

Windows PowerShell



NAJLEPSZE PRAKTYKI

Tytuł oryginału: Windows PowerShell Best Practices
Tłumaczenie: Łukasz Piwko
ISBN: 978-83-283-0478-9

Authorized translation from the English language edition: WINDOWS POWERSHELL
BEST PRACTICES; ISBN 0735666490; by Ed Wilson; published by Microsoft Press, a division
of Microsoft Corporation, Inc.
Copyright © 2013 by Ed Wilson.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means,
electronic or mechanical, including photocopying, recording or by any information storage retrieval system,
without permission from Pearson Education, Inc., or Microsoft Press.

Polish language edition published by HELION S.A., under license and with the permission of Pearson
Education, Inc. Copyright © 2015.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej
publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną,
fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje
naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich
właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były
kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane
z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie
ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji
zawartych w książce.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
<ftp://ftp.helion.pl/przyklady/winpsp.zip>

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
<http://helion.pl/user/opinie/winpsp>
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

Przedmowa	15
Wprowadzenie	17

Część I Podstawowe wiadomości o konsoli Windows PowerShell21

Rozdział 1. Przegląd możliwości konsoli Windows PowerShell	23
Czym jest konsola Windows PowerShell	23
Instalowanie konsoli Windows PowerShell	25
Wdrażanie konsoli Windows PowerShell	26
Polecenia cmdlet	27
Używanie narzędzi wiersza poleceń	28
Kwestie bezpieczeństwa dotyczące konsoli Windows PowerShell	29
Kontrolowanie wykonywania poleceń cmdlet Windows PowerShell	30
Zatwierdzanie poleceń	30
Zawieszanie potwierdzenia wykonania poleceń cmdlet	31
Praca z konsolą Windows PowerShell	31
Włączanie konsoli Windows PowerShell	32
Konfigurowanie konsoli Windows PowerShell	33
Przekazywanie opcji do poleceń cmdlet	34
Korzystanie z opcji pomocy	35
Zdobywanie informacji w pomocy	38
Dodatkowe źródła informacji	43
Rozdział 2. Polecenia CIM	45
Przeglądanie klas usługi WMI za pomocą poleceń CIM	45
Sposób użycia parametru classname	45
Znajdowanie metod klas WMI	47
Filtrowanie klas według kwalifikatora	48

Wyszukiwanie egzemplarzy klas WMI	49
Zmniejszanie liczby zwróconych własności i egzemplarzy	50
Usuwanie niepotrzebnych informacji	51
Praca z klasami Association	52
Dodatkowe źródła informacji	57

Część II Planowanie skryptów 59

Rozdział 3. Moduł Active Directory 61

Podstawowe wiadomości o module Active Directory	61
Instalowanie modułu Active Directory	62
Rozpoczynanie pracy z modułem Active Directory	63
Zastosowanie modułu Active Directory	63
Wyszukiwanie posiadaczy roli FSMO	65
Dokumentowanie Active Directory	70
Zmianie nazw lokalizacji usługi Active Directory	73
Zarządzanie użytkownikami	74
Tworzenie użytkownika	77
Znajdowanie kont użytkowników i ich odblokowywanie	78
Znajdowanie wyłączonych użytkowników	80
Znajdowanie nieużywanych kont użytkowników	81
Dodatkowe źródła informacji	85

Rozdział 4. Znajdowanie możliwości zastosowania skryptów 87

Automatyzowanie rutynowych zadań	87
Interfejs automatyzacji	88
Odczytywanie rejestru za pomocą metody RegRead	91
Odczytywanie rejestru za pomocą WMI	91
Odczytywanie rejestru za pomocą klas platformy .NET	92
Macierzyste techniki Windows PowerShell	
do wykonywania niektórych zadań	93
Wymagania strukturalne	96
Wymagania dotyczące bezpieczeństwa	96
Wykrywanie bieżącego użytkownika	97
Wykrywanie roli użytkownika	107
Wymagania dotyczące wersji platformy .NET	111
Wymagania dotyczące systemu operacyjnego	113
Wymagania dotyczące aplikacji	117
Wymagania dotyczące modułów	118
Dodatkowe źródła informacji	119

Rozdział 5. Konfigurowanie środowiska skryptowego	121
Konfigurowanie profilu	121
Tworzenie aliasów	122
Tworzenie funkcji	125
Przesłanie istniejących poleceń	126
Przekazywanie wielu parametrów	129
Tworzenie zmiennych	134
Tworzenie dysków PowerShell	141
Włączanie obsługi skryptów	146
Tworzenie profilu	148
Wybór odpowiedniego profilu	148
Tworzenie innych profili	150
Używanie funkcji z innych skryptów	152
Tworzenie biblioteki funkcji	153
Dołączanie pliku	154
Dodatkowe źródła informacji	156
 Rozdział 6. Unikanie pułapek podczas pisania skryptów	 157
Brak obsługi poleceń	157
Skomplikowane konstruktory	159
Kwestie dotyczące zgodności wersji	160
Sprawdzanie wersji systemu operacyjnego	165
Brak obsługi WMI	167
Praca z obiektami i przestrzeniami nazw	167
Pobieranie listy dostawców WMI	172
Praca z klasami WMI	173
Zmienianie ustawień	176
Modyfikowanie wartości przez rejestr	178
Brak obsługi platformy .NET	182
Używanie statycznych metod i własności	182
Zależność od wersji	185
Brak obsługi COM	185
Brak obsługi aplikacji zewnętrznych	191
Dodatkowe źródła informacji	195
 Rozdział 7. Śledzenie możliwości zastosowania skryptów	 197
Ewaluacja potrzeby napisania skryptu	197
Odczytywanie pliku tekstowego	198
Eksportowanie historii poleceń	204
Polecenia promienne	205
Wysyłanie zapytań do Active Directory	208
Po prostu używaj wiersza poleceń	214

Obliczanie korzyści z użycia skryptu	217
Powtarzalność	218
Możliwość opisanie w dokumentacji	222
Zdolność do adaptacji	223
Współpraca nad skryptami	227
Dodatkowe źródła informacji	227

Część III Projektowanie skryptów 229

Rozdział 8. Projektowanie skryptów 231

Podstawowe wiadomości o funkcjach	231
Definiowanie funkcji w celu ułatwienia wielokrotnego wykorzystania kodu	240
Definiowanie funkcji z dwoma parametrami	244
Ograniczenia typu	249
Funkcje przyjmujące więcej niż dwa parametry	252
Definiowanie logiki biznesowej w funkcjach	254
Definiowanie funkcji w celu ułatwienia modyfikacji skryptów	256
Podstawowe wiadomości o filtrach	263
Dodatkowe źródła informacji	268

Rozdział 9. Projektowanie pomocy do skryptów 271

Dodawanie dokumentacji w komentarzach jednowierszowych	271
Praca z folderami tymczasowymi	277
Używanie wielowierszowych znaczników komentarzowych w Windows PowerShell 4.0	279
Tworzenie komentarzy wielowierszowych za pomocą znaczników komentarzowych	279
Tworzenie jednowierszowych komentarzy przy użyciu znaczników komentarzowych	280
Używanie pomocy komentarzowej	281
13 zasad pisania efektywnych komentarzy	286
Aktualizuj dokumentację razem ze skryptem	287
Dodawaj komentarze podczas pisania kodu	288
Pisz z myślą o użytkownikach z różnych krajów	288
Zamieszczaj informacje w nagłówku	290
Podaj listę warunków używania	291
Opisuj niedoskonałości	292
Nie dodawaj zbędnych informacji	293
Uzasadniaj powody napisania kodu	293
Zastosowanie komentarzy jednowierszowych	294
Unikaj komentarzy na końcu wiersza	295
Opisuj struktury zagnieżdżone	295

Używaj standardowego zestawu słów kluczowych	296
Opisuj wszelkie dziwne fragmenty kodu	297
Dodatkowe źródła informacji	300
Rozdział 10. Projektowanie modułów	301
Podstawowe wiadomości o modułach	301
Znajdowanie i ładowanie modułów	302
Wyświetlanie listy dostępnych modułów	302
Ładowanie modułów	305
Instalowanie modułów	308
Tworzenie folderu na moduły	309
Sposób użycia zmiennej \$modulePath	311
Tworzenie dysku modułu	313
Sprawdzanie zależności modułów	314
Używanie modułów pochodzących z udziałów	318
Tworzenie modułu	319
Dodatkowe źródła informacji	325
Rozdział 11. Obsługa wejścia i wyjścia	327
Wybór najlepszej metody pobierania danych	328
Wczytywanie danych z wiersza poleceń	328
Sposób użycia instrukcji Param	335
Pobieranie haseł na wejściu	348
Pobieranie łańcuchów połączenia	356
Monitowanie o informacje	357
Wybór najlepszej metody zwracania danych	358
Wysyłanie informacji na ekran	360
Wysyłanie wyników do pliku	366
Wysyłanie danych równocześnie na ekran i do pliku	367
Wysyłanie wyników na adres e-mail	371
Wysyłanie wyników z funkcji	371
Dodatkowe źródła informacji	377
Rozdział 12. Obsługa błędów	379
Obsługa brakujących parametrów	380
Przypisywanie parametrowi wartości domyślnej	380
Tworzenie parametrów obowiązkowych	381
Ograniczanie możliwości wyboru	382
Ograniczanie liczby opcji za pomocą metody PromptForChoice	382
Znajdowanie dostępnych komputerów za pomocą polecenia ping	384
Sprawdzanie zawartości tablicy za pomocą operatora -contains	385
Testowanie własności za pomocą operatora -contains	387

Postępowanie w przypadku braku uprawnień	389
Podejmowanie nieudanych prób	391
Sprawdzanie, czy skrypt posiada potrzebne uprawnienia, i eleganckie kończenie pracy	393
Sposób użycia instrukcji #Requires	393
Postępowanie w przypadku braku dostawców WMI	396
Niepoprawne typy danych	403
Błędy zakresu	408
Sposób użycia funkcji sprawdzającej wartości graniczne	408
Ograniczanie dopuszczalnych wartości parametru	409
Dodatkowe źródła informacji	410

