n<tab><mezera>notepad<mezera>-e<tab><mezera>s<tab>*.

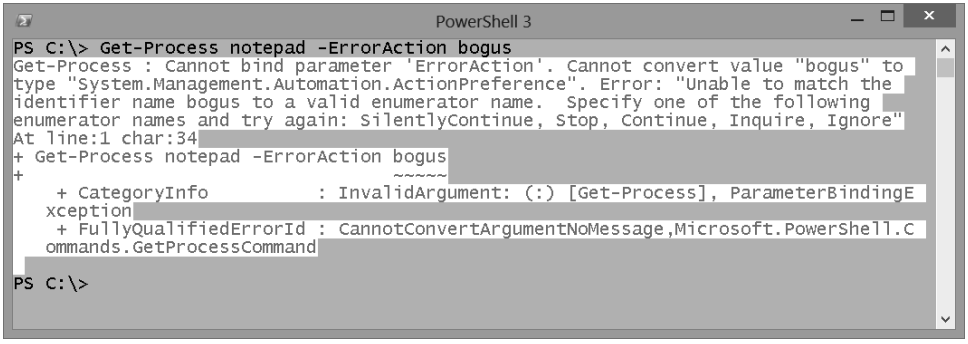
Příkaz můžete zkrátit pomocí aliasu parametru *-EA*, díky němuž není potřeba uvádět název -ErrorAction (ačkoli díky expanzi tabulátoru je potřeba zadat stejný počet znaků: *-E<tab>* nebo *-EA*). Při práci s rutinou *Get-Process* je navíc nastaven výchozí parametr *Name*. To znamená,

že parametr -Name rutiny *Get-Process* je výchozí, a rutina *Get-Process* tedy interpretuje každý řetězec na první pozici jako název procesu. Upravený příkaz je uveden v následujícím příkladu:

```
PS C:\> Get-Process notepad -ea SilentlyContinue

PS C:\>
```

Jestliže si nejste jisti, jaké jsou platné hodnoty parametru -ErrorAction, můžete uvést libovolnou hodnotu a poté si pečlivě přečíst výslednou chybovou zprávu. První dva řádky v textu chybové zprávy sdělují, že prostředí Windows PowerShell nedokáže převést hodnotu na typ *System.Management.Automation.ActionPreference*. Na čtvrtém řádku chybové zprávy jsou uvedeny povolené hodnoty parametru -ErrorAction. Jedná se o hodnoty *SilentlyContinue*, *Stop*, *Continue*, *Inquire* a *Ignore*. Tento postup s vynucením chyby je znázorněn na obrázku 2.1.



Obrázek 2.1: Záměrným vynucením chyby lze zjistit přípustné hodnoty parametrů prostředí Windows PowerShell

Spuštění přepisu prostředí Windows PowerShell

Mezi užitečné funkce konzoly prostředí Windows PowerShell patří rutina *Start-Transcript*. Chcete-li spustit přepis prostředí Windows PowerShell, zadejte na prázdný řádek v konzole prostředí Windows PowerShell příkaz *Start-Transcript*. Následující příklad uvádí příkaz a příslušný výstup:

```
PS C:\> Start-Transcript
Transcript started, output file is
C:\Users\ed.IAMMRED\Documents\PowerShell_transcript.20121120142251.txt
```

Po spuštění přepisu prostředí Windows PowerShell se všechny příkazy, výstupy příkazů a dokonce i chybové zprávy ukládají do souboru přepisu. Soubor přepisu lze využít několika způsoby. Některé z nich jsou uvedeny v následujícím seznamu:

- **Nástroj odstraňování potíží** – přepis slouží jako cenná pomůcka při řešení potíží, protože uvádí příkazy a konkrétní chybové zprávy, kterými prostředí na tyto příkazy reagovalo.
- **Výukový nástroj** – zkoumáte-li různé příkazy prostředí Windows PowerShell a naleznete nový užitečný příkaz, máte k dispozici záznam příkazů a příslušných výstupů.

- **Nástroj auditu** – přepis obsahuje uživatelská jména, časy a záznamy o všech příkazech a výstupy těchto příkazů.

