Microsoft

Ed Wilson

Windows PowerShell® 5.0
Krok po kroku

Wydanie trzecie

Przekład: Natalia Chounlamany, Marek Włodarz

APN Promise, Warszawa 2016

Windows PowerShell® 5.0 Krok po kroku

Authorized Polish translation of the English language edition entitled: Windows PowerShell® Step by Step, Third Edition, ISBN 978-0-7356-7511-7, by Ed Wilson, published by Pearson Education, Inc, publishing as Microsoft Press, A Division Of Microsoft Corporation.

Polish language edition published by APN PROMISE SA Copyright © 2016

Autoryzowany przekład z wydania w języku angielskim, zatytułowanego: Windows PowerShell® Step by Step, Third Edition, ISBN 978-0-7356-7511-7, by Ed Wilson, opublikowanego przez Pearson Education, Inc, publikującego jako Microsoft Press, oddział Microsoft Corporation.

APN PROMISE SA, biuro: ul. Kryniczna 2, 03-934 Warszawa tel. +48 22 35 51 600, fax +48 22 35 51 699 e-mail: mspress@promise.pl

ISBN: 978-83-7541-179-9

Przekład: Natalia Chounlamany, Marek Włodarz
Korekta: Ewa Swędrowska
Skład i łamanie: MAWart Marek Włodarz

Spis treści

Wprowadzenie

Windows PowerShell to de facto standard dla administratorów systemów Windows. W ramach programu Microsoft Engineering Common Criteria wsparcie dla funkcji zarządczych Windows PowerShell zapewniają wszystkie produkty serwerowe m.in. Microsoft SQL Server, Exchange, System Center oraz SharePoint. Znajomość, a nawet biegłe opanowanie, tej technologii jest już nie tyle „mile widziane", lecz stanowi wręcz podstawowe wymaganie w ofertach pracy. Książka *Windows PowerShell 5.0 Krok po kroku* oferuje solidny fundament dla specjalisty IT, który chce być na bieżąco z tą kluczową technologią zarzadzania.

Kto powinien przeczytać tę książkę

Książka ta ma na celu szybko przygotować specjalistów IT do efektywnego wykorzystywania technik Windows PowerShell 5.0. Książka *Windows PowerShell 5.0 Krok po kroku* jest adresowana szczególnie do następujących osób:

- **Konsultanci sieci Windows** Zainteresowani standaryzacją i automatyzacją procesów instalacyjnych oraz konfiguracją komponentów sieciowych Microsoft .NET.
- **Administratorów sieci Windows** Zainteresowani automatyzacją codziennych zadańzarządzania systemami Windows lub komponentami sieciowymi .NET.
- **Microsoft Certified Solutions Experts (MCSEs) oraz Microsoft Certified Trainers (MCTs)** Windows PowerShell jest kluczowym komponentem wielu cykli szkoleniowych i egzaminów certyfikacyjnych firmy Microsoft.
- **Ogólny personel techniczny** Zainteresowani gromadzeniem informacji i konfigurowaniem ustawieńna komputerach Windows.
- **Użytkownicy zaawansowani** Zainteresowani uzyskaniem maksymalnej mocy i konfigurowalności swojego komputera Windows, czy to w domu, czy w niezarządzanym środowisku pracy.

Założenia

Autor oczekuje, że Czytelnik zna system operacyjny Windows, toteż nie zamieszcza wyjaśnieńbazowych terminów sieciowych czy systemowych. Książka zakłada też, że Czytelnik ma przynajmniej podstawowe przygotowanie w zakresie programowania

obiektowego i rozumie pojęcia i stosowalność takich technik, jak metody, właściwości czy rzutowanie. Przydatna będzie też przynajmniej podstawowa znajomość platformy .NET. Bardziej szczegółowe zagadnienia dotyczące programowania obiektowego, projektowania i skryptowania są wyjaśniane, gdy się pojawią.

Być może nie jest to książka dla Ciebie, jeśli...

Książka ta nie jest przewodnikiem referencyjnym po Windows PowerShell 5.0, ale ma na celu wprowadzić Czytelnika w świat PowerShell i zachęcić go do dalszych eksperymentów i poznawania środowiska na własną rękę. Oznacza to jednak, że szczególnie pogłębione, ezoteryczne tematy nie zostały w niej omówione. Choć pojawiają się niektóre zaawansowane zagadnienia, w ogólności zaczynamy od bardzo podstawowych pojęć i zmierzamy w kierunku bardziej złożonych zagadnień

Z drugiej strony, jeśli ktoś nigdy nie używał komputera i nie orientuje się w elementarnych pojęciach dotyczących systemów operacyjnych, sieci czy programowania, ta książka zdecydowanie nie jest przeznaczona dla niego.

Struktura książki

Niniejszą książkę można podzielić na trzy części. Pierwsza część poświęcona jest wierszowi polecenia Windows PowerShell. Druga zawiera omówienie skryptów Windows PowerShell. W trzeciej części znaleźć można opis bardziej zaawansowanych technik Windows PowerShell, a także możliwości stosowania Windows PowerShell do realizowania różnych zadań zarządczych. Ta trzyczęściowa struktura jest umowna i nie została odzwierciedlona w konkretnym podziale na strony, jednak może pomóc w przystąpieniu do lektury tej dość długiej książki.

Niniejsza książka nie stanowi szczegółowego kompendium wiedzy, a raczej ogólny przegląd możliwości Windows PowerShell. Każdy rozdział oferuje nowe doświadczenia, techniki i umiejętności. Osoby zainteresowane wybranym tematem mogą wykorzystać zdobytą wiedzę, przystępując do bardziej zaawansowanych szkoleń. Na przykład, czytelnicy, których w wyniku lektury rozdziału 21 zafascynuje funkcja DSC, powinni pamiętać, że poznali dopiero przedsmak jej możliwości. Istnieje szereg specjalistów MVP Windows PowerShell koncentrujących się na tym jednym aspekcie technologii Windows PowerShell.

W którym miejscu rozpocząć lekturę

Poszczególne części książki *Windows PowerShell 5.0 Krok po kroku* poświęcone są różnym technologiom. W zależności od własnych potrzeb i posiadanej znajomości narzędzi Microsoft można skoncentrować się na różnych częściach książki. Poniższa tabela może pomóc w zaplanowaniu lektury.

Jeśli jesteś	Postępuj następująco
Nowicjuszem Windows PowerShell	Skoncentruj się na rozdziałach 1–3 oraz 5–9 lub przeczytaj całą książkę od początku do końca.
Specjalistą IT, który zna podstawy Windows PowerShell i chce jedynie nauczyć się zarządzać zasobami sieciowymi	Przewertuj rozdziały 1–3, jeśli potrzebujesz przypomnieć sobie podstawowe zagadnienia. Poczytaj o nowych technologiach w rozdziałach 4, 14 oraz 20–22.
Zainteresowany Active Directory	Przeczytaj rozdziały 15–17.
Zainteresowany skryptami Windows PowerShell	Przeczytaj rozdziały 5–8, 18 oraz 19.
Zaznajomiony z wersją Windows PowerShell 3.0	Przeczytaj rozdział 1, przejrzyj rozdziały 8 i 18, a następnie przeczytaj rozdziały 20–22.
Zaznajomiony z wersją Windows PowerShell 4.0	Przeczytaj rozdział 1, przejrzyj rozdziały 8, 18 i 21, a następnie przeczytaj rozdział 22.

Wszystkie rozdziały zawierają dwa ćwiczenia, w których można krok po kroku przetestować zdobyte umiejętności.

Wymagania systemowe

Do wykonania ćwiczeń praktycznych zawartych w książce potrzebne będą następujące elementy sprzętowe i programowe:

- Windows 10, Windows 7, Windows Server 2012 R2, Windows Server 2012, Windows Server 2008 R2 lub Windows Server 2008 z Service Pack 2.
- Komputer z procesorem 1,6 GHz lub szybszym (zaleca się 2 GHz)
- 1 GB (w systemie 32-bitowym) lub 2 GB (64-bitowym) pamięci RAM
- 3,5 GB dostępnego miejsca na dysku
- Napęd dyskowy 5400 RPM lub szybszy
- Karta graficzna wspierająca DirectX 9 o rozdzielczości 1024 x 768 lub wyższej
- Połączenie z Internetem w celu pobrania oprogramowania lub omawianych przykładów

W zależności od konfiguracji Windows do uruchomienia pewnych poleceńniezbędne mogą być uprawnienia administratora lokalnego.

Pobieranie skryptów

Większość rozdziałów książki zawiera przykłady kodu, które można wpisywać własnoręcznie, aby wypróbować nowe umiejętności i techniki. Wszystkie przykładowe skrypty można pobrać z następującej strony:

http://aka.ms/PS3E/files

Z tej strony należy pobrać plik PS3E_675117_Scripts.zip.

Instalowanie skryptów

Postępuj zgodnie z poniższą instrukcją w celu zainstalowania na swoim komputerze skryptów, które można wykorzystać w prezentowanych w tej książce ćwiczeniach.

1. Rozpakuj plik PS3E_675117_Scripts.zip pobrany z witryny książki.
2. Jeśli wyświetlony zostanie monit, przejrzyj umowę licencyjną. Jeśli akceptujesz jej warunki, zaznacz opcję Accept, a następnie kliknij przycisk Next.

Korzystanie ze skryptów

Pliki skryptów zostaną rozpakowane do folderów, których nazwy odpowiadają rozdziałom książki.

Podziękowania

Chciałbym podziękować za pomoc następującym osobom: moim redaktorom Kathy Krause oraz Jaime Odell z firmy OTSI, za przekształcenie tekstu w coś przypominającego prawdziwą angielszczyznę i wskazywanie mi różnych konwencji Microsoft; mojemu recenzentowi i przyjacielowi Brian Wilhite, Microsoft PFE, którego spostrzegawczość ochroniła mój profesjonalny wizerunek; Jason Walker z Microsoft Consulting Services oraz Gary Siepser i Ashley McGlone z Microsoft PFE, którzy przejrzeli mój konspekt i dostarczyli wiele sugestii. Na koniec wreszcie chcę podziękować mojej żonie Teresie Wilson, Windows PowerShell MVP (znanej także jako Scripting Wife), która przeczytała każdą stronę i dodała liczne sugestie, które będą bardzo przydatne początkującym skrypciarzom.