Rozdział 13. Testowanie skryptów 411

Techniki testowania podstawowej składni	411
Szukanie błędów	415
Uruchamianie skryptu	417
Dokumentowanie pracy	419
Testowanie wydajności skryptów	421
Zapisywanie i przeglądanie danych	421
Użycie potoku Windows PowerShell	423
Szacowanie wydajności różnych wersji skryptu	426
Sposób użycia parametrów standardowych	434
Sposób użycia parametru debug	434
Sposób użycia parametru Verbose	436
Sposób użycia parametru whatif	437
Tworzenie dzienników za pomocą funkcji Start-Transcript	441
Zaawansowane techniki testowania skryptów	443
Dodatkowe źródła informacji	445

Rozdział 14. Dokumentowanie skryptów 447

Pobieranie dokumentacji z pomocy	447
Pobieranie dokumentacji z komentarzy	452
Sposób użycia parsera AST	455
Dodatkowe źródła informacji	457

Część IV Wdrażanie skryptu 459

Rozdział 15. Ustawianie zasad wykonywania skryptów 461

Wybór zasady wykonywania dla skryptu	461
Przeznaczenie zasad wykonywania skryptów	462
Różne zasady wykonywania skryptów	462
Co to jest strefa internetowa	463

Wdrażanie zasady wykonywania skryptu	465
Modyfikowanie rejestru	465
Użycie polecenia Set-ExecutionPolicy	466
Wdrażanie zasady wykonywania skryptów za pomocą zasady grupowej	469
Podpisywanie kodu	472
Dodatkowe źródła informacji	474
Rozdział 16. Uruchamianie skryptów	475
Skrypty logowania	475
Co uwzględnić w skryptach logowania	476
Metody wywoływania skryptów logowania	480
Folder skryptów	482
Wdrażanie lokalne	482
Lokalne wdrażanie pakietu MSI	482
Samodzielne skrypty	483
Diagnostyka	483
Raportowanie i kontrolowanie	483
Skrypty pomocy technicznej	484
Unikaj edytowania	484
Dostarcz dobrej jakości pomoc	484
Dodatkowe źródła informacji	487
Rozdział 17. Kontrola wersji skryptów	489
Dlaczego warto stosować kontrolę wersji	489
Unikanie wprowadzania błędów	490
Precyzyjne usuwanie usterek	491
Śledzenie zmian	491
Lista skryptów	491
Zachowanie zgodności z innymi skryptami	491
Wewnętrzny numer wersji w komentarzach	493
Programy do kontroli wersji	496
Dodatkowe źródła informacji	497
Rozdział 18. Rejestrowanie wyników	499
Zapisywanie danych w pliku tekstowym	499
Projektowanie metody rejestrowania wyników skryptu	500
Miejsce przechowywania tekstu	509
Przechowywanie dzienników w lokalizacjach sieciowych	513
Zapisywanie danych w dzienniku zdarzeń	517
Sposób użycia dziennika aplikacji	519
Tworzenie własnego dziennika zdarzeń	519
Zapisywanie danych w rejestrze	520
Dodatkowe źródła informacji	522

Rozdział 19. Rozwiązywanie problemów ze skryptami	523
Podstawy debugowania w Windows PowerShell	523
Błędy składniowe	524
Błędy wykonawcze	524
Błędy logiczne	527
Sposób użycia polecenia Set-PSDebug	530
Śledzenie wykonywania skryptu	531
Wykonywanie skryptu krok po kroku	535
Włączanie trybu ścisłego	542
Debugowanie skryptów	545
Ustawianie punktów wstrzymania	547
Reagowanie na punkty wstrzymania	555
Wyświetlanie listy punktów wstrzymania	556
Włączanie i wyłączanie punktów wstrzymania	558
Usuwanie punktów wstrzymania	559
Dodatkowe źródła informacji	561
 Rozdział 20. Praca w środowisku Windows PowerShell ISE	 563
Uruchamianie środowiska Windows PowerShell ISE	563
Zawartość okna Windows PowerShell ISE	563
Praca w okienku skryptu	565
Rozwijanie nazw za pomocą klawisza Tab i funkcja IntelliSense	567
Sposób użycia fragmentów kodu w środowisku Windows PowerShell ISE	569
Tworzenie skryptów przy użyciu gotowych fragmentów kodu	569
Tworzenie nowych gotowych fragmentów kodu w Windows PowerShell ISE	569
Usuwanie fragmentów kodu zdefiniowanych przez użytkownika	570
Dodatkowe źródła informacji	571
 Rozdział 21. Narzędzia do pracy zdalnej i zadania Windows PowerShell	 573
Narzędzia do pracy zdalnej Windows PowerShell	573
Klasyczne sposoby pracy zdalnej	573
Zdalne zarządzanie systemem Windows	583
Zadania Windows PowerShell	590
Dodatkowe źródła informacji	596
 Rozdział 22. Przepływy pracy w Windows PowerShell	 597
Do czego służą przepływy pracy w Windows PowerShell	597
Wymagania przepływów pracy	598
Prosty przepływ pracy	598
Równoległe wykonywanie poleceń	599
Aktywności przepływów pracy	602
Polecenia Windows PowerShell jako aktywności	602
Niedozwolone polecenia rdzenne	604

Nieautomatyczne aktywności z poleceń	604
Aktywności równoległe	605
Ustalanie punktów kontrolnych dla przepływów pracy w Windows PowerShell	606
Czym są punkty kontrolne	606
Tworzenie punktów kontrolnych	606
Dodawanie punktów kontrolnych	607
Dodawanie aktywności sekwencyjnej do przepływu pracy	609
Dodatkowe źródła informacji	611

Rozdział 23. Usługa konfiguracji żądanego stanu programu PowerShell 613

Podstawowe informacje o Usłudze konfiguracji żądanego stanu programu PowerShell	613
Proces DSC	614
Parametry konfiguracji	617
Ustawianie zależności	618
Dane konfiguracji	619
Kontrolowanie dryfu konfiguracji	623
Dodatkowe źródła informacji	625

O autorze	627
------------------------	------------

Skorowidz	629
------------------------	------------

Rozdział 7

Śledzenie możliwości zastosowania skryptów

- Ocenianie, czy skrypt jest potrzebny
- Obliczanie korzyści z zastosowania skryptu
- Współpraca nad skryptami
- Dodatkowe źródła informacji

Należy śledzić możliwości zastosowania skryptów, aby mieć pewność, że najbardziej korzystne skrypty zostaną napisane najszybciej. Taki spis możliwych do oskryptowania przypadków może efektywnie służyć do zarządzania skryptami w przedsiębiorstwie. Kluczem jest właściwe posługiwanie się tym zbiorem informacji.

Ewaluacja potrzeby napisania skryptu

Nie wszystko w konsoli Windows PowerShell 4.0, podobnie jak w Windows PowerShell 3.0, musi być oskryptowane. Programiści języków Microsoft VBScript i Perl często mają odczucie, że w danej sytuacji powinni napisać skrypt. Ale wiele czynności można wykonać w wierszu poleceń bez używania jakiegokolwiek skryptu.

Jedną z wielkich zalet konsoli Windows PowerShell jest możliwość używania instrukcji językowych bezpośrednio w wierszu poleceń. Instrukcja `for` umożliwia tworzenie pętli, do obsługi których w innych językach konieczne byłoby napisanie skryptu. Aby ułatwić pracę w wierszu poleceń, twórcy konsoli Windows PowerShell umożliwili dzielenie poleceń na kilka wierszy. Po zakończeniu wpisywania można po raz drugi nacisnąć klawisz *Enter*, aby je wykonać. Poniżej znajduje się polecenie wysyłające polecenie `ping` do wszystkich adresów IP z przedziału od 192.168.2.1 do 192.168.2.10:

```
PS C:\> for($i = 1 ; $i -le 10 ; $i++)
>> { Test-Connection -Destination 192.168.2.$i -Count 1 -ErrorAction SilentlyContinue
|
>> Format-Table -property Address, statusCode, ResponseTime -AutoSize }
>>
```

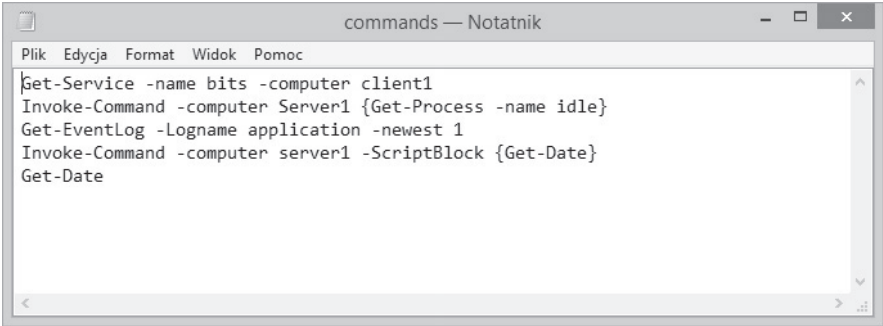
Address	statusCode	ResponseTime
-----	-----	-----
192.168.2.1	0	1
Address	statusCode	ResponseTime
-----	-----	-----
192.168.2.3	0	2
Address	statusCode	ResponseTime
-----	-----	-----
192.168.2.5	0	0
Address	statusCode	ResponseTime
-----	-----	-----
192.168.2.10	0	10

Jeśli zastosuje się parę sztuczek składni Windows PowerShell, np. aliasy, częściowe parametry i argumenty pozycyjne, powyższe polecenie można znacznie skrócić. Poniżej znajduje się właśnie taka skrócona wersja:

```
1..10 | % {Test-Connection 10.1.1.$_ -cou 1 -ea 0 | ft Address, StatusCode, ResponseTime -au}
```

Odczytywanie pliku tekstowego

Najprostszy skrypt Windows PowerShell jest zbiorem poleceń PowerShell zapisanych w pliku o określonym rozszerzeniu. Jeśli ktoś nie chce pisać skryptu, może zapisać zbiór poleceń w pliku tekstowym, jak pokazano na rysunku 7.1.