Zastavení a prohlížení přepisu prostředí Windows PowerShell

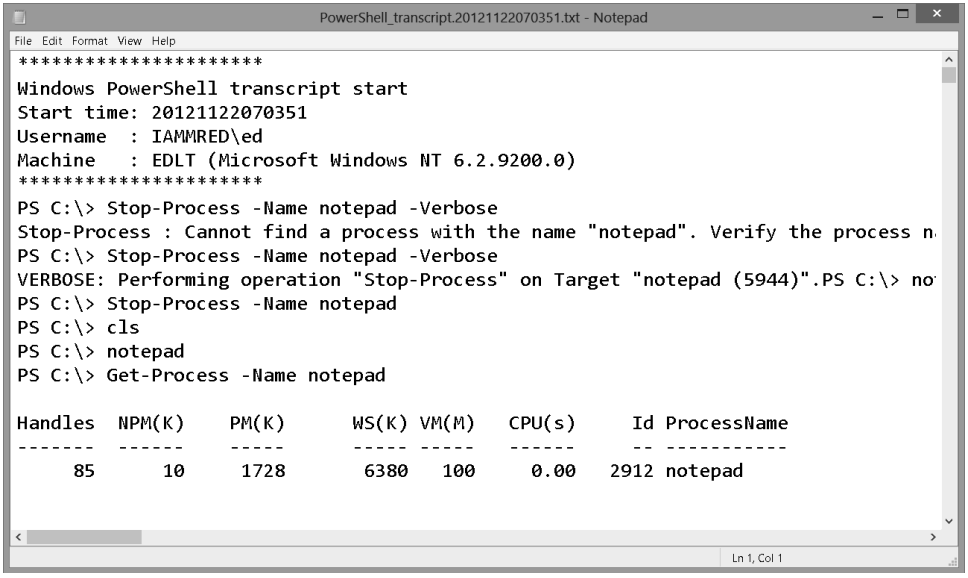
Přepis prostředí Windows PowerShell zahrnuje všechny příkazy a jejich výstupy včetně chyb. Chcete-li přepis prostředí Windows PowerShell zobrazit, musíte jej nejdříve zastavit. K tomu slouží rutina *Stop-Transcript*. Po zastavení přepisu se cesta k souboru přepisu objeví v konzole prostředí Windows PowerShell, jak je patrné v následujícím příkladu:

```
PS C:\> Stop-Transcript

Transcript stopped, output file is
C:\Users\ed.IAMMRED\Documents\PowerShell_transcript.20121122074731.txt

PS C:\>
```

Soubor přepisu lze analyzovat v prostředí Windows PowerShell (jedná se o neformátovaný text) nebo jej můžete otevřít v programu Notepad (Poznámkový blok). Na obrázku 2.2 vidíme soubor protokolu přepisu prostředí Windows PowerShell otevřený v programu Notepad.



Obrázek 2.2: Soubor přepisu prostředí Windows PowerShell lze snadno otevřít v programu Notepad a zkontrolovat příkazy i chyby

Vyhledávání v tématech nápovědy

V rámci prostředí Windows PowerShell jsou k dispozici dva typy témat nápovědy. Témata prvního typu popisují rutiny a způsob jejich použití. Témata nápovědy k rutinám obvykle obsahují základní popis rutiny, syntaxi, vysvětlení parametrů a příklady použití rutiny. Příklady obvykle leží v rozsahu od mimořádně jednoduchých až po středně složité ukázky, které zahrnují více rutin. Témata nápovědy k rutinám jsou dostupná pomocí rutiny *Get-Help* (nebo funkce *Help*) po zadání názvu rutiny.

Druhým typem témat nápovědy jsou koncepční témata. Koncepční témata nápovědy neobsahují více částí (jako např. popis, syntaxi či příklady). Místo toho se jedná o souvislé textové soubory, které nepopisují jednotlivé rutiny, ale spíše podrobně vysvětlují základní principy prostředí Windows PowerShell. K dispozici jsou například soubory nápovědy, které se týkají jazyka WQL (WMI Query Language), vzdálené komunikace s prostředím Windows PowerShell, pracovních toků, a dokonce i obsluhy chyb. Koncepční témata nápovědy jsou dostupná pomocí rutiny *Get-Help* (nebo funkce *Help*) po zadání výrazu začínajícího řetězcem „About_". Rutina *Get-Help* umožňuje použít expanzi tabulátoru a cyklicky procházet jednotlivá koncepční témata nápovědy. Chcete-li to provést, zadejte na příkazový řádek tuto sekvenci:

```
Get-Help About_ + <TAB> + <ENTER>
```

V mém počítači se systémem Windows 8 se aktuálně nachází asi 110 koncepčních témat About_ celkem se 47 190 slovy.

Poznámka: Než začnete pracovat s nápovědou prostředí Windows PowerShell, musíte spustit rutinu *Update-Help*, která stáhne aktuální soubory nápovědy. Příkaz ke svému spuštění vyžaduje práva správce (viz kapitola 1, „Přehled prostředí Windows PowerShell 3.0") a připojení k Internetu.

Používání rutiny Get-Help

Chcete-li v nápovědě najít informace o použití konkrétní rutiny prostředí Windows PowerShell, použijte rutinu *Get-Help* a uveďte název rutiny v parametru -Name. Když je rutina *Get-Help* spuštěna v tomto režimu, poskytuje základní informace nápovědy včetně následujících prvků: Name, Synopsis, Syntax, Description, Related Links a Remarks (název, přehled, syntaxe, popis, související odkazy a komentáře). Tento typ příkazu je uveden v následující ukázce:

```
Get-Help -Name Get-Process
```

Výchozím parametrem rutiny *Get-Help* je parametr -Name, a proto jej nemusíme zadávat pokaždé, když chceme najít nápovědu k rutině. Následující příkaz například rovněž zobrazí informace nápovědy k rutině *Get-Process*:

```
Get-Help Get-Process
```