Errata, aktualizacje i wsparcie dla książki

Dołożyliśmy wszelkich starań, aby zapewnić poprawność tej książki i dołączonych do niej materiałów. Errata tej książki będzie dostępna na stronie:

http://aka.ms/PS3E/errata

Czytelników, którzy odkryją błędy niewyszczególnione na liście, prosimy o ich zgłoszenie za pośrednictwem tej samej strony.

Dodatkowe informacje można uzyskać, wysyłając email do działu Microsoft Press Book Support:

mspinput@microsoft.com

Należy mieć jednak na uwadze, że powyższe adresy nie służą do uzyskiwania pomocy technicznej w zakresie działania oprogramowania oraz sprzętu firmy Microsoft. Do tego celu służy strona:

http://support.microsoft.com

Darmowe e-booki wydawnictwa Microsoft Press

Wydawnictwo Microsoft Press oferuje różne darmowe e-booki, począwszy od ogólnych przeglądów technicznych po dogłębne analizy specjalistycznych zagadnień. Są one dostępne w formacie PDF, EPUB lub Mobi for Kindle i gotowe do pobrania za strony:

http://aka.ms/mspressfree

Warto regularnie tam zaglądać w poszukiwaniu nowych tytułów!

Państwa opinia jest dla nas cenna

Wydawnictwo Microsoft Press na pierwszym miejscu stawia satysfakcję czytelników, dlatego Państwa zdanie jest dla nas bardzo ważne. Prosimy o wyrażenie opinii na temat tej książki na poniższej stronie:

http://aka.ms/tellpress

Zdajemy sobie sprawę z tego, jak zajęci bywają nasi czytelnicy, dlatego ograniczyliśmy się do kilku pytań Odpowiedzi są przesyłane bezpośrednio do redaktorów w wydawnictwie Microsoft Press (ankiety są anonimowe). Z góry dziękujemy za opinie!

Zostańmy w kontakcie

Pozostańmy w stałym kontakcie. Można nas znaleźć w serwisie Twitter: *http://twitter.com/MicrosoftPress*.

ROZDZIAŁ 1

Przegląd cech Windows PowerShell 5.0

Po zapoznaniu się z tym rozdziałem Czytelnik będzie umiał:

- Określić podstawowe zastosowania i możliwości Windows PowerShell.
- Zainstalować Windows PowerShell.
- Korzystać z podstawowych narzędzi wiersza polecenia wewnątrz Windows PowerShell
- Korzystać z systemu pomocy.
- Uruchamiać podstawowe polecenia cmdlet.
- Uzyskiwać pomoc na temat poleceńcmdlet.

Wersja Windows PowerShell 5.0 zapewnia ogromne możliwości administratorom sieci systemów Windows. Dzięki połączeniu siły dojrzałego języka skryptowego z dostępem do narzędzi wiersza poleceń, interfejsów Windows Management Instrumentation (WMI), a nawet VBScript (Microsoft Visual Basic Scripting Edition), Windows PowerShell zapewnia skuteczność i łatwość stosowania. Implementacja setek poleceń cmdlet oraz zaawansowanych funkcji tworzy środowisko, w którym można dokonywać skomplikowanych zmian za pomocą prostych, czytelnych linii kodu. Jako część standardu Microsoft Common Engineering Criteria, Windows PowerShell szybko staje się podstawowym rozwiązaniem zarządzania dla platformy Windows.

Istota Windows PowerShell

Zapewne największą przeszkodą dla wielu administratorów sieci w przejściu do korzystania z Windows PowerShell 5.0 jest niezrozumienie, co to naprawdę jest. Pod wieloma względami jest to zamiennik dla sędziwej powłoki CMD (command – polecenie). I rzeczywiście, w systemie Windows Server 2012 w trybie Core możliwe jest zastąpienie powłoki CMD przez Windows PowerShell, tak że po uruchomieniu serwera używa on PowerShell jako domyślnego interfejsu użytkownika.

Jak można się przekonać, po uruchomieniu Windows PowerShell można używać w nim dobrze znanych poleceń, na przykład *cd* do zmiany katalogu roboczego lub *dir* do wyświetlenia jego zawartości – tak samo jak w oknie CMD.

```
PS C:\Windows\System32> cd\
PS C:\> dir

    Directory: C:\

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        7/10/2015   7:07 PM                FSO
d-----         7/9/2015   5:24 AM                PerfLogs
d-r---         7/9/2015   6:59 AM                Program Files
d-r---        7/10/2015   7:27 PM                Program Files (x86)
d-r---        7/10/2015   7:18 PM                Users
d-----        7/10/2015   6:00 PM                Windows

PS C:\>
```

Można również połączyć tradycyjne polecenia interpretera CMD z innymi narzędziami, takimi jak *fsutil*, co prezentuje kolejny przykład:

```
PS C:\> md c:\test

    Directory: C:\

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        7/11/2015  11:14 AM                test

PS C:\> fsutil file createnew c:\test\myfile.txt 1000
File c:\test\myfile.txt is created
PS C:\> cd c:\test
PS C:\test> dir

    Directory: C:\test

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        7/11/2015  11:14 AM           1000 myfile.txt

PS C:\test>
```

Przykłady te prezentują interaktywne wykorzystanie Windows PowerShell. Interaktywność jest jedną z głównych funkcji Windows PowerShell. Aby użyć PowerShell interaktywnie, wystarczy otworzyć jego okno zachęty i zacząć wpisywać polecenia – po jednym na raz, albo łącząc je ze sobą, podobnie jak w pliku wsadowym. Do tego tematu powrócę później, gdyż najpierw muszę przedstawić bardziej podstawowe informacje.

Korzystanie z poleceń cmdlet

Oprócz aplikacji konsolowych systemu Windows i wbudowanych poleceń w środowisku PowerShell dostępne są polecenia *cmdlet* (wymawiane jako *„komandlet”*), budujące jego prawdziwą potęgę. Polecenia cmdlet mogą być tworzone przez każdego. Zespół programistów Windows PowerShell w firmie Microsoft przygotował podstawowy zbiór poleceń, ale oprócz tego powstały setki poleceń opracowanych przez inne grupy projektowe. Większość poleceń jest dołączona do systemu Windows 10, aczkolwiek niektóre pakiety stanowią część specjalistycznych programów, takich jak Microsoft Exchange czy Microsoft SQL Server. Polecenia cmdlet są plikami wykonywalnymi, które wykorzystują funkcjonalności wbudowane w środowisko PowerShell i dzięki temu są tak łatwe do napisania. Nie są to skrypty zawierające nieskompilowany kod – są zbudowane przy wykorzystaniu usług specjalnej przestrzeni nazw Microsoft .NET Framework. Windows PowerShell 5.0 w wersji dla Windows 10 wyposażony jest w około 1300 poleceń cmdlet i w miarę dodawania nowych funkcji lub ról dołączane są kolejne. Polecenia te mają na celu zapewnienie administratorom sieci czy konsultantom dostępu do siły Windows PowerShell bez konieczności tworzenia złożonych skryptów. Jedną z głównych przewag PowerShell jest to, że wszystkie polecenia wykorzystują standardową konwencję nazewniczą określaną terminem czasownik-rzeczownik (verb-noun), na przykład *Get-Help* ("otrzymaj pomoc"), *Get-EventLog* lub *Get-Process*. Polecenia używające czasownika Get (weź, pobierz) wyświetlają informacje o elemencie umieszczonym po prawej stronie łącznika. Polecenia używające czasownika Set (ustal, określ) powodują ustawienie odpowiednich cech elementu po prawej stronie łącznika. Na przykład polecenie cmdlet *Set-Service* pozwala (między innymi) zmienić tryb uruchomieniowy usługi (ang. service). Wszystkie polecenia używają jednego ze standardowych czasowników. W celu wyświetlenia listy tych czasowników można użyć polecenia cmdlet *Get-Verb*. W Windows PowerShell 5.0 występuje około stu zaakceptowanych czasowników.

Instalowanie Windows PowerShell

Windows PowerShell 5.0 jest dołączony do systemu operacyjnego Windows 10. W witrynie Microsoft Download Center dostępny jest pakiet Windows Management Framework 5.0, który zawiera zaktualizowane wersje Windows Remote Management (WinRM), WMI oraz Windows PowerShell 5.0. Nie istnieje pakiet dla systemu Windows 10, gdyż oprogramowanie jest zawarte w samym systemie. Instalacja pakietu Windows Management Framework 5.0 w systemach Windows 7, Windows 8.1, Windows Server 2008 R2, Windows Server 2012 oraz Windows Server 2012 R2 wymaga uruchomienia .NET Framework 4.5.

Wdrażanie Windows PowerShell w starszych systemach operacyjnych

Po pobraniu pakietu instalacyjnego z witryny *http://www.microsoft.com/downloads* można wdrożyć go na komputerach przedsiębiorstwa wykonując jedno z poniższych działań:

- Utworzyć pakiet instalacyjny Windows Management Framework i ogłosić go dla odpowiedniej jednostki organizacyjnej (OU) lub zbioru jednostek.
- Utworzyć obiekt zasad grupy (GPO) instalacji oprogramowania w usługach AD DS i połączyć go z odpowiednią jednostką organizacyjną.
- Dołączyć pakiet jako zaaprobowany do serwera SUS (Software Update Services), jeśli jest dostępny.
- Umieścić pakiet Windows Management Framework 5.0 w centralnym udziale plikowym, aby użytkownicy mogli go zainstalować we własnym zakresie.

Jeśli wdrożenie odbywa się na pojedynczym komputerze, a nie w całym przedsiębiorstwie, najprostszą metodą wdrażania będzie pobranie pakietu i przejście przez kroki kreatora instalacji.

Uwaga Aby móc zacząć używać narzędzi wiersza poleceń Windows PowerShell, należy uruchomić powłokę, wybierając Start | Run | PowerShell. Pojawi się okno polecenia ze znakiem zachęty PS>, w którym można zacząć wpisywać polecenia.

Korzystanie z narzędzi wiersza poleceń

Jak wspomniałem wcześniej, z wnętrza Windows PowerShell można bezpośrednio wywoływać narzędzia trybu wiersza poleceń. Przewagą wykorzystania PowerShell jako środowiska dla tych narzędzi, w przeciwieństwie do korzystania z tradycyjnego interpretera CMD, są funkcje przekierowywania wyjścia i formatowania dostępne w PowerShell. Dodatkowo, jeśli mamy już pliki wsadowe (.cmd lub .bat) wykorzystujące istniejące narzędzia wiersza poleceń, można je łatwo zmodyfikować, aby mogły być uruchamiane w środowisku PowerShell. Poniższa procedura ilustruje dołączenie wyników polecenia ipconfig do pliku tekstowego.