RYSUNEK 7.1. Plik zawierający zbiór poleceń Windows PowerShell

Przy użyciu konsoli Windows PowerShell można odczytać plik *commands.txt* i wykonać znajdujące się w nim polecenia za pomocą polecenia *Get-Content*, które służy właśnie do pobierania poleceń z plików tekstowych. Domyślnym parametrem polecenia *Get-Content* jest path. Podczas pracy w wierszu poleceń nie trzeba go podawać. Ścieżka może być lokalna lub nawet UNC (ang. *Universal Naming Convention*), pod warunkiem że ma się odpowiednie uprawnienia do odczytu pliku tekstowego. Najlepszym sposobem użycia tej techniki jest przekazanie wyników do polecenia *Invoke-Expression*. Każde polecenie przekazywane przez potok do tego polecenia zostaje przez nie wykonane, jak pokazano poniżej:

```
Get-Content -Path C:\fso\Commands.txt | Invoke-Expression
```

Wynik tego polecenia pokazano na rysunku 7.2.

```

PS C:\> Get-Content -Path .\bin\Commands.txt | Invoke-Expression

Status      Name                DisplayName
-----
Running     bits                Background Intelligent Transfer Ser...

Id           : 0
Handles      : 0
CPU          :
Name         : Idle
PSComputerName : Server1

MachineName   : client1.iammred.net
Data          : {}
Index         : 685
Category      : (0)
CategoryNumber : 0
EventID       : 1005
EntryType     : Information
Message       : The description for Event ID '1073742829' in Source
                'Customer Experience Improvement Program' cannot be
                found. The local computer may not have the necessary
                registry information or message DLL files to display the
                message, or you may not have permission to access them.
                The following information is part of the event:
Source        : Customer Experience Improvement Program
ReplacementsStrings : {}
InstanceId    : 1073742829
TimeGenerated : 8/27/2013 3:00:01 PM

```

RYSUNEK 7.2. Polecenia Windows PowerShell bez problemu przetwarzają zawartość plików tekstowych i wykonują znajdujące się w nich polecenia

Podczas korzystania z funkcji pracy zdalnej konsoli Windows PowerShell do pracy z niezaufaną domeną łatwo się pomylić przy używaniu takich poleceń jak `Get-Content`. Parametr `-path` odnosi się do ścieżki lokalnej dla komputera docelowego, a nie tego, z którego wykonuje się polecenie. W poniższym przykładzie ścieżka `c:\fso\commands.txt` wskazuje plik tekstowy o nazwie `commands.txt` znajdujący się w folderze `fso` na dysku C komputera o nazwie Sydney w domenie Woodbridgebank.com. Jeśli plik ten nie zostanie tam znaleziony, wystąpi błąd.

```

PS C:\> invoke-command -ComputerName sydney.woodbridgebank.com -Credential
administrator@woodbridgebank.com -ScriptBlock {get-content -Path C:\fso\Commands.txt
| Invoke-Expression}
Invoke-Command : Cannot find path 'C:\fso\Commands.txt' because it does not exist.At
line:1 char:15
+ invoke-command <<<< -ComputerName sydney.woodbridgebank.com -Credential
administrator@woodbridgebank.com -ScriptBlock {get-content -Path
C:\fso\Commands.txt | Invoke-Expression}
+ CategoryInfo          : ObjectNotFound: (C:\fso\Commands.txt:String)
[Get-Content], ItemNotFoundException
+ FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.GetContentCommand

```

Pewnie zastanawiasz się, czy możesz wskazać plik `commands.txt` na komputerze, z którego wykonywane jest polecenie, za pomocą ścieżki UNC. Jako że zdalna domena jest niezaufana, nie ma żadnego kontekstu zabezpieczeń, który pozwoliłby zdalnemu poleceniu uzyskać dostęp do systemu plików komputera lokalnego. Polecenie znajdujące się w parametrze `ScriptBlock` zostanie wykonane w kontekście komputera docelowego, którym w tym przypadku jest Sydney.Woodbridgebank.com.

Komputer lokalny, z którego wykonywane jest polecenie, nazywa się Windows 8.NWTraders.com. Jako że te dwie domeny nie mają zaufanych relacji, nie można podać danych poświadczających, które umożliwiłyby wykonanie polecenia. Wynik próby wykonania polecenia przedstawiono poniżej:

```
PS C:\> invoke-command -ComputerName sydney.woodbridgebank.com -Credential
administrator@woodbridgebank.com -ScriptBlock {get-content -Path '\\Windows 8\fso\Commands.txt'
| Invoke-Expression}
Invoke-Command : Cannot find path '\\Windows 8\fso\Commands.txt' because it does not exist.
At line:1 char:15
+ invoke-command <<<< -ComputerName sydney.woodbridgebank.com -Credential
administrator@woodbridgebank.com -ScriptBlock {get-content -Path '\\Windows 8\fso\Commands.txt'
| Invoke-Expression}
    + CategoryInfo          : ObjectNotFound: (\\Windows 8\fso\Commands.txt:String)
      [Get-Content], ItemNotFoundException
    + FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.GetContentCommand
```

Mylące może być to, że polecenie `Get-Content` w pojedynkę działa bardzo dobrze. Na komputerze o nazwie *client1* zawierającym folder o nazwie *bin*, w którym znajduje się plik tekstowy o nazwie *commands.txt*, polecenie to zostanie wykonane, pod warunkiem że umieścimy je w pojedynczym cudzysłowie, jak pokazano poniżej:

```
PS C:\> Get-Content -Path '\\client1\bin\Commands.txt'
Get-Service -Name bits -ComputerName Windows 8
Get-Process -Name explorer -ComputerName berlin
Get-EventLog -LogName application -Newest 1 -ComputerName berlin,Windows 8
Invoke-Command -ComputerName Berlin { Get-Date }
Get-Date
```

Ale tego należało się spodziewać, ponieważ zalogowany użytkownik ma uprawnienia dostępu do użytego folderu, więc może też wczytać za pomocą polecenia `Get-Content` ścieżkę UNC do pliku *commands.txt*.

Można zmapować dysk na zdalnej domenie i skopiować plik z lokalnego komputera do odpowiedniego folderu na serwerze zdalnym. Oczywiście konieczne będzie otwarcie dodatkowych portów w zaporze systemu Windows, co może, ale nie musi być wykonalne, w zależności od konfiguracji sieci. Jeśli zdecydujesz się na takie rozwiązanie, możesz dokonać zmian konfiguracyjnych za pomocą konsoli Windows PowerShell, jak pokazano poniżej:

```
PS C:\> Invoke-Command -ComputerName Sydney.WoodBridgeBank.Com -Credential
Administrator@WoodbridgeBank.com -ScriptBlock { netsh advfirewall firewall set rule group="File
and Printer Sharing" new enable=Yes }

Updated 28 rule(s).
Ok.
```

Po wprowadzeniu wyjątku do zapory można zmapować dysk przy użyciu graficznego interfejsu użytkownika, polecenia `Net Use` z poziomu konsoli Windows PowerShell lub dowolną inną metodą. Po zmapowaniu dysku można skopiować plik *commands.txt* na serwer za pomocą polecenia `Copy-Item`, jak pokazano poniżej:

```
Copy-Item -Path C:\fso\Commands.txt -Destination z:
```

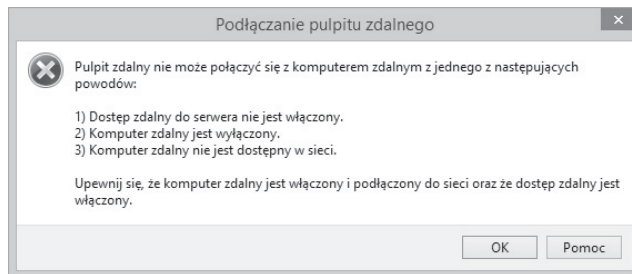
UWAGA

Podczas kopiowania obiektów na zmapowany dysk za pomocą polecenia `Copy-Item` należy mieć na uwadze strukturę tego zmapowanego dysku. Często mapuje się dyski w celu umożliwienia używania ich na zdalnym komputerze. Zdalny udział prawie zawsze jest udziałem folderu, a nie całego dysku. Jako że nasz dysk zdalny jest punktem mapowania jednego folderu, miejsce docelowe ulega zmianie. W poniższym poleceniu dysk *Z* jest udziałem folderu *Fso* na serwerze zdalnym. Parametr `-destination` kieruje do korzenia zmapowanego dysku, a nie do folderu *Z:\fso*.

Teraz można użyć pliku *commands.txt* bezpośrednio w poleceniu Windows PowerShell, jak pokazano poniżej:

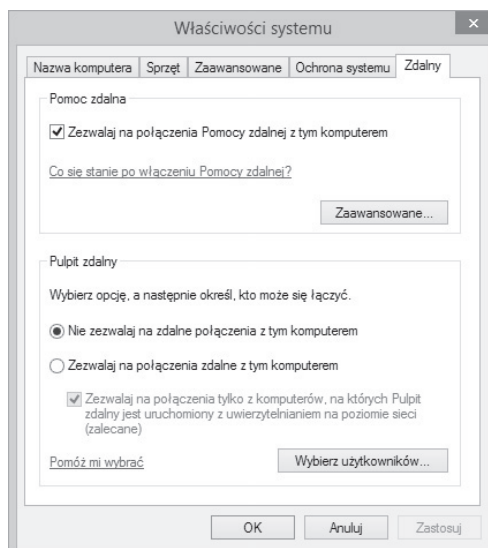
```
PS C:\> invoke-command -ComputerName Sydney.WoodbridgeBank.com -Credential administrator@WoodBridgeBank.com -ScriptBlock { Get-Content -Path C:\fso\Commands.txt | Invoke-Expression }
```

Jednym z rozwiązań dylematu mapowania dysków jest użycie pulpitu zdalnego, który umożliwia dostęp do zasobów lokalnych, jeśli chcemy mieć je dostępne w swojej sesji. Wybierając pulpit zdalny, klikając przycisk *Options* i wybierając kartę *Local resources*, możemy wybrać połączenia drukarki, dostęp do schowka oraz udostępnić lokalne dyski w sesji RDP (ang. *Remote Desktop Protocol*). Ustawienia pulpitu zdalnego można otworzyć, klikając kolejno *Start/Wszystkie programy/Akcesoria/Podłączanie pulpitu zdalnego*. Jeśli pulpit zdalny nie został jeszcze skonfigurowany, system wyświetli informację o braku dostępu widoczną na rysunku 7.3.