Jakmile se seznámíte se základními aspekty fungování určité rutiny, postačí vám jen krátké připomenutí, jak s konkrétní rutinou pracovat. V těchto případech se hodí parametr *-Examples*. Spustíte-li rutinu *Get-Help* s parametrem -Examples, vrátí následující prvky témata nápovědy: Name, Synopsis a Examples (název, přehled a příklady). Tento příkaz je uveden v následující ukázce:

```
Get-help -name Get-Service -Examples
```

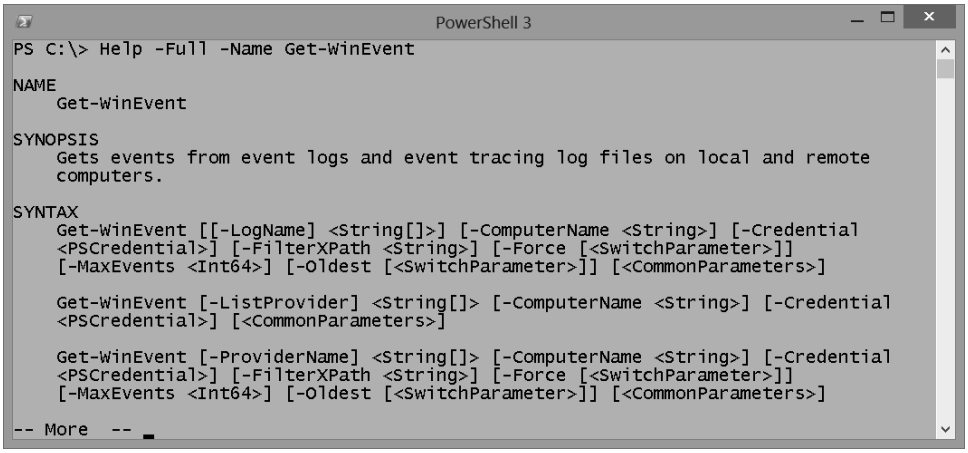
Když začnete zkoumat pokročilejší funkce rutin prostředí Windows PowerShell nebo potřebujete o dané rutině prostředí Windows PowerShell úplné informace, uveďte parametr *-Full*. Parametr *-Full* vynutí, že prostředí Windows PowerShell vrátí všechny dostupné informace nápovědy. K těmto informacím patří následující prvky: Name, Synopsis, Syntax, Description, Parameters, Inputs, Outputs, Notes, Examples, Related Links a Remarks (název, přehled, syntaxe, popis, parametry, vstupy, výstupy, poznámky, příklady, související odkazy a komentáře). Sekce Parameters kromě toho obsahuje údaje o tom, zda je parametr povinný, jakou má pozici, výchozí hodnoty a to, zda přijímá zřetězený vstup či zástupné znaky.

Stránkování výstupu nápovědy

Když pomocí rutiny *Get-Help* s parametrem -Full zobrazíte kompletní informace nápovědy o určité rutině, je vypsáno několik stránek. Ve výchozím nastavení se informace nápovědy posunují v okně konzoly prostředí Windows PowerShell. Pokud objem informací nepřekročí vyrovnávací paměť konzoly, můžete přejít zpět na začátek výstupu nápovědy a přečíst si kompletní text. Jestliže však rozsah informací přesáhne kapacitu vyrovnávací paměti konzoly, je nutné buď změnit nastavení vyrovnávací paměti, nebo odeslat výstup do stránkovače. Chcete-li, aby výstup zobrazil jednu stránku a pozastavil se, dokud nestisknete mezerník nebo Enter, použijte funkci *Help*. Pracuje podobně jako rutina *Get-Help*, protože přijímá stejné parametry. Rozdíl spočívá v tom, že informace najednou neposílá přímo do konzoly prostředí Windows PowerShell, ale odešle je do stránkovače, který je zobrazuje po jednotlivých stránkách. Stránkovač je pokročilý a na základě detekce velikosti konzoly prostředí Windows PowerShell určí objem údajů, které lze v jednu chvíli zobrazit. Máte-li tedy menší okno konzoly prostředí Windows PowerShell, zobrazí se najednou jen několik řádků. Pokud je okno konzoly prostředí Windows PowerShell větší, stránkovač zobrazí více řádků výstupu. Následující příkaz zobrazí po jednotlivých stránkách úplné informace nápovědy o rutině *Get-WinEvent*:

```
Help -Full -Name Get-WinEvent
```

Obrázek 2.3 dokládá, jak lze pomocí funkce Help zobrazit stránkované informace o rutině *Get-WinEvent*.