➜ Wywoływanie polecenia *ipconfig*

1. Uruchom powłokę Windows PowerShell, wybierając Start | Run | Windows PowerShell. Okno poleceń PowerShell otworzy się domyślnie w głównym folderze zalogowanego użytkownika np. C:\Users\Ed.
2. Wpisz polecenie **ipconfig /all**:

```
PS C:\> ipconfig /all
```

3. Przekieruj wyniki polecenia *ipconfig /all* do pliku tekstowego, jak poniżej:

```
PS C:\> ipconfig /all >ipconfig.txt
```

4. Otwórz utworzony plik w Notatniku, aby przejrzeć jego zawartość:

```
PS C:\> notepad ipconfig.txt
```

Wpisywanie pojedynczych poleceń jest oczywiście użyteczne, ale zazwyczaj potrzebujemy więcej niż jednego, aby zrealizować założony cel – na przykład zgromadzić informacje konfiguracyjne potrzebne do rozwiązania problemów z instalacją lub niedostateczną wydajnością. I tu objawia się przewaga Windows PowerShell. W przeszłości w takim przypadku trzeba było albo napisać plik wsadowy, albo wpisywać kolejne polecenia ręcznie. Oto przykład prostego pliku wsadowego rejestrującego informacje przydatne do rozwiązywania problemów z siecią:

TroubleShoot.bat

```
ipconfig /all >C:\tshoot.txt
route print >>C:\tshoot.txt
hostname >>C:\tshoot.txt
net statistics workstation >>C:\tshoot.txt
```

Oczywiście przy ręcznym wpisywaniu poleceń musielibyśmy czekać na wykonanie każdego polecenia przed wywołaniem kolejnego. W takiej sytuacji łatwo można „zagubić się" we właściwej sekwencji lub wpisać kolejne polecenie przed zakończeniem przetwarzania poprzedniego, co doprowadzi do bezużytecznych wyników. Windows PowerShell eliminuje ten problem. Można teraz wpisać wiele poleceń w jednym wierszu, po czym odejść od komputera lub zająć się innymi zadaniami, podczas gdy system generuje wyniki. Nie ma potrzeby tworzenia pliku wsadowego w tym celu.

Wskazówka Można wpisywać wiele poleceń w jednym wierszu Windows PowerShell, aby zostały wykonane po kolei. Poszczególne polecenia należy rozdzielać średnikami.

Poniższy przykład prezentuje wywołanie wielu poleceń w jednym wierszu.

➜ Uruchamianie wielu poleceń

1. Otwórz okno poleceń Windows PowerShell, wybierając Start | Run (Uruchom) | PowerShell. Okno poleceń Windows PowerShell otworzy się domyślnie w głównym folderze zalogowanego użytkownika.
2. Wpisz polecenie **ipconfig /all**, przesyłając wynik do pliku tekstowego *Tshoot.txt* przy użyciu znaku przekierowania (>):

```
ipconfig /all >tshoot.txt
```

3. W tym samym wierszu wpisz średnik i dopisz polecenie *print*. Wynik tego polecenia dodaj do pliku *Tshoot.txt*, używając znaku przekierowania z dołączeniem (>>). Wiersz polecenia będzie wyglądał następująco:

```
ipconfig /all >tshoot.txt; route print >>tshoot.txt
```

4. Nadal w tym samym wierszu wstaw kolejny średnik i dodaj polecenie *hostname*, ponownie kierując wynik do pliku *Tshoot.txt*, używając znaku przekierowania z dołączeniem:

```
ipconfig /all >tshoot.txt; route print >>tshoot.txt; hostname >>tshoot
.txt
```

5. Dołącz polecenie *net statistics workstation*, oddzielając je od poprzednich znakiem średnika i dołączając wyjście polecenia do pliku *Tshoot.txt*. Ostateczna postać wiersza polecenia będzie wyglądać następująco:

```
ipconfig /all >tshoot.txt; route print >>tshoot.txt; hostname >>tshoot
.txt; net statistics workstation >>tshoot.txt
```

Problemy dotyczące zabezpieczeń

Biorąc pod uwagę ogromną elastyczność i wszechstronność Windows PowerShell, powstają nieuniknione obawy dotyczące zabezpieczeń systemowych. Jednak problemy te zostały uwzględnione już na etapie projektowania powłoki PowerShell.

Po uruchomieniu Windows PowerShell domyślnie wybierany jest główny folder bieżącego użytkownika. Gwarantuje to, że znajdujemy się w katalogu, w którym dysponujemy uprawnieniami do wykonywania szeregu działań. Jest to znacznie bezpieczniejsze, niż zaczynanie od głównego katalogu dysku lub katalogu systemowego.

Wykonywanie skryptów jest domyślnie zablokowane i można nim łatwo zarządzać poprzez Zasady grupy. Możliwe jest również włączenie tej funkcjonalności dla określonego użytkownika lub sesji.

Kontrolowanie wykonywania poleceń cmdlet

Każdemu chyba zdarzyło się otworzyć interpreter CMD, wpisać jakieś polecenie i nacisnąć Enter, aby sprawdzić, co ono robi. Jednak co by było, gdyby tym poleceniem było *Format C:*? Czy na pewno chcielibyśmy sformatować dysk C w naszym komputerze? W tym podrozdziale pokażę niektóre parametry, które pozwalają sterować sposobem wykonywania poleceń cmdlet. Choć nie wszystkie polecenia obsługują te parametry, robi to większość poleceń dołączonych do Windows PowerShell. Trzy parametry przełącznika (nazywane również parametrami przełącznika), których można użyć do sterowania wykonywaniem, to *-WhatIf*, *-Confirm* oraz *suspend*. *Suspend* nie jest w istocie parametrem przełącznika przekazywanym do polecenia cmdlet, ale

działaniem, które można podjąć po zgłoszeniu potwierdzenia, a zatem dodatkową metodą kontrolowania wykonywania poleceń

Uwaga Aby użyć parametru *-WhatIf*, należy wpisać całe polecenie cmdlet i dopisać po nim **-***WhatIf* na końcu listy parametrów. Opcja ta jest dostępna tylko dla poleceńzmieniających stan systemu. Z tego względu nie można (ani nie ma potrzeby) stosowania parametru *-WhatIf* dla poleceńinformacyjnych, takich jak *Get-Process*.

Polecenia cmdlet zmieniające stan systemu (na przykład *Set-Service*) wspierają *tryb prototypowy*, który jest włączany poprzez użycie parametru przełącznika *-WhatIf*. To, czy dane polecenie obsługuje ten parametr, jest zależne od programisty tworzącego cmdlet. Zespół projektowy Windows PowerShell zaleca implementację parametru *-WhatIf* we wszystkich poleceniach, które mogą dokonywać zmian. Poniższy przykład prezentuje wykorzystanie parametru przełącznika *-WhatIf*:

➜ Użycie parametru *-WhatIf* do prototypowania polecenia

1. Otwórz okno Windows PowerShell, o ile nie zostało uruchomione wcześniej.
2. Uruchom program Notatnik. W tym celu wpisz **notepad** i naciśnij Enter, jak pokazano poniżej:

```
notepad
```

3. Zidentyfikuj proces związany z uruchomioną sesją Notatnika, używając polecenia cmdlet *Get-Process*. Wpisz dostateczną część nazwy procesu, aby ją zidentyfikować, uzupełniając pozostałą część gwiazdką (*) – w ten sposób nie trzeba wpisywać całej (niekiedy długiej) nazwy procesu.

```
Get-Process note*
```

4. Przejrzyj wynik polecenia *Get-Process*, aby znaleźć identyfikator procesu. Wynik polecenia z mojego komputera jest przedstawiony poniżej. Trzeba pamiętać, że niemal na pewno numer procesu na innym komputerze będzie się różnił od pokazanego:

```
Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
114      8         1544       8712 ...54      0.00   3756 notepad
```

5. Użyj parametru *-WhatIf*, aby się przekonać, co by się stało po użyciu polecenia *Stop-Process* wobec identyfikatora znalezionego w punkcie 4 (w kolumnie *Id* wyniku). Parametr *-Id* pozwala podać identyfikator procesu w poleceniu. Polecenie to w moim przypadku wygląda następująco:

```
Stop-Process -id 3756 -whatif
```

6. Przejrzyj wynik polecenia. Informuje ono, że jego „prawdziwe” wykonanie spowoduje zatrzymanie procesu notepad o identyfikatorze wskazanym w poleceniu:

```
What if: Performing the operation "Stop-Process" on target "notepad (3756)".
(What if: Wykonanie operacji "Stop-Process" wobec celu "notepad (3756)")
```

Potwierdzanie akcji

Jak wiemy już z poprzedniego podpunktu, możemy wykorzystać parametr *-WhatIf* do wykonania polecenia w trybie prototypowym. W ten sposób możemy dowiedzieć się, co polecenie zrobi. Jeśli jednak chcemy być monitowani przed rzeczywistym wykonaniem polecenia, możemy użyć parametru *-Confirm*.

➜ Potwierdzanie wykonania poleceń

1. Otwórz okno Windows PowerShell, uruchom program Notatnik i zidentyfikuj jego proces, tak jak w krokach 1 – 4 poprzedniego ćwiczenia.
2. Użyj parametru *-Confirm*, aby wymusić monitowanie dla polecenia *Stop-Process* w celu zatrzymania procesu Notepad zidentyfikowanego przy pomocy polecenia *Get-Process note**.

```
Stop-Process -id 3756 -confirm
```

   Polecenie *Stop-Process* wywołane z parametrem *-Confirm* wyświetli następujący monit.

```
Confirm
Are you sure you want to perform this action?
Performing operation "Stop-Process" on Target "notepad (3756)".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"):
(Potwierdź
Czy na pewno chcesz wykonać to działanie?
Tak  Tak na wszystkie  Nie  Nie na wszystkie  Wstrzymaj  Pomoc (domyślnie Tak))
```

3. Wpisz **y** i naciśnij Enter. Notatnik zostanie zamknięty, a Windows PowerShell wyświetli znak zachęty, jak poniżej:

```
PS C:\>
```

Wskazówka W celu wstrzymania wykonywania polecenia należy w monicie potwierdzenia wpisać s i nacisnąć Enter.