RYSUNEK 7.3. Odmowa dostępu podczas próby połączenia się z pulpitem zdalnym

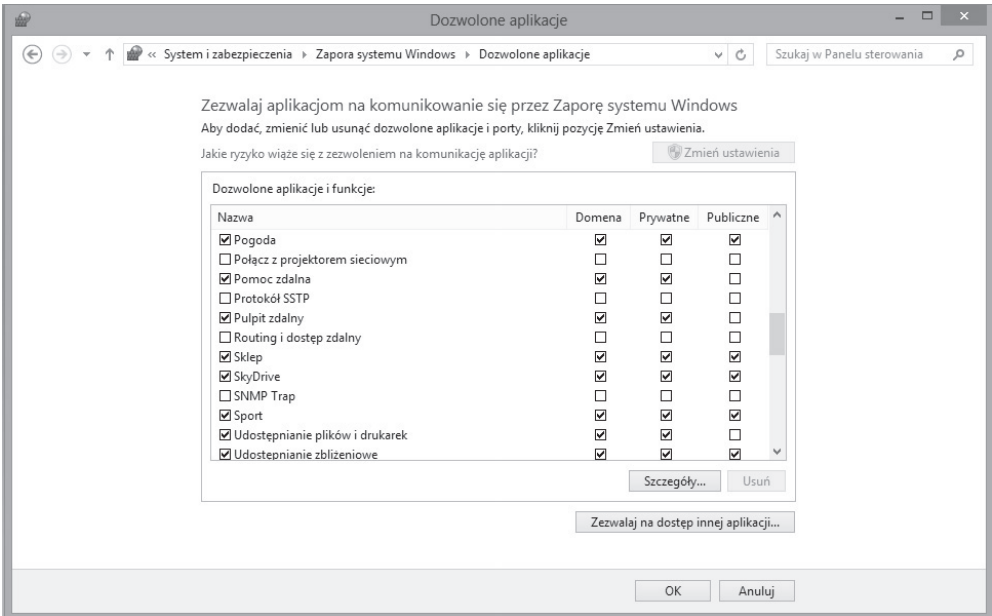
Aby włączyć zdalny pulpit w systemach Microsoft Windows 2012 i Windows 2012 R2, należy w menedżerze serwera w węźle *Server lokalny* wybrać opcję *Pulpit zdalny*. W systemach Windows 8 i Windows 8.1 należy w Panelu sterowania otworzyć aplet *System*, kliknąć odnośnik *Ustawienia zdalne*. Pojawi się okno dialogowe *Właściwości systemu* z otwartą kartą *Zdalny*, jak widać na rysunku 7.4.



RYSUNEK 7.4. Pulpit zdalny musi być włączony

W dolnej części okna znajdują się trzy opcje dotyczące pulpitu zdalnego. Domyślnie połączenie z pulpitem zdalnym jest zabronione. Najbezpieczniejszym rozwiązaniem jest zaznaczenie opcji *Zezwalaj na połączenia tylko z komputerów, na których Pulpit zdalny jest uruchomiony z uwierzytelnianiem na poziomie sieci (zalecane)*. Dodatkowo można określić, którzy użytkownicy mogą nawiązywać połączenie. Domyślnie uprawnienia takie mają wszyscy członkowie grupy administratorów domeny.

Po włączeniu pulpitu zdalnego automatycznie zostanie utworzony wyjątek umożliwiający przepływ ruchu RDP przez zaporę systemu Windows. Warto dokładnie sprawdzić, czy na pewno wyjątek został utworzony. Na rysunku 7.5 widać wyjątek zapory w systemie Windows 8.1.



RYSUNEK 7.5. Pulpit zdalny musi mieć wyjątek w zaporze systemu Windows

Zapiski praktyka

Brian Wilhite

Premier Field Engineer (PFE), Microsoft Corporation

W ramach swoich codziennych obowiązków administratora systemów pracującego w dość dużej firmie często muszę rozwiązywać problemy wymagające użycia konsoli Windows PowerShell. Najczęściej korzystam z Instrumentacji zarządzania Windows (WMI), która jest moją ulubioną technologią w systemach Windows. Konsola PowerShell ułatwia pracę z WMI na niespotykaną wcześniej skalę. Jedną z najlepszych rzeczy, jakie można robić za pomocą narzędzi WMI i Windows PowerShell, jest tworzenie procesów na zdalnych komputerach. To bardzo przydatna możliwość dla każdego, kto musi instalować, aktualizować lub odinstalowywać programy na komputerach z systemem Windows w swoim środowisku, w którym nie ma programów do zarządzania typu System Center Configuration Manager.

Ostatnio na naszych serwerach ktoś przez pomyłkę zainstalował pewien program. Aby go szybko usunąć, postanowiłem za pomocą WMI zdalnie utworzyć proces *MSIExec.exe* z ciągiem dezinstalacji dla MSI. Ciąg dezinstalacji (*UninstallString*) dla każdego programu można znaleźć w rejestrze, typowo w następującej lokalizacji (<SID> to identyfikator SID użytkownika, z którym został zainstalowany dany program, a <GUID produktu> to niepowtarzalny identyfikator interesującego nas produktu): *HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Installer\UserData\<SID>\Products\<GUID produktu>\InstallProperties*.

W ten sposób można znaleźć ciąg dezinstalacji (*UninstallString*) wybranego programu, który chcemy usunąć z systemu. Usunąłem prawdziwy identyfikator GUID produktu, aby nie oczerniać niewinnej firmy:

```
MsiExec.exe /X{<GUID produktu>} /quiet
```

W tym przypadku wiem, bo to sprawdziłem, że przełącznik */quiet* zadziała dla produktu, który chcę usunąć. Ale Ty możesz mieć inne doświadczenia.

Do wykonania tego zadania użyjemy polecenia *Invoke-WmiMethod*. Ponadto użyjemy metody *Create* klasy *Win32_Process* w celu utworzenia procesu na zdalnym komputerze. Najlepiej mieć włączone narzędzia do pracy zdalnej Windows PowerShell w swoim komputerze, ale jeśli nie są włączone, nie panikuj. Opisuję kilka technik realizacji tego zadania.

W pierwszej technice przyjąłem założenie, że na docelowych komputerach zdalnych włączone są narzędzia do pracy zdalnej Windows PowerShell. Zmienna *\$Computers* zawiera zbiór nazw komputerów, których użyjemy jako urządzeń docelowych:

```
Invoke-Command -ScriptBlock {
    Invoke-WmiMethod -Class Win32_Process '
        -Name Create '
        -ArgumentList 'MsiExec.exe /X{<Product GUID>} /quiet' '
        -ComputerName $Computers
```

Drugą technikę można zastosować, gdy narzędzia do pracy zdalnej Windows PowerShell na komputerach zdalnych nie są włączone:

```
$Computers | ForEach-Object {
    Invoke-WmiMethod -Class Win32_Process '
        -Name Create '
        -ArgumentList 'MsiExec.exe /X{<Product GUID>} /quiet' '
        -ComputerName $_}
```

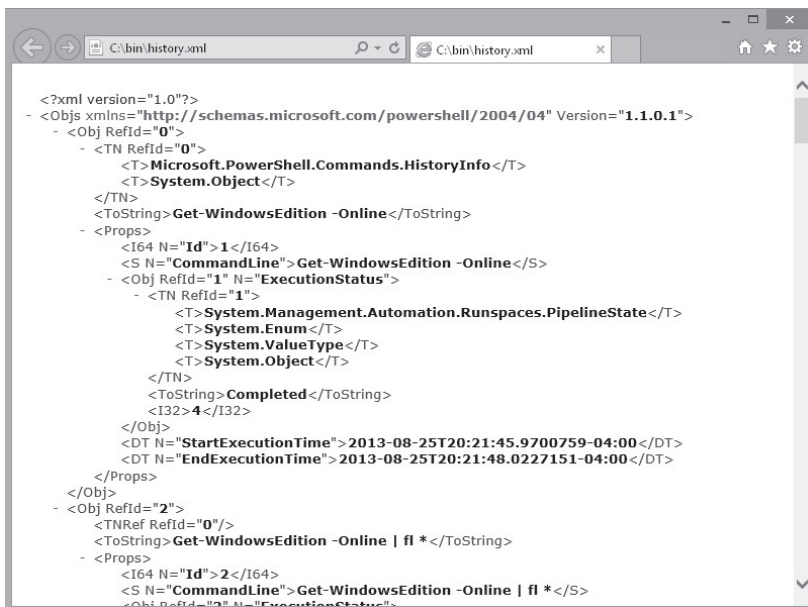
Z własnego doświadczenia wiem, że w takich sytuacjach jak ta, gdy komputerów są setki lub nawet tysiące, bardziej niezawodnym rozwiązaniem jest przekazanie nazw komputerów do polecenia *ForEach-Object* zamiast przekazywania całej listy do parametru *ComputerName*. Oba przedstawione rozwiązania powinny zwrócić obiekt zawierający *ReturnValue* i *ProcessId*. Jeśli tworzenie procesu powiedzie się, *ReturnValue* będzie mieć wartość 0. Zero wskazuje, że proces został utworzony, ale nie mówi nic o tym, czy dezinstalacja się powiodła. Dlatego należy to jeszcze sprawdzić w dziennikach zdarzeń wszystkich systemów.

Eksportowanie historii poleceń

Wiele prac administracyjnych wykonywanych przy użyciu konsoli Windows PowerShell polega na wpisywaniu serii poleceń w konsoli. Niezależnie od tego, czy edytujemy rejestr, czy zatrzymujemy różne procesy i usługi, aby zapewnić spójne środowisko operacyjne, należy skopiować prace konfiguracyjne na kilka różnych serwerów. W przeszłości trzeba było do tego tworzyć skrypty. Jeśli polecenia do zduplikowania są serią poleceń wpisywanych w konsoli, można obejść się bez skryptu, eksportując historię poleceń do pliku *.xml* za pomocą polecenia `Get-History`, jak pokazano poniżej:

```
Get-History | Export-Clixml -Path C:\fso\history.xml
```

Wynikiem tego polecenia jest plik *.xml* zawierający wszystkie polecenia, jakie wpisano w konsoli. Na rysunku 7.6 widać zawartość takiego przykładowego pliku:



RYСУNEK 7.6. Plik *.xml* z historią poleceń

Polecenia z tego pliku można importować za pomocą polecenia `Import-Clixml`. Jego wynik przekazuje się do polecenia `Add-History`, aby polecenia zawarte w zaimportowanym pliku dodać do historii konsoli. Należy użyć przełącznika `-passthru`, aby polecenia przekazać zarówno do polecenia `Add-History`, jak i `ForEach-Object`. W poleceniu `ForEach-Object` można wykonać każde polecenie z historii za pomocą polecenia `Invoke-History`. Poniżej przedstawiono opisywany zestaw poleceń wraz z wynikiem jego wykonania:

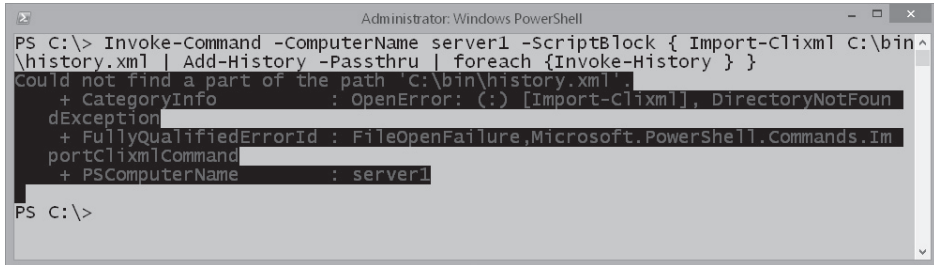
```
PS C:\> Import-Clixml -Path C:\fso\history.xml | Add-History -Passthru |
>> ForEach-Object { Invoke-History }
>>
if(!(test-path -path c:\fso4)) { new-item c:\fso4 -ItemType directory }

Directory: C:\
```

Mode	LastWriteTime	Length	Name
d----	1/9/2009 12:33 AM		fso4

```
Get-Command >> C:\fso4\commands.txt
notepad C:\fso4\commands.txt
```

Technikę tę można też zastosować zdalnie, posługując się poleceniem `Invoke-Command`. Pamiętaj, że ścieżka `path` jest względna w odniesieniu do komputera będącego celem, a nie komputera wykonującego polecenie. Jeśli o tym zapomnisz, wystąpi błąd, jak widać na rysunku 7.7.



RYСУNEK 7.7. Błąd spowodowany użyciem lokalnych ścieżek do plików

Jeśli najpierw skopiujesz plik na komputer docelowy i dostosujesz swój wiersz poleceń, to technika polegająca na zaimportowaniu i wykonaniu historii sprawdzi się wyśmienicie. Zaletą konsoli Windows PowerShell jest to, że z poleceniem `Copy-Item` można używać ścieżek UNC. Właśnie to sprawia, że technika ta jest godna uwagi, ponieważ umożliwia łatwe przenoszenie plików na zdalny komputer, jak widać poniżej:

```
PS C:\> Copy-Item C:\fso\history.xml \\berlin\c$\fso
PS C:\> Import-Clixml -Path C:\fso\history.xml | Add-History -Passthru | ForEach -Object {
Invoke-History }
if(!(test-path -path c:\fso4)) { new-item c:\fso4 -ItemType directory }
```

Directory: C:\

Mode	LastWriteTime	Length	Name
d----	1/9/2009 12:40 AM		fso4

```
Get-Command >> C:\fso4\commands.txt
notepad C:\fso4\commands.txt
```

Polecenia promieniste

Polecenia promieniste (ang. *fan-out command*) to takie, które uruchamia się z centralnego komputera na pewnej liczbie zdalnych komputerów. Jednym ze sposobów ich wykonywania jest użycie polecenia `Invoke-Command`, jak pokazano poniżej:

```
PS C:\> Invoke-Command -Computer berlin,Windows 8 -Script '
>> {"$env:computername $(get-date)"}
>>
WINDOWS 8 01/09/2009 08:31:42
BERLIN 01/09/2009 08:31:47
```

W sposób promienisty można wykonać wiele poleceń, przekazując im tablicę nazw komputerów jako parametr `-computername`. Wadą tej metody jest to, że zwrócone wyniki są prawie bezużyteczne. Najlepiej wytłumaczyć to na konkretnym przykładzie. Poniżej użyto polecenia `Get-Service` do pobrania informacji o konfiguracji usług z dwóch komputerów. Pierwszy z nich nazywa się `Windows 8`, a drugi `Berlin`. Jak widać, wyniki polecenia dla tych dwóch komputerów są pomieszczone i nie ma kolumny informującej, do którego komputera odnosi się dana informacja. W związku z tym wynik ten jest przydatny tylko w tym względzie, że pozwala szybko przejrzeć usługi na dwóch komputerach i znaleźć różnice w ich konfiguracji. Poniżej znajduje się omawiane polecenie i część zwróconego przez nie wyniku:

```
PS C:\> Get-Service -ComputerName Windows 8, Berlin
Status   Name                DisplayName
-----
Running  1-vmsvc             Virtual Machine Additions Services ...
Running  1-vmsvc             Virtual Machine Additions Services ...
Running  AeLookupSvc         Application Experience
Stopped  AeLookupSvc         Application Experience
Stopped  ALG                 Application Layer Gateway Service
Stopped  ALG                 Application Layer Gateway Service
Stopped  Appinfo             Application Information
Stopped  Appinfo             Application Information
Stopped  AppMgmt             Application Management
Stopped  AppMgmt             Application Management
Stopped  AudioEndpointBu...  Windows Audio Endpoint Builder
Stopped  AudioEndpointBu...  Windows Audio Endpoint Builder
Stopped  Audiosrv Windows    Audio
Stopped  Audiosrv Windows    Audio
Running  BFE Base Filtering Engine
Running  BFE Base Filtering Engine
Running  BITS Background     Intelligent Transfer Ser...
Stopped  BITS Background     Intelligent Transfer Ser...
Stopped  Browser Computer    Browser
Running  Browser Computer    Browser
>>> reszta wyników opuszczona >>>
```

Jak widać w powyższych danych, na jednym komputerze usługa `AeLookupSvc` jest uruchomiona, a na drugim zatrzymana. Aby dowiedzieć się, na którym działa, a na którym nie, wystarczy użyć polecenia `Get-Service`.

```
PS C:\> Get-Service -Name AeLookupSvc -computer Windows 8
Status   Name                DisplayName
-----
Stopped  AeLookupSvc         Application Experience

PS C:\> Get-Service -Name AeLookupSvc -computer Berlin
Status   Name                DisplayName
-----
Running  AeLookupSvc         Application Experience
```

Możesz pomyśleć, że pierwsze wystąpienie nazwy usługi należy do komputera, który jest pierwszy na liście. Ale jak widać, usługa `AeLookupSvc` jest uruchomiona na komputerze `Berlin`, a zatrzymana na komputerze `Windows 8`. W wyniku jest właśnie taka kolejność, ale w poleceniu promienistym najpierw wymieniony jest komputer `Windows 8`. Teraz pewnie pomyślisz, że najpierw wyświetlane są wyniki dla pierwszego komputera, a potem dla drugiego. Ale zanim

dojdiesz do wniosku, że tak musi być, sprawdź jeszcze jakąś inną usługę. W wyniku polecenia usługa BITS najpierw jest oznaczona jako uruchomiona, a potem jako zatrzymana. Sprawdźmy, jaki jest jej stan na poszczególnych komputerach:

```
PS C:\> Get-Service -Name Bits -computer berlin
Status Name DisplayName
-----
Stopped BITS Background Intelligent Transfer Ser...
```

```
PS C:\> Get-Service -Name Bits -computer Windows 8
Status Name DisplayName
-----
Running BITS Background Intelligent Transfer Ser...
```

Jak widać, usługa BITS jest zatrzymana na komputerze Berlin i uruchomiona na komputerze Windows 8. Wniosek z tego taki, że używając polecenia `Get-Service` w sposób promienisty przez dostarczenie mu tablicy nazw komputerów za pomocą parametru `-computername`, można uzyskać przydatne informacje. Chociaż jeśli chcemy dowiedzieć się, jaki jest status konkretnych usług na konkretnym komputerze, to dane te nie będą już tak pomocne. Najlepiej wynik tego polecenia przekazać do polecenia `Format-Table` i wybrać własność `machineName`. Wartość własności `displayName` jest taka sama jak w kolumnie *Nazwa* w konsoli MMC. Omawiane polecenie i część jego wyniku pokazano poniżej:

```
PS C:\> Get-Service -ComputerName berlin,Windows 8 |
format-table name, status, machinename, displayName -AutoSize
```

Name	Status	MachineName	DisplayName
1-vmsrvc	Running	Windows 8	Virtual Machine Additions...
1-vmsrvc	Running	berlin	Virtual Machine Additions...
AeLookupSvc	Running	berlin	Application Experience
AeLookupSvc	Stopped	Windows 8	Application Experience
ALG	Stopped	berlin	Application Layer Gateway...
ALG	Stopped	Windows 8	Application Layer Gateway...
Appinfo	Stopped	berlin	Application Information
Appinfo	Stopped	Windows 8	Application Information
AppMgmt	Stopped	Windows 8	Application Management
AppMgmt	Stopped	berlin	Application Management
AudioEndpointBuilder	Stopped	berlin	Windows Audio Endpoint Bu...
AudioEndpointBuilder	Stopped	Windows 8	Windows Audio Endpoint Bu...
Audiosrv	Stopped	berlin	Windows Audio
Audiosrv	Stopped	Windows 8	Windows Audio
BFE	Running	Windows 8	Base Filtering Engine
BFE	Running	berlin	Base Filtering Engine
BITS	Stopped	berlin	Background Intelligent Tr...
BITS	Running	Windows 8	Background Intelligent Tr...
Browser	Running	Windows 8	Computer Browser
Browser	Stopped	berlin	Computer Browser

Jako że wartość własności `displayName` często jest długa, więc i często nie mieści się w kolumnie o szerokości 80 znaków. Jeśli wymienisz ją na początku kolejki wybranych własności w poleceniu `Format-Table`, kilka kolumn może nie zostać wyświetlonych, jak pokazano poniżej:

```
PS C:\> Get-Service -ComputerName berlin,Windows 8 | format-table name, displayname, status,
machinename -AutoSize
```

```
WARNING: 2 columns do not fit into the display and were removed.
```

Name	DisplayName
-----	-----
1-vmsrvc	Virtual Machine Additions Services Application
1-vmsrvc	Virtual Machine Additions Services Application
AeLookupSvc	Application Experience
AeLookupSvc	Application Experience

Jak widać, w kodzie tym w ogóle nie było sensu wybierać własności `machineName`, bo i tak nie jest widoczna. Rozwiązaniem tego problemu może być umieszczanie potencjalnie długiej własności na końcu listy argumentów polecenia. Dzięki temu konsola Windows PowerShell skróci wartość tej własności, zamiast zapełniać cały ekran danymi, które i tak pewnie udałoby się odgadnąć, dysponując tylko ich częścią.