Obrázek 2.3: Funkce Help umožňuje zobrazit úplné informace nápovědy po jednotlivých stránkách

Hledání rutin prostředí Windows PowerShell pomocí rutiny Get-Help

Díky tomu, že rutina *Get-Help* přijímá zástupné znaky, lze pomocí této rutiny vyhledávat rutiny týkající se určitého tématu. Chcete-li například pomocí rutiny *Get-Help* najít rutiny související s procesy, použijte příkaz podobný následující ukázce:

```
Get-Help *process
```

Výhoda spočívá v tom, že se díky zástupným znakům můžete bez dlouhého psaní zaměřit na určitý druh rutin. Tento postup je zřejmý z následující ukázky:

```
PS C:\> get-help *P*ce?s

Name              Category    Module                      Synopsis
----              --------    ------                      --------
Debug-Process     Cmdlet      Microsoft.PowerShell.M...   Debugs one ...
Get-Process       Cmdlet      Microsoft.PowerShell.M...   Gets the pr...
Start-Process     Cmdlet      Microsoft.PowerShell.M...   Starts one ...
Stop-Process      Cmdlet      Microsoft.PowerShell.M...   Stops one o...
Wait-Process      Cmdlet      Microsoft.PowerShell.M...   Waits for t...
```

Pokud vás zajímá konkrétně nápověda rutiny, zadejte při použití příkazu *Get-Help* kategorii rutiny, jak je zřejmé v následujícím příkladu:

```
Get-Help process -Category cmdlet
```

Použití koncepčních témat nápovědy typu About

Pomocí rutiny *Get-Help* s parametrem -Name lze vyhledávat koncepční témata nápovědy typu About_. Zástupné znaky však hledání komplikují a mohou způsobit, že některá témata nebudou nalezena. Následuje příklad použití zástupných znaků:

```
PS C:\> Get-Help -Name about*wmi*

Name                Category   Module              Synopsis
----                --------   ------              --------
about_WMI           HelpFile
about_WMI_Cmdlets   HelpFile                       Provides ba...
```

Bez koncového zástupného znaku příkaz vrátí pouze téma about_WMI. Lepší způsob hledání v koncepčních tématech nápovědy typu About_ je založen na zadání parametru HelpFile *-Category*. Tento příkaz je čistší a kromě toho se snáze zadává. Následující příkaz uvádí příkaz a příslušný výstup:

```
PS C:\> Get-Help -Name wmi -Category HelpFile

Name                Category   Module              Synopsis
----                --------   ------              --------
about_WMI           HelpFile
about_WMI_Cmdlets   HelpFile                       Provides ba...
```

Poznámka: Předchozí příkaz je zdánlivě delší. Když však používáte expanzi tabulátoru pro název rutiny, název parametru a povolené hodnoty parametrů, lze příkaz ve skutečnosti zadat pomocí menšího počtu stisknutí kláves než příkaz se zástupnými znaky.

Chcete-li najít koncepční témata nápovědy týkající se společných parametrů, použijte rutinu *Get-Help* a při výběru kategorie *HelpFile* zadejte slovo „common“, jak je zřejmé v následující ukázce:

```
PS C:\> Get-Help common -Category HelpFile

Name                              Category   Module      Synopsis
----                              --------   ------      --------
about_CommonParameters            HelpFile               Describes t...
about_ActivityCommonParameters    HelpFile               Describes t...
about_WorkflowCommonParameters    HelpFile               This topic ...
```

Některá koncepční témata nápovědy jsou poměrně dlouhá. Proto může být vhodnější místo rutiny *Get-Help* použít funkci *Help*. Lze přitom uplatnit stejný postup. Jestliže například chcete najít koncepční témata nápovědy související s operátory, stačí při zadávání hodnoty *HelpFile* do parametru -Category zadat pouze část klíčového slova „operator“, jak je uvedeno v následujícím příkladu:

```
PS C:\> Help oper -Category HelpFile

Name                              Category   Module        Synopsis
----                              --------   ------        --------
about_Arithmetic_Operators        HelpFile                 Describes   t...
about_Assignment_Operators        HelpFile                 Describes   h...
about_Comparison_Operators        HelpFile                 Describes   t...
about_Logical_Operators           HelpFile                 Describes   t...
about_Operators                   HelpFile                 Describes   t...
```