Wstrzymywanie potwierdzania poleceń

Możliwość wyświetlania monitu o potwierdzenie przed wykonaniem polecenia jest niezwykle użyteczna i niekiedy może być krytycznym elementem utrzymywania najwyższego możliwego poziomu dostępności systemu. Łatwo można sobie wyobrazić sytuację, gdy wpisujemy długie, skomplikowane polecenie, po czym przypominamy sobie, że najpierw trzeba sprawdzić coś innego. Na przykład możemy chcieć zatrzymać wiele procesów, ale musimy przejrzeć ich szczegóły, aby się upewnić, że wymusimy zatrzymanie nieodpowiedniego procesu. W takiej sytuacji, zamiast kasować pracowicie wpisany wiersz (i musieć wprowadzić go ponownie później), możemy posłużyć się opcją wstrzymania wykonania polecenia.

➜ Wstrzymywanie wykonywania polecenia cmdlet

1. Otwórz okno Windows PowerShell, uruchom program Notatnik i zidentyfikuj jego proces, tak jak w krokach 1 – 4 ćwiczenia „Użycie parametru *-WhatIf* do prototypowania polecenia". Wynik na moim komputerze jest pokazany poniżej. Trzeba pamiętać, że niemal na pewno identyfikator procesu na innym komputerze będzie się różnił od pokazanego:

```
Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
     39       2      944        400    29     0.05   3576 notepad
```

2. Użyj parametru *-Confirm*, aby wymusić monitowanie przed wykonaniem polecenia *Stop-Process*, jak poniżej:

```
Stop-Process -id 3576 -confirm
```

   Polecenie wyświetli monit podobny do pokazanego w poprzednim przykładzie.

3. Aby wstrzymać wykonanie polecenia *Stop-Process*, wpisz **s**. Znak zachęty PowerShell zmieni się na „podwójną strzałkę", jak poniżej:

```
PS C:\>>
```

4. Użyj polecenia *Get-Process*, aby uzyskać listę uruchomionych procesów o nazwach zaczynających się na literę *n*, jak poniżej:

```
Get-Process n*
```

   Na moim komputerze występowały dwa takie procesy: uruchomiony przeze mnie proces Notepad oraz jeszcze jeden, związany z jedną z usług systemowych:

```
Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    269     168     4076       2332 ...98     0.19   1632 NisSrv
    114       8     1536       8732 ...54     0.02   3576 notepad
```

5. Powróć do monitu potwierdzenia, wpisując **exit** i naciskając Enter.

   Monit o potwierdzenie zostanie wyświetlony ponownie.

6. Wpisz **y** i naciśnij Enter, aby zatrzymać proces Notepad. Nie ma już dalszych potwierdzeń, zatem znak zachęty zmieni się na standardowy, jak poniżej:

   ```
   PS C:\>
   ```

Posługiwanie się Windows PowerShell

W tej części zajmiemy się uzyskiwaniem dostępu do powłoki Windows PowerShell i konfigurowaniem konsoli.

Wywoływanie Windows PowerShell

Po zainstalowaniu Windows PowerShell w systemie Windows jest on natychmiast dostępny do użycia. Jednak ciągłe sięganie do menu Start czy też używanie sekwencji klawiszy Windows+R w celu wywołania monitu *run* jest czasochłonne i niewygodne. W przypadku systemu Windows 10 problem jest nieco mniejszy, gdyż można po prostu wpisać **PowerShell** na ekranie Start. Dla większej wygody wolałem przypiąć Windows PowerShell oraz PowerShell ISE do ekranu Start.

W systemie Windows Server 2012 R2 w trybie Core zdecydowałem się zastąpić domyślną powłokę CMD konsolą Windows PowerShell. Jest to idealne rozwiązanie dla mnie i mojego sposobu pracy, zatem napisałem skrypt, który to realizuje. Skrypt ten może być wywoływany podczas logowania, aby automatycznie wykonać niezbędne zmiany. W systemie Windows 10 skrypt dodaje skróty do konsoli Windows PowerShell i PowerShell ISE do ekranu Start oraz paska zadań pulpitu. W Windows 7 skróty tworzone są w pasku zadań i menu Start. Skrypt ten działa tylko w systemach w języku angielskim. Aby móc użyć go w systemach lokalizowanych w innych językach, trzeba zmienić wartości zmiennych *$pinToStart* oraz *$pinToTaskBar* na odpowiedniki w tych językach.

Uwaga Korzystanie ze skryptów Windows PowerShell omówione jest w rozdziale 5, „Używanie skryptów PowerShell". Rozdział ten zawiera szczegółowe informacje, jak działają skrypty i jak należy je uruchamiać

Skrypt nosi nazwę PinToStart.ps1 i wygląda następująco:

PinToStart.ps1

```
$pinToStart = "Pin to Start"

$file = @((Join-Path -Path $PSHOME  -childpath "PowerShell.exe"),
          (Join-Path -Path $PSHOME  -childpath "powershell_ise.exe") )
Foreach($f in $file)
```

```
{$path = Split-Path $f
 $shell=New-Object -com "Shell.Application"
 $folder=$shell.Namespace($path)
 $item = $folder.parsename((Split-Path $f -leaf))
 $verbs = $item.verbs()
 foreach($v in $verbs)
   {if($v.Name.Replace("&","") -match $pinToStart){$v.DoIt()}}
```

Konfigurowanie konsoli Windows PowerShell

Konsola Windows PowerShell ma wiele konfigurowalnych aspektów. Możliwe elementy można umieścić w pliku konfiguracyjnym PSconsole. Aby wyeksportować bieżącą konfigurację do pliku, można użyć polecenia *Export-Console*, jak w poniższym przykładzie:

```
PS C:\> Export-Console myconsole
```

Plik PSConsole zostanie zapisany w bieżącym katalogu z domyślnym rozszerzeniem .psc1. Jest to plik w formacie XML. Domyślny (niezmodyfikowany) plik konfiguracji konsoli został przedstawiony poniżej:

```
<?xml version="1.0" encoding="utf-8"?>
<PSConsoleFile ConsoleSchemaVersion="1.0">
  <PSVersion>5.0.10224.0</PSVersion>
  <PSSnapIns />
</PSConsoleFile>
```

➜ Kontrolowanie opcji uruchamiania PowerShell

1. Okno PowerShell można uruchomić bez paska tytułu, używając argumentu *-NoLogo*, jak poniżej:

   ```
   PowerShell -nologo
   ```

2. Aby uruchomić konkretną wersję Windows PowerShell (przy założeniu, że jest zainstalowana na komputerze), można użyć argumentu *-Version*:

   ```
   PowerShell -version 3
   ```

3. Uruchomienie Windows PowerShell przy użyciu wskazanego pliku konfiguracyjnego wymaga użycia argumentu *-PSConsoleFile*, jak poniżej:

   ```
   PowerShell -psconsolefile myconsole.psc1
   ```

4. Możliwe jest uruchomienie Windows PowerShell w celu wykonania określonego polecenia, po czym nastąpi samoczynne wyjście (zamknięcie konsoli). W tym celu należy użyć argumentu *-Command*. Samo polecenie do wykonania musi być poprzedzone znakiem & i ujęte w nawiasy klamrowe, jak poniżej:

   ```
   Powershell -command "& {Get-Process}"
   ```

Przekazywanie opcji do poleceń cmdlet

Jedną z najbardziej użytecznych cech Windows PowerShell jest standaryzacja składni używanej przez polecenia cmdlet. Znacząco upraszcza to naukę obsługi powłoki i języka. Tabela 1-1 zawiera listę najczęściej stosowanych parametrów. Trzeba pamiętać, że nie wszystkie parametry są implementowane we wszystkich poleceniach. Jeśli jednak dany parametr jest zaimplementowany, zostanie zinterpretowany w taki sam sposób przez każde polecenie, gdyż interpretację wykonuje sam silnik PowerShell.

Tabela 1-1 Typowe parametry

Parametr	Znaczenie
-WhatIf	Powoduje, że polecenie nie jest wykonywane, ale wyświetlana jest informacja, co nastąpiłoby po jego wykonaniu (tryb prototypu).
-Confirm	Nakazuje wyświetlanie monitu przed właściwym wykonaniem polecenia.
-Verbose	Powoduje wyświetlanie bardziej szczegółowych informacji o wykonaniu polecenia.
-Debug	Powoduje wyświetlanie informacji debugowania.
-ErrorAction	Nakazuje wykonanie określonego działania, gdy wystąpi błąd. Dopuszczalne akcje to *Continue* (kontynuuj), *Ignore* (ignoruj), *Inquire* (pytaj), *SilentlyContinue* (kontynuuj bez powiadamiania), *Stop* (zatrzymaj) oraz *Suspend* (wstrzymaj).
-ErrorVariable	Poleca użycie określonej zmiennej do przechowania informacji o błędzie. Jest to uzupełnienie standardowej zmiennej *$Error*.
-OutVariable	Nakazuje przekierowanie informacji wyjściowych do wskazanej zmiennej.
-OutBuffer	Nakazuje przechowanie określonej liczby obiektów przed wywołaniem następnego polecenia cmdlet w ciągu przekazywania (pipeline).

Uwaga Aby uzyskać informacje pomocy na temat dowolnego polecenia cmdlet, należy użyć polecenia *Get-Help <nazwa cmdlet>*. Na przykład *Get-Help Get-Process* zwraca informacje o składni i stosowaniu polecenia *Get-Process*.