Innym rozwiązaniem jest pozbycie się parametru `-autosize` polecenia `Format-Table` i użycie zamiast niego parametru `Wrap`. Parametr ten sprawia, że długie wartości są zawijane, a nie obcinane. W zależności od tego, czego potrzebujesz, może to być przydatne lub irytujące. Poniżej znajduje się przykład danych zwróconych dzięki użyciu parametru `-Wrap`:

```
PS C:\> Get-Service -ComputerName berlin,Windows 8 | format-table name, displayname, status, machinename -Wrap
```

Name	DisplayName	Status MachineName
-----	-----	-----
1-vmsrvc	Virtual Machine Additions Services Application	Running Windows 8
1-vmsrvc	Virtual Machine Additions Services Application	Running berlin
AeLookupSvc	Application Experience	Running berlin
AeLookupSvc	Application Experience	Stopped Windows 8

UWAGA

Może myślisz, że najlepszym rozwiązaniem jest użycie zarówno parametru `-autosize`, jak i `-Wrap`. Pozwoli to zmaksymalizować wykorzystanie przestrzeni ekranowej na prezentację zwróconych danych (dzięki parametrowi `-autosize`) i zawijać zbyt długie wiersze (dzięki parametrowi `-Wrap`). Tak się nie da, ale jeśli to zrobisz, program nie zwróci żadnego błędu. Konsola Windows PowerShell priorytetowo traktuje parametr `-autosize` i jeśli jest użyty, ignoruje parametr `-Wrap`. Kolejność ich wpisania nie ma przy tym znaczenia.

Wysyłanie zapytań do Active Directory

Większość administratorów sieci myśli, że aby wysłać zapytanie do Active Directory przy użyciu konsoli Windows PowerShell 1.0, trzeba napisać skrypt. Częściowo przekonanie to jest pozostałością po dniach świetności języka VBScript i odzwierciedla konieczność używania technologii ADO (ang. *ActiveX Data Object*) do wywoływania zapytań LDAP (ang. *Lightweight Directory Access Protocol*) do Active Directory. Wprawdzie można używać klasy `System.DirectoryServices.DirectorySearcher` w wierszu poleceń Windows PowerShell, ale nie jest to zbyt wygodne. Istnieją co prawda zewnętrzne polecenia cmdlet i dostawcy umożliwiający wykonywanie zapytań do Active Directory z wiersza poleceń, ale wielu administratorów sieci ma

słuszne opory przed instalowaniem na serwerach produkcyjnych oprogramowania społecznościowego bez wsparcia. Istnieje jeszcze drugie rozwiązanie w wierszu poleceń, polegające na użyciu narzędzia *DSQuery.exe*, ale mało komu przychodzi ono do głowy. Ale w konsoli Windows PowerShell 2.0 i jej nowszych wersjach sytuacja radykalnie się zmieniła. Za pomocą technik opisanych w tym podrozdziale informatyk może korzystać z dobrych rozwiązań do wykonywania zapytań do Active Directory w wierszu poleceń.

Zastosowanie akceleratora [ADSIEnumerator]

Jeśli ktoś chce wysyłać zapytania do Active Directory z poziomu wiersza poleceń konsoli Windows PowerShell, to ma kilka możliwości do wyboru. Jedną z nich jest użycie akceleratora typu [ADSIEnumerator], który jest skrótem nazwy klasy `System.DirectoryServices.DirectoryEnumerator`. Akcelerator ten, jak nietrudno się domyślić, jedynie pomaga zmniejszyć ilość wpisywanego tekstu. Oczywiście i tak należy mu podać odpowiedni konstruktor, aby rzeczywiście utworzyć egzemplarz klasy. Jeśli nie użyjesz akceleratora [ADSIEnumerator], musisz użyć polecenia `New-Object`, aby utworzyć obiekt. Polecenie to możesz wpisać w nawiasie, aby wymusić utworzenie obiektu, a następnie możesz wywołać metodę `FindAll` z obiektu `DirectoryEnumerator`. Otrzymana kolekcja obiektów `DirectoryEntry` zostaje potokowo przekazana do polecenia `Select-Object`, w którym zwracana jest własność `path`, jak pokazano poniżej:

```
PS C:\> (New-Object DirectoryServices.DirectoryEnumerator "ObjectClass=user").Find All() | Select path
Path
----
LDAP://CN=Administrator,CN=Users,DC=nwtraders,DC=com
LDAP://CN=Guest,CN=Users,DC=nwtraders,DC=com
LDAP://CN=BERLIN,OU=Domain Controllers,DC=nwtraders,DC=com
LDAP://CN=krbtgt,CN=Users,DC=nwtraders,DC=com
LDAP://CN=WINDOWS 8,CN=Computers,DC=nwtraders,DC=com
LDAP://CN=Windows 8Admin,OU=Students,DC=nwtraders,DC=com
List Truncated -
```

Aby użyć akceleratora typu [ADSIEnumerator], należy przekazać mu odpowiedni konstruktor, którym w wielu przypadkach jest filtr według składni LDAP. Składnia filtrów wyszukiwania LDAP jest zdefiniowana w dokumencie RFC 2254 i jest reprezentowana przez łańcuchy Unicode. Za pomocą filtrów wyszukiwania można definiować efektywne kryteria wyszukiwania. W tabeli 7.1 zamieszczono parę przykładów zastosowania tej składni.

TABELA 7.1. Przykłady filtrów wyszukiwania LDAP

Filtr wyszukiwania	Opis
<code>ObjectClass=Computer</code>	Wszystkie obiekty komputerów
<code>ObjectClass=OrganizationalUnit</code>	Wszystkie obiekty jednostek organizacyjnych
<code>ObjectClass=User</code>	Wszystkie obiekty użytkowników oraz wszystkie obiekty komputerów
<code>ObjectCategory=User</code>	Wszystkie obiekty użytkowników
<code>(&(ObjectCategory=User) (ObjectClass=Person))</code>	Wszystkie obiekty użytkowników

TABELA 7.1. Przykłady filtrów wyszukiwania LDAP — ciąg dalszy

Filtr wyszukiwania	Opis
L=Berlin	Wszystkie obiekty o lokalizacji Berlin
Name=*Berlin*	Wszystkie obiekty o nazwach zawierających słowo Berlin
(&(L=Berlin)(ObjectCategory=↪OrganizationalUnit))	Wszystkie jednostki organizacyjne o lokalizacji Berlin
(&(ObjectCategory=OrganizationalUnit)(Name=*Berlin*))	Wszystkie jednostki organizacyjne, których nazwa zawiera słowo Berlin
(&(ObjectCategory=OrganizationalUnit)(Name=*Berlin*)(!L=Berlin))	Wszystkie jednostki organizacyjne o nazwie zawierającej słowo Berlin, ale nie o lokalizacji Berlin
(&(ObjectCategory=OrganizationalUnit)(Name=*Berlin*)(!L=*))	Wszystkie jednostki organizacyjne o nazwie zawierającej słowo Berlin, ale niemające określonej lokalizacji
(&(ObjectCategory=OrganizationalUnit)((L=Berlin)(L=Charlotte)))	Wszystkie jednostki organizacyjne o lokalizacji Berlin lub Charlotte

Jak widać w przedstawionych przykładach, filtr wyszukiwania można zdefiniować na dwa sposoby. Pierwsza metoda polega na prostym przypisaniu filtra. Składa się on z atrybutu, operatora oraz wartości, jak poniżej:

```
PS C:\> ([ADSISearcher]"Name=Charlotte").FindAll() | Select Path
```

Path

```
LDAP://OU=Charlotte,DC=nwtraders,DC=com
```

Druga metoda użycia filtra wyszukiwania LDAP polega na łączeniu kilku filtrów. Najpierw wpisuje się operator, potem filtr A, następnie filtr B. Można łączyć wiele filtrów i operatorów, jak widać w przykładach przedstawionych w tabeli 7.1. Poniżej znajduje się przykład użycia takiego filtra złożonego:

```
PS C:\> ([ADSISearcher]"(|(Name=Charlotte)(Name=Atlanta))").FindAll() | Select Path
```

Path

```
LDAP://OU=Atlanta,DC=nwtraders,DC=com
```

```
LDAP://OU=Charlotte,DC=nwtraders,DC=com
```

W tabeli 7.2 znajduje się zestawienie operatorów, których można używać zarówno w prostych, jak i złożonych filtrach.

W tabeli 7.3 znajduje się zestawienie znaków specjalnych. Jeśli któryś z nich musi wystąpić w filtrze wyszukiwania w dosłownym znaczeniu, należy go zastąpić sekwencją specjalną.

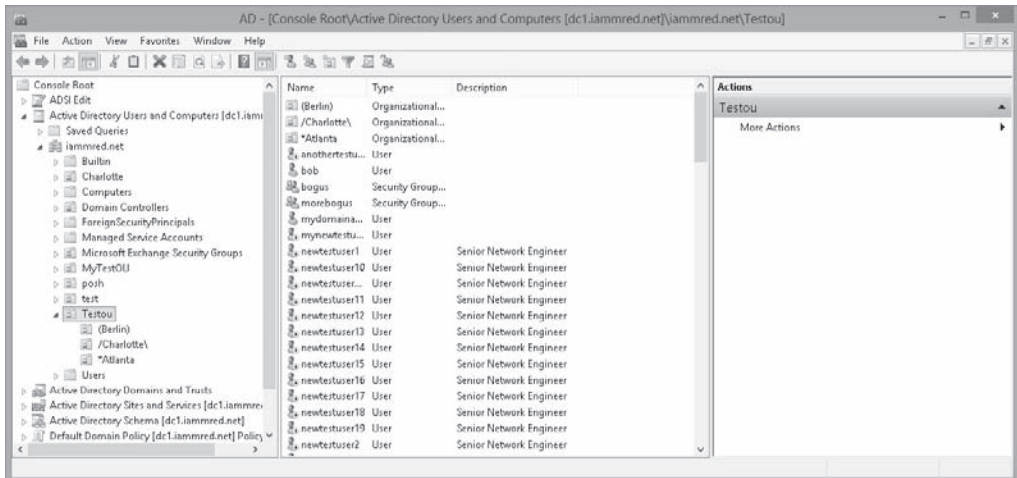
Jak widać na rysunku 7.8, znaki specjalne mogą być używane w nazwach jednostek organizacyjnych w Active Directory.