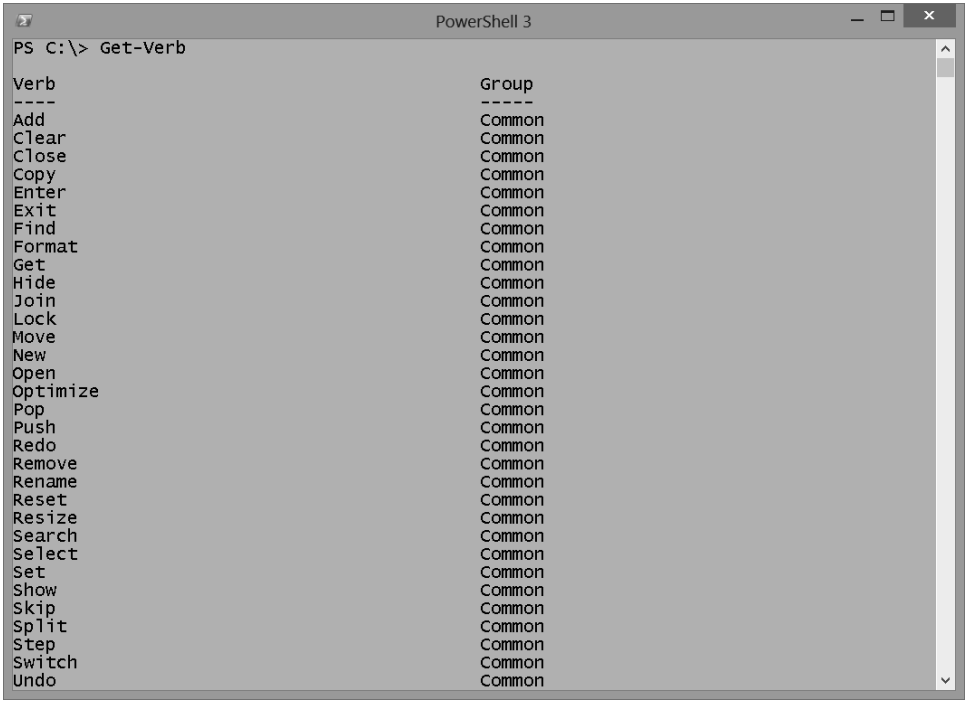
```
about_Operator_Precedence       HelpFile                Lists the   W...
about_Properties                HelpFile                Describes   h...
about_Type_Operators            HelpFile                Describes   t...
```

Nalezení rutin příkazem Get-Command

Existuje několik různých metod, jak pomocí rutiny *Get-Command* najít rutiny prostředí Windows PowerShell. Nejjednodušší způsob je založen na rutině *Get-Command*, která umožňuje najít rutiny používající určité sloveso jako Get – viz následující příklad:

```
Get-Command -Verb Get
```

V prostředí Windows PowerShell 3.0 se používá 98 autorizovaných sloves. Seznam povolených sloves se zobrazí při použití funkce *Get-Verb*. Obrázek 2.4 znázorňuje použití funkce *Get-Verb* spolu s částí příslušného výstupu.



Obrázek 2.4: Pomocí rutiny Get-Verb lze rozbalit seznam schválených sloves prostředí Windows PowerShell

Chcete-li načítat, shromažďovat nebo sbírat informace, pravděpodobně byste měli uvést sloveso Get. V tomto případě použijte rutinu *Get-Command* a hodnotu „get“ uveďte v parametru -Verb, jak je patrné v následující ukázce:

```
Get-Command -Verb get
```

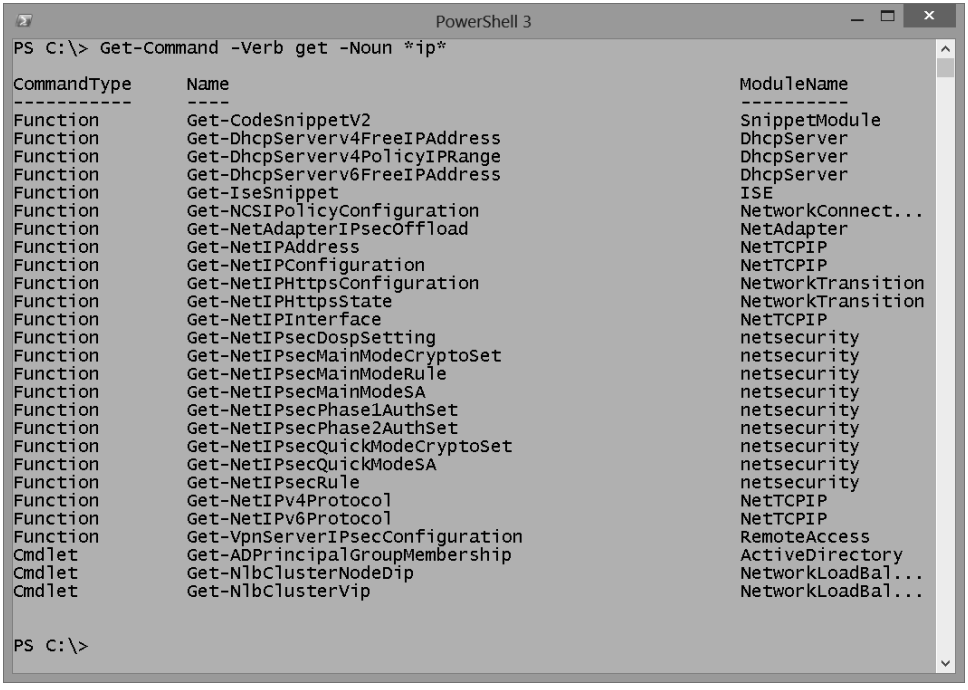
V mém počítači se systémem Windows 8 a nainstalovanými nástroji RSAT (Remote Server Administration Tools – nástroje pro vzdálenou správu serveru) systému Windows Server 2012 vrací příkaz se slovesem Get 577 rutin (a funkcí). Vzhledem k velkému počtu rutin (a funkcí) Get je pravděpodobně vhodné uvést současně i parametr -Noun, který omezí počet položek na výstupu. Většina názvů rutin prostředí Windows PowerShell souvisí s funkcí nebo technologií, kterou spravují, a tak lze hledání rutin usnadnit pomocí parametru -Noun. Následující příklad znázorňuje, jak použít parametr -Noun k hledání rutin, které obsahují písmena TCP:

```
PS C:\> Get-Command -Verb get -Noun *tcp*

CommandType     Name                                               ModuleName
-----------     ----                                               ----------
Function        Get-NetTCPConnection                               NetTCPIP
Function        Get-NetTCPSetting                                  NetTCPIP


PS C:\>
```