Korzystanie z opcji pomocy

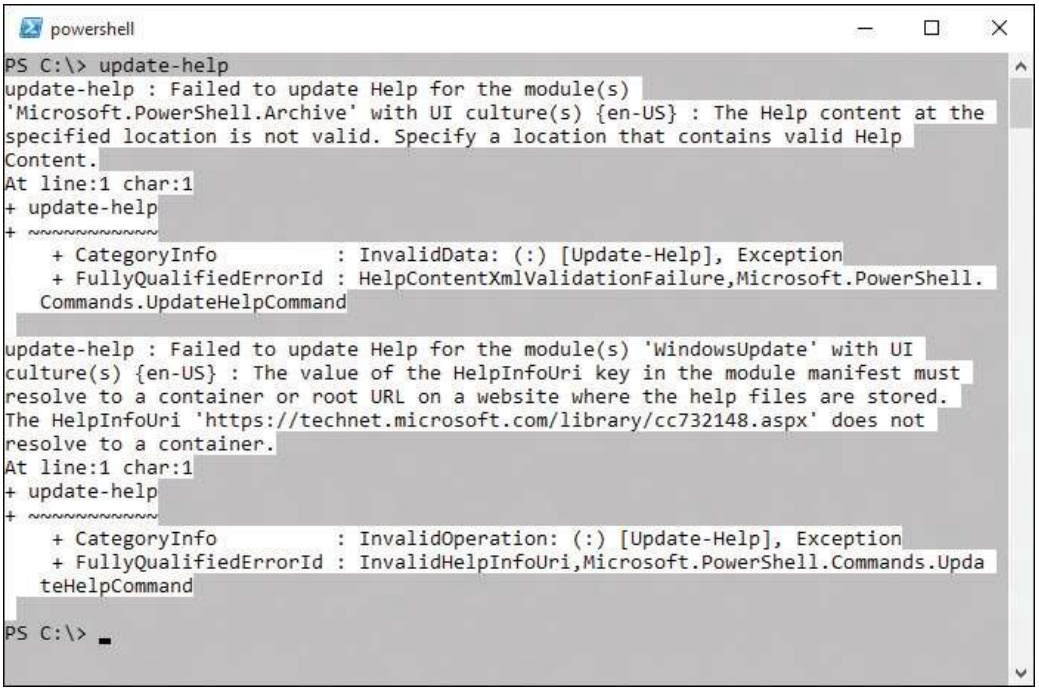
Jednym z pierwszych poleceń, które należy wywołać przy rozpoczynaniu pracy z Windows PowerShell, jest *Update-Help*. Wynika to z faktu, że Windows PowerShell nie jest domyślnie wyposażony w pliki pomocy dla wszystkich poleceń cmdlet. Nie oznacza to, że taka pomoc nie istnieje – jedynie to, że pomoc wykraczająca poza podstawową składnię wymaga pobrania dodatkowych materiałów.

Domyślna instalacja Windows PowerShell 5.0 zawiera liczne moduły, które zmieniają się pomiędzy różnymi wdrożeniami zależnie od wersji systemu operacyjnego, zainstalowanych w nim funkcji i aktywnych ról. W istocie Windows PowerShell 5.0 zainstalowany w systemie Windows 7 zawiera mniej modułów i poleceń, niż dostępne w stacji roboczej Windows 10. Nie oznacza to jednak chaosu, gdyż najważniejsze polecenia cmdlet – *core* – nie zmieniają się pomiędzy poszczególnymi instalacjami. Różnice pomiędzy instalacjami wynikają z faktu, że dodatkowe funkcje i role systemu operacyjnego często powodują doinstalowanie nowych modułów Windows PowerShell i związanych z nimi poleceń cmdlet.

Modułowa budowa Windows PowerShell wymaga dodatkowej uwagi podczas aktualizowania plików pomocy. Proste wywołanie *Update-Help* nie uaktualni wszystkich modułów dostępnych w określonym systemie. W rzeczywistości niektóre moduły mogą w ogóle nie obsługiwać aktualizacji pomocy – w tym wypadku próba uaktualnienia wygeneruje błąd. Najprostszą metodą upewnienia się, że aktualizacja obejmie wszystkie możliwe elementy, jest użycie parametru *-Module* oraz przełącznika *-Force*. Polecenie powodujące uaktualnienie pomocy dla wszystkich zainstalowanych modułów (tych, które wspierają aktualizację pomocy) będzie wyglądało następująco:

```
Update-Help -Module * -Force
```

Rezultat uruchomienia polecenia Update-Help w typowym systemie klienckim Windows 10 przedstawia rysunek 1-1.



Rysunek 1-1 Błędy wyświetlane po próbie aktualizacji pomocy w modułach, które nie obsługują aktualizacji.

Sposobem na uniknięcie wyświetlania ekranu pełnego błędów jest wywołanie polecenia *Update-Help* z przełącznikiem blokującym wyświetlanie wszystkich komunikatów o błędach:

```
Update-Help -Module * -Force -ea 0
```

Podejście takie powoduje jednak, że nigdy nie będziemy mieć pewności, czy rzeczywiście uzyskaliśmy zaktualizowaną treść pomocy dla wszystkich elementów, dla których jest to możliwe. Lepszym podejściem byłoby ukrycie błędów pojawiających się w trakcie procesu aktualizacji, ale wyświetlenie tych, które występują po jego zakończeniu. Dodatkową korzyścią takiego podejścia jest wyświetlenie błędów procesu czyszczącego (cleaner). Technikę tę ilustruje skrypt UpdateHelpTrackErrors.ps1. Pierwszą rzeczą, jaką robi ten skrypt, jest wyczyszczenie stosu błędów poprzez wywołanie metody *clear*. Następnie skrypt wywołuje polecenie cmdlet *Update-Help* z parametrem *-Module* i przełącznikiem *-Force*. Dodatkowo użyty jest przełącznik *-ErrorAction* (*ea* widoczne w kodzie skryptu to alias) z wartością 0 (zero). Powoduje to, że błędy nie są wyświetlane podczas przebiegu polecenia. Skrypt kończy pętla *For*, przechodząca przez zgromadzone komunikaty błędów i wyświetlająca zarejestrowane wyjątki. Cały skrypt został przedstawiony poniżej.

UpdateHelpTrackErrors.ps1

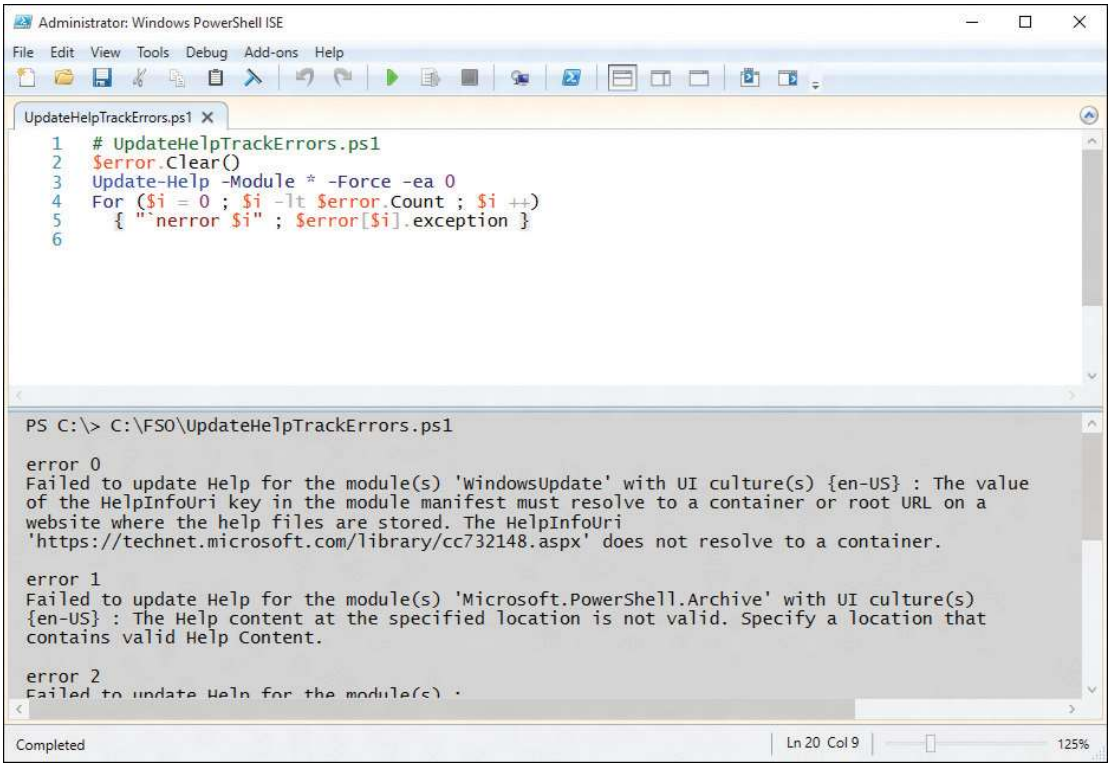
```
$error.Clear()
Update-Help -Module * -Force -ea 0
For ($i = 0 ; $i -lt $error.Count ; $i ++)
  { "'nerror $i" ; $error[$i].exception }
```

Uwaga Informacje dotyczące pisania skryptów Windows PowerShell i wykorzystywania pętli *For* znaleźć można w rozdziale 5.

W trakcie wykonywania skryptu wyświetlany jest pasek postępu. Po jego ukończeniu wyświetlane są wszystkie napotkane błędy. Skrypt i powiązane z nim komunikaty błędów są widoczne na rysunku 1-2.

Mogłoby się wydawać, że ustalenie, które moduły otrzymały zaktualizowane pliki pomocy, umożliwia również wykorzystanie przełącznika *-Verbose* dla polecenia *Update-Help*. Jednak dane wyjściowe powodują tak szybkie przewijanie ekranu, że trudno jest cokolwiek na nim zobaczyć. Aby rozwiązać ten problem, można przekierować wyjście polecenia do pliku tekstowego. W przykładzie poniżej wszystkie moduły próbują zaktualizować swoje pliki pomocy. Pełne komunikaty zwrotne są kierowane do pliku *updatedhelp.txt* zlokalizowanego w folderze fso na dysku C.

```
Update-Help -module * -force -verbose 4>>c:\fso\updatedhelp.txt
```



Rysunek 1-2 Wyczyszczone wyjście błędów dla procesu aktualizacji pomocy

Windows PowerShell zapewnia wysoki poziom „odkrywalności" (ang. *discoverability*); innymi słowy, aby nauczyć się korzystać z PowerShell, należy po prostu zacząć go używać. Ważną rolę w tym praktycznym poznawaniu pełnią serwery pomocy online. System pomocy w Windows PowerShell jest dostępny kilkoma metodami.