TABELA 7.2. Logiczne operatory filtrów wyszukiwania LDAP

Operator	Opis
=	Równość
~=	Równość w przybliżeniu
<=	Mniejszy niż lub równy w porządku leksykograficznym
>=	Większy niż lub równy w porządku leksykograficznym
&	Koniunkcja
	Alternatywa logiczna
!	Negacja logiczna

TABELA 7.3. Znaki specjalne filtrów wyszukiwania LDAP

Znak ASCII	Zastępcza sekwencja specjalna
*	\2a
(	\28
)	\29
\	\5c
NUL	\00
/	\2f



RYSUNEK 7.8. Nazwy jednostek organizacyjnych w Active Directory

Na rysunku tym widać jednostkę organizacyjną o nazwie **Atlanta*. Aby pobrać tę konkretną jednostkę, należy użyć znaku \2a, jak pokazano poniżej:

```
PS C:\> ([ADSISearcher]"name=\2aAtlanta").FindAll() | Select Path
```

Path

```
LDAP://OU=*Atlanta,DC=nwtraders,DC=com
```

Aby pobrać jednostkę organizacyjną o nazwie (*Berlin*), należy zgodnie z tabelą 7.3 użyć sekwencji specjalnych \28 i \29, jak pokazano poniżej:

```
PS C:\> ([ADSI searcher]"name=\28Berlin\29").FindAll() | Select Path
```

Path

```
LDAP://OU=(Berlin),DC=nwtraders,DC=com
```

Na rysunku 7.8 widać też jednostkę organizacyjną o nazwie */Charlotte*. Sekwencja specjalna zastępująca ukośnik / to \2f, natomiast lewy ukośnik ma sekwencję specjalną \5c. Aby więc pobrać jednostkę organizacyjną o nazwie */Charlotte* przy użyciu filtru wyszukiwania LDAP i akceleratora typu [ADSI searcher], należy użyć poniższego zapytania:

```
PS C:\> ([ADSI searcher]"name=\2fCharlotte\5c").FindAll() | Select Path
```

Path

```
LDAP://OU=\/Charlotte\\,DC=nwtraders,DC=com
```

Morał

Unikaj stosowania specjalnych znaków w nazwach jednostek organizacyjnych

Zasadniczo staram się nie używać znaków specjalnych w nazwach jednostek organizacyjnych, nazwach użytkowników, nazwach grup, nazwach komputerów itd. Podejrzewam, że nie wszystkie aplikacje potrafią je poprawnie zinterpretować, i zawsze obawiam się, czy nazwa zawierająca znak specjalny na pewno zadziała. Ponadto nawet mimo możliwości zastąpienia znaków specjalnych sekwencjami specjalnymi w wyszukiwaniu nigdy nie jest to intuicyjne i trzeba poświęcić dużo czasu na znalezienie sposobu na zastąpienie znaku sekwencją specjalną. Jeśli doda się do tego fakt, że problemy zwykle występują o godzinie 2 w nocy w niedziele (tak już jest, że wszystkie problemy z siecią zdarzają się o 2 w nocy w niedziele), gdy jest spore ryzyko, że zapomnisz użyć sekwencji specjalnej, to jesteś na najlepszej drodze do katastrofy. To, że coś można zrobić, nie znaczy, że jest to zalecane.

Znaki specjalne filtrów LDAP i odpowiadające im sekwencje specjalne są wymienione w tabeli 7.3.

Przy użyciu polecenia Invoke-Command można z łatwością wykorzystać akcelerator [ADSI searcher] do wysyłania zapytań do katalogu Active Directory niezaufałego lasu lub niezaufałej domeny. Wówczas należy podać pełną nazwę domeny komputera, ponieważ gdy używana jest tylko nazwa NetBIOS serwera, istnieje ryzyko, że nazwy nie będą w pełni rozwiązywane. Ponadto najlepiej jest przekazać dane poświadczające jako główną nazwę użytkownika (ang. *User Principal Name* — UPN). Po uruchomieniu polecenia zostaje wyświetlone okno dialogowe, w którym należy wpisać hasło. Polecenie to pokazano poniżej:

```
PS C:\> Invoke-Command -ComputerName Sydney.WoodBridgeBank.Com -Credential '
administrator@WoodBridgeBank.com -ScriptBlock {[ADSI searcher]"L=Berlin").FindAll()}}
PSComputerName      : sydney.woodbridgebank.com
RunspaceId          : 112f974a-00aa-417c-8a13-9033a49354bd
PSShowComputerName  : True
Path                : LDAP://OU=Berlin Bank,DC=woodbridgebank,DC=com
Properties           : {ou, dscorepropagationdata, whencreated, name...}
```

Posługiwanie się poleceniami Active Directory

Polecenia Active Directory są dostępne od systemu Windows 2008 R2. Znajdują się w module, więc najpierw należy je załadować za pomocą polecenia `Import-Module`. Oczywiście można po prostu wybrać ikonę Active Directory Windows PowerShell, która powoduje uruchomienie konsoli PowerShell z od razu załadowanymi poleceniami Active Directory. Bardzo dobrze, że polecenia Active Directory znajdują się w module, ponieważ dzięki temu za pomocą polecenia `Import-Module` można je dodać ze zdalnego komputera do sesji Windows PowerShell, w której ich brakuje. W tym celu należy wykonać następujące czynności:

1. Ustanów zdalną sesję z serwerem z systemem Windows 2008 R2.
2. Zaimportuj polecenia Active Directory za pomocą polecenia `Import-Module`.
3. Wykonaj zapytanie Active Directory.
4. Zamknij połączenie ze zdalną sesją.
5. Usuń zdalną sesję.

UWAGA

Używając polecenia `Remove-PSSession` z parametrem `-id`, należy pamiętać, że identyfikator sesji nie zawsze jest znany. Identyfikator pierwszej sesji to 1, a drugiej 2. Konsola Windows PowerShell prowadzi rejestr wszystkich sesji. Ale użytkownik może nie wiedzieć, który numer sesji jest aktualny. Dlatego najlepiej jest pobrać listę wszystkich sesji PowerShell za pomocą polecenia `Get-PSSession`. Ponadto mam zwyczaj usuwać odłączone sesje, których nie planuję używać w najbliższej przyszłości. W ten sposób zwalnię trochę zasobów.

Poniżej przedstawiono przykład usuwania nieużywanej sesji:

```
PS C:\> $ps = New-PSSession -ComputerName Sydney.WoodBridgeBank.Com -Credential
administrator@WoodBridgeBank.Com
PS C:\> Enter-PSSession $ps
[sydney.woodbridgebank.com]: PS C:\> Import-Module ActiveDirectory
[sydney.woodbridgebank.com]: PS C:\> Get-ADOrganizationalUnit -Filter "L -eq 'Berlin'"
Name                : Berlin Bank
Country              : DE
PostalCode           :
City                 : Berlin
ManagedBy           :
StreetAddress        :
State                : Berlin
ObjectGUID           : dde90f41-128c-4567-9822-00de5a4c96cc
ObjectClass          : organizationalUnit
DistinguishedName    : OU=Berlin Bank,DC=woodbridgebank,DC=com
[sydney.woodbridgebank.com]: PS C:\> Exit-PSSession
PS C:\> Get-PSSession

    Id Name                ComputerName    State    Configuration
    -- --
    1 Session1            sydney.woodb... Broken    Microsoft.PowerShell
PS C:\> Remove-PSSession -Id 1
```

Oprócz składni filtrów Active Directory, w których używa się operatorów Windows PowerShell i obsługiwane są konwersje typów wzbogaconych, można też używać składni filtrów LDAP opisanej w poprzednim podrozdziale. Aby użyć składni LDAP, należy zamiast parametru `-filter` użyć parametru `-LDAPFilter` oraz w cudzysłowie podać filtr wyszukiwania LDAP, jak pokazano poniżej:

```
PS C:\> Get-ADOrganizationalUnit -LDAPFilter '(L=Berlin)'  
Name           : Berlin Bank  
Country        : DE  
PostalCode     :  
City           : Berlin  
ManagedBy     :  
StreetAddress  :  
State          : Berlin  
ObjectGUID     : dde90f41-128c-4567-9822-00de5a4c96cc  
ObjectClass    : organizationalUnit  
DistinguishedName : OU=Berlin Bank,DC=woodbridgebank,DC=com
```

Po prostu używaj wiersza poleceń

Wiele poleceń można wykonać bezpośrednio w zwykłym wierszu poleceń przy użyciu starych narzędzi. Nie ma w tym nic złego i polecenia te są też obsługiwane przez konsolę Windows PowerShell. Wskazówką, że stare narzędzia wiersza poleceń są obsługiwane przez Windows PowerShell, powinno być to, że można je wyszukiwać za pomocą polecenia cmdlet `Get-Command`. Do wyszukiwania plików wykonywalnych za pomocą tego polecenia można używać symboli wieloznacznych, jak pokazano poniżej:

```
PS C:\> Get-Command ds*  
CommandType Name Definition  
-----  
Application ds16gt.dll C:\Windows\system32\ds16gt.dll  
Application ds32gt.dll C:\Windows\system32\ds32gt.dll  
Application dsa.msc C:\Windows\system32\dsa.msc  
Application dsacls.exe C:\Windows\system32\dsacls.exe  
Application dsadd.exe C:\Windows\system32\dsadd.exe  
Application dsadmin.dll C:\Windows\system32\dsadmin.dll  
Application dsauth.dll C:\Windows\system32\dsauth.dll  
Application dsdbutil.exe C:\Windows\system32\dsdbutil...  
Application dsdmo.dll C:\Windows\system32\dsdmo.dll  
Application dsget.exe C:\Windows\system32\dsget.exe  
Application dskquota.dll C:\Windows\system32\dskquota...  
Application dskquoui.dll C:\Windows\system32\dskquoui...  
Application dsmsgmt.exe C:\Windows\system32\dsmsgmt.exe  
Application dsmod.exe C:\Windows\system32\dsmod.exe  
Application dsmove.exe C:\Windows\system32\dsmove.exe  
Application dsound.dll C:\Windows\system32\dsound.dll  
Application dsprop.dll C:\Windows\system32\dsprop.dll  
Application dsprov.dll C:\Windows\System32\Wbem\dsp...  
Application dsprov.mof C:\Windows\System32\Wbem\dsp...  
Application dsquery.dll C:\Windows\system32\dsquery.dll  
Application dsquery.exe C:\Windows\system32\dsquery.exe  
Application dsrm.exe C:\Windows\system32\dsrm.exe  
Application dssec.dat C:\Windows\system32\dssec.dat  
Application dssec.dll C:\Windows\system32\dssec.dll  
Application dsenh.dll C:\Windows\system32\dsenh.dll  
Application dsite.msc C:\Windows\system32\dsite.msc  
Application dsuixt.dll C:\Windows\system32\dsuixt.dll  
Application dsuiwiz.dll C:\Windows\system32\dsuiwiz.dll  
Application dswave.dll C:\Windows\system32\dswave.dll
```