Pokud vás však zajímají rutiny týkající se protokolu IP, mohou vás výsledky hledání se zástupnými znaky poněkud zklamat. Je to způsobeno tím, že vzor písmen *ip* se vyskytuje v názvech mnoha rutin (a funkcí). Příkaz a příslušný výstup je znázorněn na obrázku 2.5.



Obrázek 2.5: Když je součástí názvu rutiny krátký znakový vzor, nejsou zástupné znaky efektivní

Chcete-li najít rutiny týkající se konkrétně určité technologie, uveďte parametr -Module. Rutiny (a funkce) prostředí Windows PowerShell se totiž nacházejí v modulech a tyto moduly seskupují funkce správy podle jednotlivých technologií. Jestliže tedy potřebujete spravovat protokoly TCP/IP, použijte rutiny z modulu *NetTCPIP*, jak je zřejmé v následujícím příkladu:

```
PS C:\> Get-Command -Verb get -Noun *ip* -Module NetTcpIp

CommandType     Name                               ModuleName
-----------     ----                               ----------
Function        Get-NetIPAddress                   NetTCPIP
Function        Get-NetIPConfiguration             NetTCPIP
Function        Get-NetIPInterface                 NetTCPIP
Function        Get-NetIPv4Protocol                NetTCPIP
Function        Get-NetIPv6Protocol                NetTCPIP
```

Všechny rutiny (funkce) obsažené v modulu *NetTCPIP* lze zjistit pomocí rutiny *w* s uvedením samotného parametru -Module. Díky tomu se ve výstupu objeví všechna podstatná jména i slovesa. Příkaz je uveden v následující ukázce:

```
Get-Command -Module NetTcpIp
```

Pamatujte, že název modulu je možné dokončit pomocí expanze tabulátoru. Lze také zadávat zástupné znaky.

Pokud vás zajímá konkrétně konfigurace objektů, obvykle se používá sloveso Set. Následující ukázka uvádí funkce prostředí Windows PowerShell z modulu *NetTCPIP*, které obsahují sloveso Set:

```
PS C:\> Get-Command -Module NetTCPIP -Verb set

CommandType     Name                               ModuleName
-----------     ----                               ----------
Function        Set-NetIPAddress                   NetTCPIP
Function        Set-NetIPInterface                 NetTCPIP
Function        Set-NetIPv4Protocol                NetTCPIP
Function        Set-NetIPv6Protocol                NetTCPIP
Function        Set-NetNeighbor                    NetTCPIP
Function        Set-NetOffloadGlobalSetting        NetTCPIP
Function        Set-NetRoute                       NetTCPIP
Function        Set-NetTCPSetting                  NetTCPIP
Function        Set-NetUDPSetting                  NetTCPIP
```

Používání rutiny Get-Member

Chcete-li najít vlastnosti, metody nebo události objektu v prostředí Windows PowerShell, použijte rutinu *Get-Member*. Vlastnosti objektu popisují daný objekt. Automobily mají například vlastnosti jako barva, konstrukční typ, spotřeba paliva a cena. Metody zajišťují jednotlivé akce. Automobil má například metody „rozjeď se“, „zastav na červené“ a často také „pusť hudbu“.

Vše v prostředí Windows PowerShell je objekt. Ve skutečnosti je prostředí Windows PowerShell mezi prostředími příkazových řádků a skriptování jedinečné tím, že vrací a řetězí objekty. Většina jiných prostředí příkazových řádků a skriptování je textového typu. U textových prostředí je nutné při hledání informací analyzovat text pomocí samostatných nástrojů. To je jeden z problémů při používání starých příkazů (jako je *ping*, *ipconfig* a *tracert*) v rámci prostředí Windows PowerShell: Příkazy vracejí text, ze kterého lze obtížně získat konkrétní informaci.