Jedną z nich jest wspomniane wcześniej polecenie cmdlet *Get-Help*. Aby uzyskać informacje o stosowaniu tego polecenia, można je wywołać podając je jako parametr:

```
Get-Help Get-Help
This command prints out help about the Get-Help cmdlet. The output from this cmdlet
is illustrated here:
```

Spowoduje to wyświetlenie informacji o składni polecenia, podobnych do pokazanych poniżej [1]:

```
NAZWA
    Get-Help

PODSUMOWANIE
    Wyświetla informacje o poleceniach i koncepcjach Windows PowerShell.
```

1 Pliki pomocy Windows PowerShell są zazwyczaj dostępne tylko w języku angielskim i taki rezultat uzyska Czytelnik na swoim komputerze. W tym miejscu zdecydowaliśmy się umieścić tłumaczenie informacji zwracanych przez polecenie, ale należy pamiętać, że – przynajmniej w czasie pracy nad tą książką – polska wersja informacji pomocy nie była dostępna (wszystkie przypisy pochodzą od tłumacza).

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

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

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

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

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

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

OPIS
    Polecenie cmdlet Get-Help informacje o poleceniach i koncepcjach Windows
PowerShell, w tym o poleceniach cmdlet, funkcjach, poleceniach CIM, przepływach,
dostawcach, aliasach i skryptach.

    Aby uzyskać pomoc na temat polecenia Windows PowerShell, wpisz "Get-Help"
uzupełnione o nazwę polecenia, tak jak: Get-Help Get-Process. Aby uzyskać listę
wszystkich tematów pomocy zainstalowanych w systemie, wpisz: Get-Help *. Można
wyświetlić cały temat pomocy lub użyć parametrów cmdlet Get-Help do uzyskania
wybranych fragmentów, takich jak opis składni, parametrów lub przykłady zastosowania.

    Koncepcyjne tematy pomocy w Windows PowerShell zaczynają się od prefiksu
"about_", na przykład "about_Comparison_Operators". Aby wyświetlić wszystkie tematy
"about_", wpisz: Get-Help about_*. Aby wyświetlić konkretny temat, wpisz: Get-Help
about_<topic-name>, na przykład Get-Help about_Comparison_Operators.

    Aby uzyskać pomoc dla dostawcy Windows PowerShell, należy wpisać "Get-Help",
a następnie nazwę dostawcy. Na przykład, do uzyskiwania pomocy dotyczącej dostawcy
Certificate służy polecenie Get-Help Certificate.

    Oprócz stosowania "Get-Help" można również wpisać "help" lub "man",
co spowowoduje wyświetlanie informacji po jednym ekranie na raz, albo "<cmdlet-name>
-?", co daje efekt identyczny jak Get-Help, ale działa tylko dla poleceń.

    Get-Help pobiera zawartość pomocy z plików pomocy (help) zainstalowanych
na komputerze. Bez plików pomocy Get-Help wyświetla tylko podstawowe informacje
o poleceniach. Niektóre moduły Windows PowerShell są wyposażone w pliki pomocy.
Jednak począwszy od wersji Windows PowerShell 3.0 moduły dołączone do systemu Windows
nie zawierają plików pomocy. Aby pobrać lub uaktualnić pliki pomocy dla modułu
Windows PowerShell 3.0, należy posłużyć się cmdlet Update-Help.
```

```
    Zawartość tematów pomocy jest również dostępna w trybie online w TechNet Library.
Aby uzyskać wersję online tematu pomocy, należy użyć parametru Online, na przykład:
Get-Help Get-Process -Online. Wszystkie tematy pomocy znajdują się w witrynie
dostępnej pod adresem: http://go.microsoft.com/fwlink/?LinkID=107116.

    Jeśli wpiszesz "Get-Help" uzupełnione o dokładną nazwę tematu pomocy lub słowo
unikatowe dla tego tematu, zostanie wyświetlona treść tego tematu. Jeśli wpiszesz
słowo lub wzorzec występujący w wielu tematach, Get-Help wyświetli listę pasujących
tytułów. Jeśli użyjesz słowa, które nie występuje w żadnym tytule tematu pomocy, Get-
Help wyświetli listę tematów, które zawierają to słowo w swojej treści.

    Get-Help może odczytywać tematy pomocy dla wszystkich obsługiwanych języków. Get-
Help najpierw szuka plików pomocy dla ustawień językowych zdefiniowanych w systemie
Windows, następnie dla lokalizacji nadrzędnej, tak jak "pt" (portugalski) dla "pt-BR"
(portugalski brazylijski), a następnie dla języka zastępczego. Począwszy od wersji
Windows PowerShell 3.0, jeśli Get-Help nie znajdzie pomocy w żadnym z tych języków,
wyszuka temat pomocy w języku angielskim ("en-US"), zanim zwróci komunikat o błędzie
lub wyświetli automatycznie wygenerowaną podpowiedź.

Informacje o symbolach używanych przez Get-Help w diagramach składniowych poleceń
udostępnia temat about_Command_Syntax. Informacje o atrybutach parametrów, takich jak
Required lub Position, zawiera temat about_Parameters.

    O ROZWIĄZYWANIU PROMLEMÓW: W wersjach Windows PowerShell 3.0 i 4.0 polecenie
Get-Help nie znajduje tematów about w modułach, które nie zostały zaimportowane
do bieżącej sesji. Jest to powszechnie znany problem. Aby zobaczyć temat about
w module, należy zaimportować moduł, używając polecenia cmdlet Import-Module lub
uruchamiając dowolne polecenie cmdlet z modułu.

ŁĄCZA POWIĄZANE
    Online Version: http://go.microsoft.com/fwlink/p/?linkid=289584
    Updatable Help Status Table (http://go.microsoft.com/fwlink/?LinkID=270007)
    Get-Command
    Get-Member
    Get-PSDrive
    about_Command_Syntax
    about_Comment_Based_Help
    about_Parameters

UWAGI
    Aby zobaczyć przykłady, wpisz: "get-help Get-Help -examples".
    Aby uzyskać więcej informacji, wpisz: "get-help Get-Help -detailed".
    Aby uzyskać informacje techniczne, wpisz: "get-help Get-Help -full".
    Aby uzyskać pomoc online, wpisz: "get-help Get-Help -online"
```

Cenną zaletą systemu pomocy Windows PowerShell jest to, że nie tylko wyświetla informacje o składni poleceń, ale udostępnia trzy poziomy wyświetlania: normalny, szczegółowy i pełny. Dodatkowo można uzyskać informacje o koncepcjach działania Windows PowerShell. Funkcja ta jest odpowiednikiem podręcznika instruktażowego online. Aby uzyskać listę wszystkich artykułów koncepcyjnych, należy użyć polecenia *Get-Help about**, zwracającego listę wszystkich tematów z prefiksem *about*.

```
Get-Help about*
```

Wyobraźmy sobie, że nie pamiętamy dokładnej nazwy polecenia cmdlet, którego chcielibyśmy użyć, ale pamiętamy, że to było cmdlet *get*. W celu odszukania tej nazwy możemy posłużyć się znakiem zastępczym, takim jak gwiazdka (*), jak poniżej:

```
Get-Help get*
```

Technika znaków zastępczych może być dalej rozszerzana. Jeśli pamiętamy, że chodziło o polecenie *get* i że pierwszą literą rzeczownika w nazwie było p, możemy użyć następującej składni, aby uzyskać listę pasujących poleceń:

```
Get-Help get-p*
```

Po ustaleniu, o które polecenie chodziło, może pojawić się problem, że nie mamy pewności co do prawidłowej składni. W takim przypadku możemy sięgnąć po parametr przełącznika *-Examples*, który zademonstruje składnię polecenia na przykładach. Poniższe polecenie *Get-Help* spowoduje wyświetlenie przykładów zastosowania (z komentarzami) dla polecenia cmdlet *Get-PSDrive*:

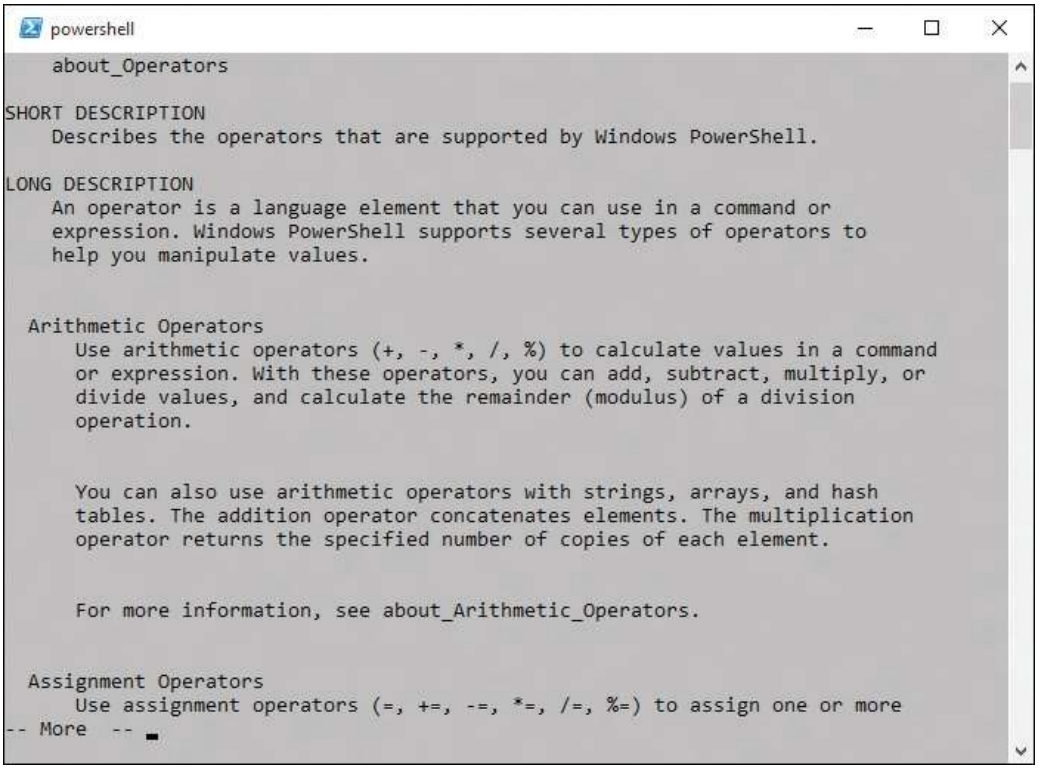
```
Get-Help Get-PSDrive -examples
```

Czytelnik łatwo zauważy, że uzyskane dane znacznie przekraczają pojemność jednego ekranu. Zawartość pomocy można wyświetlać po jednym ekranie na raz, używając funkcji Help. Funkcja ta przesyła wprowadzone przez użytkownika dane do polecenia *Get-Help*, a odbierane informacje zwrotne przekierowuje do narzędzia *more.com*. Powoduje to wyświetlenie jednego ekranu informacji oraz znaku zachęty *More* (Dalej). Naciśnięcie klawisza Enter wyświetla wówczas kolejne wiersze (po jednym), zaś klawiszem spacji można przewinąć cały kolejny ekran.

Uwaga Trzeba pamiętać, że przy korzystaniu z Windows PowerShell ISE funkcja stronicowania nie będzie działać, zatem nie zauważymy żadnej różnicy pomiędzy *Get-Help* i *Help*. W tym środowisku obydwa polecenia zachowują się tak samo. Jednak należy się spodziewać, że przy pracy w Windows PowerShell ISE użytkownik będzie korzystał raczej z polecenia *Show-Command*, a nie *Get-Help*, aby uzyskać informacje o składni i zastosowaniu.

Rysunek 1-3 prezentuje przykład sformatowanego wyjścia dla tematu pomocy przekraczającego objętość jednego ekranu, wywołanego poprzez funkcję Help.

Ciągłe wpisywanie *Get-Help* może okazać się męczące – ostatecznie, to aż osiem znaków! Rozwiązaniem jest utworzenie aliasu dla polecenia cmdlet. Alias to skrócona nazwa (kombinacja znaków), która wywoła odpowiedni program lub polecenie. Przedstawiona poniżej procedura „Tworzenie aliasu dla cmdlet *Get-Help*" przypisze to polecenie do kombinacji klawiszy GH.



Rysunek 1-3 Wykorzystanie funkcji *Help* do wyświetlania informacji po jednej stronie na raz

---

Uwaga Przed utworzeniem nowego aliasu warto sprawdzić, czy polecenie cmdlet nie ma już go zdefiniowanego, używając polecenia *Get-Alias*. Następnie można użyć *New-Alias*, aby przypisać poleceniu unikatową kombinację klawiszy.

➜ Tworzenie aliasu dla cmdlet *Get-Help*

1. Otwórz konsolę Windows PowerShell, wybierając Start | Run | Windows PowerShell.
2. Wywołaj alfabetycznie posortowaną listę aktualnie zdefiniowanych aliasów i sprawdź, czy istnieje alias dla polecenia *Get-Help* oraz czy kombinacja G+H jest aktualnie wolna. Listę tę można uzyskać poniższym poleceniem:

   ```
   Get-Alias | sort
   ```

3. Po upewnieniu się, że nie istnieje jeszcze alias dla polecenia *Get-Help* i że żadne inne nie ma przypisanego skrótu GH, sprawdź składnię dla polecenia *New-Alias*. Użyj przełącznika *-Full*, aby wyświetlić pełny tekst pomocy:

   ```
   Get-Help New-Alias -full
   ```

4. Użyj polecenia *New-Alias*, aby przypisać skrót G+H do polecenia cmdlet *Get-Help*.

   ```
   New-Alias gh Get-Help
   ```

Poznawanie poleceń: ćwiczenia krok po kroku

W kolejnych ćwiczeniach poznamy posługiwanie się narzędziami wiersza poleceń w Windows PowerShell. Przekonamy się, że korzystanie z tych narzędzi jest równie proste jak w interpreterze CMD; jednak dzięki stosowaniu tych narzędzi w środowisku Windows PowerShell uzyskujemy dostęp do nowych poziomów funkcjonalności.

➜ Korzystanie z narzędzi wiersza poleceń

1. Otwórz konsolę Windows PowerShell, jeśli nie jest jeszcze uruchomiona.
2. Zmień bieżący katalog na główny katalog dysku C, wpisując **cd c:**.

```
cd c:\
```

3. Wyświetl listę plików w katalogu głównym, używając polecenia *dir*:

```
dir
```

4. Utwórz nowy katalog w głównym katalogu dysku C, używając polecenia *md*:

```
md mytest
```

5. Uzyskaj listę wszystkich plików i katalogów zaczynających się od litery *m*.

```
dir m*
```

6. Zmień katalog bieżący na katalog roboczy Windows PowerShell. Do tego celu użyj polecenia cmdlet *Set-Location* i zmiennej *$pshome*:

```
Set-Location $pshome
```

7. Uzyskaj listę liczników pamięci dotyczących dostępnej wielkości, używając pokazanego poniżej polecenia *typeperf.exe*[2].

```
typeperf "\memory\available bytes"
```

8. Po wyświetleniu kilku kolejnych odczytów licznika w oknie Windows PowerShell przerwij działanie polecenia, naciskając Ctrl+C.
9. Wyświetl bieżącą konfigurację startową za pomocą polecenia *bcdedit* (zwróć uwagę, że to polecenie wymaga uprawnień administracyjnych):

```
bcdedit
```

10. Zmień bieżący katalog na C:\Mytest utworzony wcześniej:

```
Set-Location c:\mytest
```

2 W lokalizowanej (nie-angielskiej) wersji systemu Windows należy użyć zlokalizowanej wersji nazw liczników. W przypadku wersji polskiej odpowiedni licznik nosi nazwę „\pamięć\dostępne bajty” (wielkość liter nie ma znaczenia).

11. Utwórz plik o nazwie *mytestfile.txt* w katalogu C:\Mytest. Użyj narzędzia *fsutil*, aby utworzyć plik o wielkości 1000 bajtów. W tym celu posłuż się następującym poleceniem:

    ```
    fsutil file createnew mytestfile.txt 1000
    ```

12. Wyświetl listę wszystkich plików w katalogu Mytest przy użyciu polecenia cmdlet *Get-ChildItem*:
13. Wyświetl aktualną datę za pomocą polecenia *Get-Date*.
14. Wyczyść ekran przy użyciu polecenia *cls*.
15. Wyświetl listę wszystkich poleceńcmdlet wbudowanych w Windows PowerShell. W tym celu posłuż się poleceniem *Get-Command*.
16. Użyj polecenia *Get-Command* do pobrania polecenia *Get-Alias*. W tym celu posłuż się parametrem *-Name*, podając nazwę polecenia Get-Alias jako wartość parametru:

    ```
    Get-Command -name Get-Alias
    ```

    Na tym kończy się to ćwiczenie. Zamknij konsolę Windows PowerShell, wpisując **exit** i naciskając klawisz Enter.

W kolejnym ćwiczeniu użyjemy różnych opcji pomocy do uzyskania informacji o różnych poleceniach cmdlet.

➜ Uzyskiwanie pomocy

1. Uruchom konsolę Windows PowerShell, wybierając Start | Run | Windows PowerShell. Konsola uruchomi się domyślnie w katalogu głównym bieżącego użytkownika.
2. Użyj cmdlet *Get-Help* do uzyskania informacji o poleceniu *Get-Help* w następujący sposób:

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

3. Wyświetl szczegółowe informacje pomocy o poleceniu *Get-Help*, używając przełącznika *-Detailed*:

    ```
    Get-Help Get-Help -detailed
    ```

4. Odczytaj pogłębione informacje techniczne o poleceniu *Get-Help*, używając przełącznika *-Full*:

    ```
    Get-Help Get-Help -full
    ```

5. Aby uzyskać tylko listę przykładów użycia polecenia, użyj *przełącznika -Examples*:

    ```
    Get-Help Get-Help -examples
    ```

6. Wyświetl listę informacyjnych tematów pomocy, podając prefiks *about* z symbolem wieloznacznym * (gwiazdka) jako parametr:

```
Get-Help about*
```

7. Wyświetl listę tematów pomocy dotyczących poleceń *get*. W tym celu użyj jako parametru słowa *get* z symbolem wieloznacznym:

```
Get-Help get*
```

8. Wyświetl listę tematów pomocy dla poleceń *set*:

```
Get-Help set*
```

Na tym kończy się to ćwiczenie. Zamknij konsolę Windows PowerShell, wpisując **exit** i naciskając klawisz Enter.

Podsumowanie rozdziału 1

Aby	Zrób to
Wywołać zewnętrzne polecenie lub narzędzie trybu wiersza poleceń	Wpisz nazwę polecenia (bez rozszerzenia) po znaku zachęty Windows PowerShell.
Wywołać sekwencyjnie wiele zewnętrznych poleceń lub narzędzi	Wpisz kolejne polecenia w jednym wierszu Windows PowerShell, rozdzielając je średnikami.
Uzyskać listę uruchomionych procesów	Użyj polecenia cmdlet *Get-Process*.
Zatrzymać proces	Użyj polecenia cmdlet *Stop-Process*, stosując nazwę lub identyfikator procesu w roli parametru.
Zasymulować efekt polecenia cmdlet przed rzeczywistym wykonaniem żądanej akcji	Użyj przełącznika *-Whatlf*.
Spowodować uruchomienie Windows PowerShell, wykonanie określonego polecenia cmdlet i zamknięcie Windows PowerShell	Użyj polecenia *PowerShell*, podając jako parametr uruchomieniowy nazwę cmdlet ujętą w nawiasy klamrowe i poprzedzoną symbolem &.
Spowodować wyświetlanie monitu o potwierdzenie zatrzymania procesu	Użyj polecenia cmdlet *Stop-Process* z parametrem *-Confirm*.

ROZDZIAŁ 2

Korzystanie z poleceń cmdlet

Po zapoznaniu się z tym rozdziałem Czytelnik będzie umiał:

- Używać poleceń cmdlet Windows PowerShell w podstawowym zakresie
- Wykorzystać Get-Command do uzyskania listy poleceń cmdlet
- Konfigurować wyświetlanie wyników
- Konfigurować opcje wyszukiwania poleceń cmdlet
- Korzystać z polecenia Get-Member
- Korzystać z polecenia New-Object
- Używać polecenia Show-Command

Umieszczenie ogromnej liczby poleceń cmdlet w Windows PowerShell sprawia, że powłoka ta jest natychmiast użyteczna dla administratorów sieci i innych osób, które muszą wykonywać najrozmaitsze zadania konserwacyjne i administracyjne w systemach biurkowych i serwerowych. W tym rozdziale poznamy wiele użytecznych poleceń cmdlet, dobranych z myślą o zaprezentowaniu siły i elastyczności Windows PowerShell. Jednak prawdziwą korzyścią, jaką Czytelnik może wynieść z tego rozdziału, jest metodologia, której będziemy używać do odkrywania zastosowań różnych poleceń

Podstawy poleceń cmdlet

W rozdziale 1, „Przegląd cech Windows PowerShell 5.0”, pokazaliśmy wykorzystanie różnych dostępnych narzędzi pomocy, aby zademonstrować korzystanie z poleceń cmdlet. Przedstawiliśmy kilka poleceń, które są przydatne przy ustalaniu, jakie polecenia są dostępne i jak należy ich używać. W tym podrozdziale pokażemy kilka innych sposobów posługiwania się poleceniami cmdlet w Windows PowerShell.

Wskazówka Wpisywanie długich nazw cmdlet jest często nużące, a ponadto podatne na błędy literowe. Aby uprościć ten proces, wystarczy wpisać fragment nazwy polecenia dostateczny do jego jednoznacznego wskazania, po czym nacisnąć klawisz Tab. Co otrzymamy? Samoczynne dopełnienie nazwy (funkcja ta nazywana jest *tab completion* – uzupełnianie tabulatorem). Działa to również dla nazw parametrów i innych wpisywanych elementów, takich jak obiekty .NET, katalogi, klucze rejestru i tak dalej. Warto poeksperymentować z tą techniką. Być może już nigdy więcej nie będzie potrzeby wpisywania *Get-Command*! Jeśli wyszukiwane polecenie cmdlet nie zostało odnalezione, wystarczy kontynuować naciskanie klawisza Tab, powodując pojawianie się kolejnych dopasowań.

Ponieważ polecenia cmdlet zwracają obiekty, a nie wartości łańcuchów, można otrzymać dodatkowe informacje o uzyskiwanych obiektach. Te informacje nie byłyby dostępne, gdybyśmy posługiwali się tylko danymi tekstowymi. Aby pobrać informacje z jednego polecenia cmdlet i przekazać je do innego, można użyć znaku potoku (|, ang. pipe). Może się to wydawać skomplikowane, ale w rzeczywistości jest to bardzo proste i po ukończeniu tego rozdziału będzie całkowicie naturalne. Na najbardziej podstawowym poziomie rozważmy odczytywanie zawartości katalogu; po uzyskaniu listingu plików zapewne chcielibyśmy określić sposób jego wyświetlenia (sformatowania) – na przykład jako tabelę lub listę. Jak można zauważyć, odczytanie informacji o zawartości katalogu i sformatowanie listy to dwie oddzielne operacje. Druga z nich znajdzie się po prawej stronie symbolu potoku.

Korzystanie z cmdlet *Get-ChildItem*

W rozdziale 1 posłużyliśmy się poleceniem *dir* do wyświetlenia listy wszystkich folderów i plików w danym katalogu. Polecenie to działa, gdyż istnieje alias wbudowany w Windows PowerShell, który przypisuje cmdlet *Get-ChildItem* do sekwencji liter *dir*.

Uzyskiwanie listingu katalogu

Listę zawartości katalogu dyskowego uzyskamy wpisując w Windows PowerShell polecenie *Get-ChildItem* uzupełnione o nazwę katalogu (lub puste, aby uzyskać listing bieżącego katalogu). Przypomnijmy, że można użyć uzupełniania tabulatorem. Wystarczy wpisać **get-ch** i nacisnąć Tab, aby dopełnić nazwę. Oto przykład polecenia:

```
Get-ChildItem C:\
```

Uwaga Windows PowerShell nie rozróżnia wielkości liter, zatem *get-Childitem*, *Get-childditem* i *Get-ChildItem* zostaną zinterpretowane tak samo – dla Windows PowerShell jest to to samo polecenie.

W Windows PowerShell nie istnieje w rzeczywistości cmdlet o nazwie *dir*; powłoka nie odwołuje się też do polecenia *dir* z czasów DOS. Zamiast tego istnieje wbudowany alias *dir* dla polecenia cmdlet *Get-ChildItem*. Dlatego też wynik polecenia *dir* w Windows PowerShell różni się od wyglądu wyniku polecenia *dir* w interpreterze CMD.exe. Przy użyciu polecenia cmdlet *Get-Alias* można sprawdzić, jak wygląda powiązanie pomiędzy *dir* a *Get-ChildItem*:

```
PS C:\> Get-Alias dir

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Alias           dir -> Get-ChildItem
```

Tak więc zarówno użycie *Get-ChildItem*, jak i *dir* zwraca dokładnie tak samo wyglądającą listę, co widać na poniższym przykładzie:

```
PS C:\> dir c:\

    Directory: C:\

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        7/11/2015  11:55 AM                FSO
d-----         7/9/2015   5:24 AM                PerfLogs
d-r---         7/9/2015   6:59 AM                Program Files
d-r---        7/10/2015   7:27 PM                Program Files (x86)
d-r---        7/10/2015   7:18 PM                Users
d-----        7/10/2015   6:00 PM                Windows

PS C:\> Get-ChildItem c:\

    Directory: C:\

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        7/11/2015  11:55 AM                FSO
d-----         7/9/2015   5:24 AM                PerfLogs
d-r---         7/9/2015   6:59 AM                Program Files
d-r---        7/10/2015   7:27 PM                Program Files (x86)
d-r---        7/10/2015   7:18 PM                Users
d-----        7/10/2015   6:00 PM                Windows

PS C:\>
```

Analogicznie, jeśli użyjemy polecenia *Get-Help*, aby uzyskać informacje o *dir*, otrzymamy ten sam opis, co dla polecenia *Get-Help Get-ChildItem*. Widać to w poniższym przykładzie (skróconym tylko do nazwy i podsumowania).

```
PS C:\> Get-Help dir | select name, synopsis | Format-Table -AutoSize

Name          Synopsis
----          --------
Get-ChildItem Gets the files and folders in a file system drive.
```

```
PS C:\> Get-Help Get-ChildItem | select name, synopsis | Format-Table -AutoSize

Name          Synopsis
----          --------
Get-ChildItem Gets the files and folders in a file system drive.

PS C:\>
```

W Windows PowerShell alias i pełna nazwa cmdlet działają w taki sam sposób. Aliasy nie umożliwiają zmodyfikowania zachowania polecenia cmdlet – do tego celu trzeba użyć własnej, zdefiniowanej przez siebie funkcji lub funkcji proxy.

Formatowanie listingu katalogu przy użyciu polecenia *Format-List*

Jak wspomniano wcześniej, wynik jednego polecenia cmdlet można przekierować do innego za pomocą znaku potoku. Poniższy przykład tworzy listing katalogu i kieruje go do polecenia cmdlet *Format-List*:

```
Get-ChildItem C:\ | Format-List
```

➜ Formatowanie wyjścia przy użyciu polecenia cmdlet *Format-List*

1. Otwórz konsolę Windows PowerShell.
2. Użyj polecenia cmdlet *Get-ChildItem* do uzyskania listingu katalogu głównego dysku C.

   ```
   Get-ChildItem C:\
   ```

3. Użyj cmdlet *Format-List*, aby zmienić uporządkowanie listingu katalogu:

   ```
   Get-ChildItem C:\ | Format-List
   ```

4. Użyj parametru *-Property* polecenia *Format-List*, aby wyświetlić tylko nazwy plików i katalogów:

   ```
   Get-ChildItem C:\ | Format-List -property name
   ```

5. Zmodyfikuj polecenie, aby dodać wielkości plików z katalogu głównego. Niepojawienie się właściwości *length* oznacza, że katalog główny nie zawiera widocznych (nieukrytych) plików, ponieważ foldery nie posiadają właściwości *length*.

   ```
   Get-ChildItem C:\ | Format-List -property name, length
   ```

Korzystanie z polecenia cmdlet *Format-Wide*

W taki sam sposób, w jaki wykorzystaliśmy *Format-List* do przekształcenia wyjścia polecenia na listę, można użyć polecenia *Format-Wide* do uzyskania bardziej zwartego wyniku. Różnica polega na tym, że *Format-Wide* umożliwia wybranie tylko jednej właściwości dla wyświetlanych elementów. Możemy jednak określić, w jak wielu kolumnach będą wyświetlane informacje. Domyślnie *Format-Wide* używa dwóch kolumn.

➜ Formatowanie listingu katalogu przy użyciu *Format-Wide*

1. W oknie Windows PowerShell wywołaj cmdlet *Get-ChildItem* z nazwą katalogu, uzupełniając polecenie znakiem potoku i poleceniem *Format-Wide*, jak poniżej:

   ```
   Get-ChildItem C:\ | Format-Wide
   ```

2. Zmieńwyświetlanie na trzykolumnowe i jawnie wskaż wyświetlanie właściwości *name* (nazwa):

   ```
   Get-ChildItem | Format-Wide -Column 3 -Property name
   ```

3. Pozwól Windows PowerShell na wyświetlenie tak wielu kolumn, jak to możliwe dla bieżącej szerokości okna i długości nazw. Służy do tego przełącznik *-AutoSize*:

   ```
   Get-ChildItem | Format-Wide -Property name –AutoSize
   ```

4. Wymuś obcinanie nazw w kolumnach, wybierając liczbę kolumn większą niż wyświetlona w poprzednim przykładzie (zbyt długie nazwy zostaną obcięte, co sygnalizowane jest wielokropkiem na końcu):

   ```
   Get-ChildItem | Format-Wide -Property name -Column 8
   ```

➜ Formatowanie wyjścia polecenia przy użyciu *Format-Wide*

1. Uruchom konsolę Windows PowerShell, jeśli nie jest jeszcze otwarta.
2. Użyj polecenia *Get-ChildItem* do wyświetlenia listingu katalogu C:\Windows:

   ```
   Get-ChildItem C:\Windows
   ```

3. Użyj parametru *-Recurse*, aby nakazać poleceniu *Get-ChildItem* przejście przez całą zagnieżdżoną strukturę katalogu, uwzględniając tylko pliki .txt. Ukryj błędy, używając parametru *-ea* (*będącego aliasem dla ErrorAction*) i przypisując mu wartość 0. Oznacza to, że wszystkie błędy zostaną zignorowane([*SilentlyContinue*]):

   ```
   Get-ChildItem C:\Windows -recurse -include *.txt –ea 0
   ```

   Fragment wyniku jest przedstawiony poniżej:

   ```
   PS C:\> Get-ChildItem C:\Windows -recurse -include *.txt -ea 0

       Directory: C:\Windows\InfusedApps\Packages\
   Microsoft.3DBuilder_10.0.0.0_x64__8wekyb3d8bbwe\Common
   ```