Rutinu *Get-Member* lze používat mnoha různými způsoby. Rutina má parametry, které umožňují filtrovat výsledky a hledat určité parametry. Chcete-li zjistit, kterými způsoby lze pracovat s rutinou *Get-Member*, použijte rutinu *Get-Help*, kterou jsme se zabývali v předchozí části kapitoly. Následující příklad ukazuje příkaz, který vrací základní údaje o rutině *Get-Member*:

```
Get-Help Get-Member
```

Chcete-li si prohlédnout pouze příklady použití rutiny *Get-Member*, zadejte příkaz následujícím způsobem:

```
Get-Help Get-Member -Examples
```

Úplné informace nápovědy získáte pomocí tohoto příkazu:

```
Get-Help Get-Member -Full
```

Zkoumání členských vlastností

Chcete-li prozkoumat členské vlastnosti objektu, použijte rutinu *Get-Member* a pomocí parametru -MemberType zvolte členský typ Property. Kromě toho je nutné zadat hodnotu parametru InputObject. Parametr InputObject přijímá objekt, jehož členské vlastnosti chcete zobrazit. Kvůli tomu, aby byl objekt vytvořen před zobrazením členských vlastností, je nutné název rutiny produkující objekty umístit do závorek.

Následující příkaz vytvoří objekt *System.Datetime* spuštěním rutiny *Get-Date*. Poté zobrazí členské vlastnosti pomocí rutiny *Get-Member*:

```
PS C:\> Get-Member -InputObject (get-date) -MemberType Property

   TypeName: System.DateTime

Name          MemberType   Definition
----          ----------   ----------
Date          Property     datetime Date {get;}
Day           Property     int Day {get;}
DayOfWeek     Property     System.DayOfWeek DayOfWeek {get;}
DayOfYear     Property     int DayOfYear {get;}
Hour          Property     int Hour {get;}
Kind          Property     System.DateTimeKind Kind {get;}
Millisecond   Property     int Millisecond {get;}
Minute        Property     int Minute {get;}
Month         Property     int Month {get;}
Second        Property     int Second {get;}
```

```
Ticks          Property     long Ticks {get;}
TimeOfDay      Property     timespan TimeOfDay {get;}
Year           Property     int Year {get;}
```

Poznámka: Pokud má parametr *-MemberType* hodnotu *Property*, liší se od parametru *-MemberType* s hodnotou *Properties*. První varianta vypíše pouze vlastnosti objektu. Druhá uvádí vlastnosti objektu, ale kromě toho také vlastnosti *ScriptProperty*, *NoteProperty*, *CodeProperty* a další.

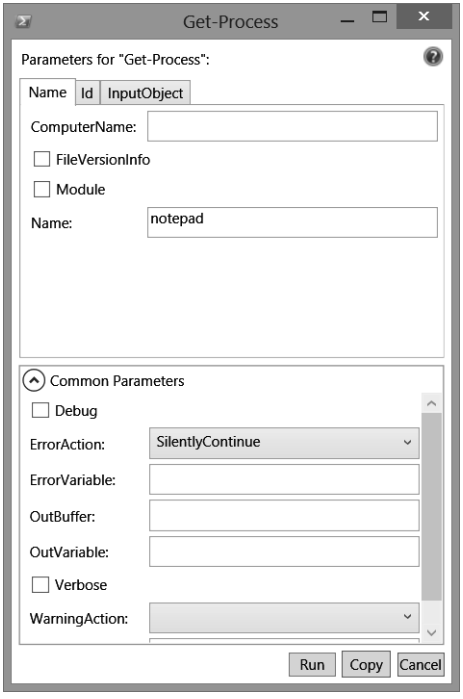
Používání rutiny Show-Command

Rutina *Show-Command* zobrazuje grafický vstupní ovladač rutiny, která je určena parametrem -Name. Následující příkaz zobrazí grafický vstupní ovladač rutiny *Get-Process*:

```
Show-Command -Name Get-Process
```

Vzhledem k tomu, že parametr -Name je výchozím parametrem rutiny *Show-Command*, není nutné jej pokaždé uvádět. Funguje tedy i následující příkaz, který parametr -Name vynechává:

```
Show-Command Get-Process
```



Obrázek 2.6: Grafický ovladač rutiny Get-Process vytvořený pomocí rutiny Show-Command

Po svém spuštění příkaz zobrazí grafický vstupní ovladač. Podoba ovladače závisí na rutině, která je zadána v rámci příkazu. Vstupní ovladač rutiny *Get-Process* znázorňuje tři různé sady

parametrů, každou na samostatné kartě. Na první kartě je zobrazena výchozí sada parametrů (Name). Pomocí rutiny *Show-Command* lze vytvořit příkaz pomocí myši nebo klávesnice, které umožňují vybrat a zadat parametry příkazu. Po vytvoření lze příkaz zkopírovat do schránky nebo spustit. Zvolíte-li možnost spuštění příkazu, nejdříve se úplný příkaz prostředí Windows PowerShell zobrazí na příkazovém řádku konzoly. Poté je spuštěn a zobrazí výsledky na dalších řádcích. Obrázek 2.6 znázorňuje vstupní ovladač rutiny *Get-Process*.

Nastavení pravidel spouštění skriptů

Prostředí Windows PowerShell ve výchozím nastavení nepovoluje spouštění skriptů. Podpora skriptů se obvykle řídí zásadami skupin. Pokud to neplatí a máte práva pro správu místního počítače, můžete podporu skriptů zapnout pomocí rutiny *Set-ExecutionPolicy* prostředí Windows PowerShell. Rutina *Set-ExecutionPolicy* umožňuje nastavit jednu ze šesti úrovní podpory. (Rozbor skriptů prostředí Windows PowerShell začíná v kapitole 10.) Možnosti jsou uvedeny v následujícím seznamu:

- **Restricted** – prostředí nenačítá konfigurační soubory ani nespouští skripty. Úroveň Restricted je výchozí.
- **AllSigned** – vyžaduje, aby všechny skripty a konfigurační soubory podepsal důvěryhodný vydavatel. Vztahuje se to i na skripty vytvořené v místním počítači.
- **RemoteSigned** – vyžaduje, aby důvěryhodný vydavatel podepsal všechny skripty a konfigurační soubory stažené z Internetu.
- **Unrestricted** – načítá všechny konfigurační soubory a spouští všechny skripty. Pokud spustíte nepodepsaný skript, který jste stáhli z Internetu, zobrazí se před spuštěním skriptu žádost o povolení.
- **Bypass** – nic není blokováno a nezobrazují se žádná upozornění ani výzvy.
- **Undefined** – odebere aktuálně přiřazené pravidlo spouštění z aktuálního oboru. Tento parametr neodebere pravidlo spouštění, které je nastaveno v oboru zásad skupin.

Poznámka: Pokud je pravidlo spouštění skriptů nastaveno pomocí zásad skupin, nemůže je změnit dokonce ani správce místního počítače.

Kromě šesti úrovní pravidel spouštění existují tři různé obory pravidel spouštění:

- **Process** – pravidlo spouštění se vztahuje pouze na aktuální proces prostředí PowerShell.
- **CurrentUser** – pravidlo spouštění se vztahuje pouze na aktuálního uživatele.
- **LocalMachine** – pravidlo spouštění ovlivňuje všechny uživatele počítače.

Chcete-li nastavit pravidlo spouštění *LocalMachine*, musíte mít práva správce místního počítače. Následující příkaz uvádí, jakým způsobem může místní správce změnit pravidlo spouštění skriptů na hodnotu *unrestricted*:

```
Set-ExecutionPolicy -Scope LocalMachine -ExecutionPolicy unrestricted
```

Uživatelé bez zvýšených oprávnění mají ve výchozím nastavení právo nastavit pravidlo spouštění skriptů pro obor uživatele *CurrentUser*, který ovlivňuje jejich vlastní pravidlo spouštění. Následující příkaz dokumentuje, jak může uživatel bez zvýšených oprávnění nastavit pravidlo spouštění skriptů na hodnotu *remotesigned*:

```
PS C:\> Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy remotesigned

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust.
Changing the execution policy might expose you to the security risks
described in the about_Execution_Policies help topic at
http://go.microsoft.com/fwlink/?LinkID=135170.
Do you want to change the execution policy?
[Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"): y
```

Vzhledem k velkému počtu dostupných pravidel spouštění skriptů možná přemýšlíte nad tím, která je pro vás nejvhodnější. Vývojový tým prostředí Windows PowerShell doporučuje nastavení *RemoteSigned*, které podle jeho názoru vyhovuje většině případů. Pamatujte, že i když popisy jednotlivých nastavení pravidel obsahují termín *Internet*, nemusí se vždy vztahovat na web, a dokonce ani na umístění za místním firewallem. Prostředí Windows PowerShell totiž původ skriptu určuje podle nastavení zóny prohlížeče Windows Internet Explorer. To znamená, že do internetové zóny patří vše, co pochází z jiného než místního počítače. Nastavení zóny prohlížeče Internet Explorer lze změnit v nastavení aplikace, v registru nebo pomocí zásad skupin.

Nechcete-li, aby se při změnách pravidla spouštění skriptů v prostředí Windows PowerShell 3.0 zobrazovala potvrzovací zpráva, zadejte parametr -Force.

Pokud chcete zobrazit pravidla spouštění pro všechny obory, uveďte při volání rutiny *Get-ExecutionPolicy* parametr -List, jak je zřejmé v následujícím příkladu:

```
PS C:\> Get-ExecutionPolicy -List

                           Scope                    ExecutionPolicy
                           -----                    ---------------
                  MachinePolicy                          Undefined
                     UserPolicy                          Undefined
                        Process                          Undefined
                   CurrentUser                       RemoteSigned
                  LocalMachine                         Restricted
```

Vytvoření základního profilu prostředí Windows PowerShell

S nastavením pravidel spouštění skriptů prostředí Windows PowerShell, kterými jsme se zabývali v předchozí části, je vhodné se seznámit proto, že profil prostředí Windows PowerShell patří mezi skripty. Ve skutečnosti se jedná o speciální skript, který má určitý název a nachází se ve vyhrazeném umístění. Z tohoto hlediska se profil prostředí Windows PowerShell podobá