Powyższe polecenie zwraca wszystkie poprawne polecenia Windows PowerShell, wliczając funkcje, polecenia cmdlet oraz pliki wykonywalne. Jeśli chcesz znaleźć tylko narzędzia wiersza poleceń, użyj parametru `commandtype`, jak pokazano poniżej:

```
PS C:\> Get-Command -Name ds* -CommandType application
```

CommandType	Name	Definition
-----	----	-----
Application	ds16gt.dll	C:\Windows\system32\ds16gt.dll
Application	ds32gt.dll	C:\Windows\system32\ds32gt.dll
Application	dsa.msc	C:\Windows\system32\dsa.msc
Application	dsac1s.exe	C:\Windows\system32\dsac1s.exe
Application	dsadd.exe	C:\Windows\system32\dsadd.exe
Application	dsadmin.dll	C:\Windows\system32\dsadmin.dll
Application	dsauth.dll	C:\Windows\system32\dsauth.dll
Application	dsdbutil.exe	C:\Windows\system32\dsdbutil...
Application	dsdmo.dll	C:\Windows\system32\dsdmo.dll
Application	dsget.exe	C:\Windows\system32\dsget.exe
Application	dskquota.dll	C:\Windows\system32\dskquota...
Application	dskquoui.dll	C:\Windows\system32\dskquoui...
Application	dsmgmt.exe	C:\Windows\system32\dsmgmt.exe
Application	dsmod.exe	C:\Windows\system32\dsmod.exe
Application	dsmove.exe	C:\Windows\system32\dsmove.exe
Application	dsound.dll	C:\Windows\system32\dsound.dll
Application	dsprop.dll	C:\Windows\system32\dsprop.dll
Application	dsprov.dll	C:\Windows\System32\Wbem\dsp...
Application	dsprov.mof	C:\Windows\System32\Wbem\dsp...
Application	dsquery.dll	C:\Windows\system32\dsquery.dll
Application	dsquery.exe	C:\Windows\system32\dsquery.exe
Application	dsrm.exe	C:\Windows\system32\dsrm.exe
Application	dssec.dat	C:\Windows\system32\dssec.dat
Application	dssec.dll	C:\Windows\system32\dssec.dll
Application	dsenh.dll	C:\Windows\system32\dsenh.dll
Application	dssite.msc	C:\Windows\system32\dssite.msc
Application	dsuixt.dll	C:\Windows\system32\dsuixt.dll
Application	dsuiwiz.dll	C:\Windows\system32\dsuiwiz.dll
Application	dswave.dll	C:\Windows\system32\dswave.dll

Łatwość obsługi i elastyczność konsoli Windows PowerShell przyczyniły się do nawrotu zainteresowania programami wiersza poleceń. Przykładem może być program *DSQuery.exe*, za pomocą którego użytkownik może szybko wysłać zapytanie do Active Directory. Biorąc pod uwagę, że w systemie Windows Server 2008 R2 dodano akcelerator typu [ADSI Searcher] i różne polecenia cmdlet do obsługi Active Directory, możesz się zastanawiać, po co w ogóle ktoś miałby używać takich narzędzi jak *DSQuery.exe*. Poniżej znajduje się polecenie *DSQuery.exe* zwracające listę jednostek organizacyjnych w domenie:

```
PS C:\> dsquery ou
"OU=Domain Controllers,DC=nwtraders,DC=com"
"OU=Students,DC=nwtraders,DC=com"
"OU=ManagedComputers,DC=nwtraders,DC=com"
"OU=TestOU,DC=nwtraders,DC=com"
```

Poniżej znajduje się polecenie zwracające listę jednostek organizacyjnych w składni opartej na użyciu akceleratora [ADSI Searcher]:

```
PS C:\> ([ADSI::Searcher]"objectClass=OrganizationalUnit").findall() | select-object -property path

Path
----
LDAP://OU=Domain Controllers,DC=nwtraders,DC=com
LDAP://OU=Students,DC=nwtraders,DC=com
LDAP://OU=ManagedComputers,DC=nwtraders,DC=com
LDAP://OU=TestOU,DC=nwtraders,DC=com
```

Składnia oparta na poleceniu `Get-ADOrganizationalUnit`, dostępnym w module Active Directory od systemu Windows Server 2008 R2, jest nieco prostsza. Podczas pracy w wierszu poleceń Windows PowerShell Active Directory nie zawsze trzeba podawać nazwy parametrów. Można też używać aliasów (np. `Select` zamiast `Select-Object`). Dzięki aliasom polecenia są krótsze, ale trudniejsze do zmodyfikowania. Poniżej znajduje się przykład użycia polecenia `Get-ADOrganizationalUnit`:

```
PS C:\> Get-ADOrganizationalUnit -Filter "name -like '*'" | Select DistinguishedName

DistinguishedName
-----
OU=Domain Controllers,DC=woodbridgebank,DC=com
OU=Test1,DC=woodbridgebank,DC=com
```

Jeśli dążysz do maksymalnego skrócenia składni, to najlepszym rozwiązaniem będzie użycie programu *DSQuery.exe*. Ale jeśli masz jeszcze inne wymagania, to lepsze mogą być inne rozwiązania. Program *DSQuery.exe* zwraca łańcuch, podczas gdy akcelerator `[ADSI::Searcher]` zwraca obiekt `DirectoryEntry`. Z kolei polecenie `Get-ADOrganizationalUnit` zwraca obiekt `Microsoft.ActiveDirectory.Management.ADOrganizationalUnit`. Każda metoda ma wady i zalety, więc wybór jednej z nich zależy od konkretnych potrzeb.

UWAGA

Uważam, że dobrym zwyczajem jest wpisywanie pełnych nazw parametrów nawet podczas pracy w wierszu poleceń w konsoli Windows PowerShell. Choć parametry często można definiować według ich pozycji, to trzeba pamiętać, który jest domyślny, oraz znać kolejność parametrów, jeśli jest ich więcej niż jeden. Często żądane informacje udaje mi się zdobyć dopiero po paru próbach. Po zmianie domyślnego parametru na kilka parametrów modyfikujących, jeśli nie używa się nazw parametrów, składnia ulega zmianie, jak pokazano poniżej:

```
PS C:\> Get-Command ds* application
Get-Command : The command could not be retrieved because the ArgumentList parameter can be
specified only when retrieving a single cmdlet or script.
At line:1 char:12
+ Get-Command <<<< ds* application
+ CategoryInfo          : InvalidArgument: (ds16gt.dLL:ApplicationInfo)
[Get-Command], PSArgumentException
+ FullyQualifiedErrorId :
CommandArgsOnlyForSingleCmdlet,Microsoft.PowerShell.Commands.GetCommandCommand
```

Oprócz kwestii związanej z typem zwrotnym narzędzie *DSQuery.exe* jest obciążone jeszcze pewnymi innymi problemami. Przede wszystkim poświęcono w nim możliwości na rzecz prostoty, co oznacza, że w zapytaniach można używać tylko paru atrybutów. Aby na przykład znaleźć

wszystkie jednostki organizacyjne w Active Directory zawierające nazwę Berlin, można użyć poniższego polecenia:

```
dsquery ou -name *berlin*
```

A żeby znaleźć wszystkie jednostki organizacyjne w Active Directory, których atrybut lokalizacji jest ustawiony na Berlin, należy użyć poleceń cmdlet Active Directory lub akceleratora [ADSISeacher]. Jeśli ktoś dobrze wie, jakie są możliwości narzędzia *DSQuery.exe*, to nie ma nic złego w tym, że go używa. Zwrócone przezeń wyniki można nawet przekazać potokowo do innych narzędzi, np. *DSMove.exe*. Program *DSMove.exe* przenosi obiekt do innej lokalizacji w Active Directory, program *DSMode.exe* służy do zmieniania wartości atrybutów, a *DSrm.exe* służy do usuwania obiektów z Active Directory.

Obliczanie korzyści z użycia skryptu

Jeśli podliczy się czas potrzebny na napisanie, przetestowanie i umieszczenie skryptu w systemie kontroli wersji, może się okazać, że cały proces jest dość czasochłonny. Dlatego też informatycy powinni oszacować potencjalne korzyści z napisania skryptu, zanim zaczną go pisać. Jak już wspomniałem w tym rozdziale, wiele tradycyjnych powodów do pisania skryptów Windows PowerShell straciło już ważność. Nie znaczy to, że nigdy nie trzeba pisać skryptów, tylko że wiele skomplikowanych zadań można wykonać przy użyciu prostych poleceń. Jeśli zastanawiasz się, czy pisać skrypt, czy nie, w tym podrozdziale znajdziesz parę wskazówek.

Zapiski praktyka

Jason A. Yoder, MCT

Prezes firmy MCTExpert, Inc.

Często polecam używanie konsoli Windows PowerShell informatykom, którzy są przyzwyczajeni do tego, że traktuje się ich jako koszt, a nie wartościowe aktywa firmy. Ale nie musi tak być. Odkąd istnieje konsola Windows PowerShell, a jest już czwarta wersja, trudno znaleźć jakiegokolwiek uzasadnienie dla ręcznego wykonywania jakichkolwiek czynności administracyjnych na serwerach firmy Microsoft. W istocie konsola ta umożliwia informatykom pokazanie, że są cennymi pracownikami, i udowodnienie, że warto inwestować w ich pracę.

Wyobraź sobie taką sytuację: za każdym razem, gdy ktoś w firmie uruchamia napisany przez Ciebie skrypt, program ten wysyła pewne ważne informacje do znajdującego się na centralnym serwerze pliku CSV. Zapisywane jest, który skrypt został uruchomiony, kto go uruchomił oraz ile czasu dzięki temu zaoszczędzono. Przy użyciu tych danych informatyk może udowodnić, jak wiele udało się zyskać dzięki niewielkiej inwestycji w napisanie paru skryptów. W niektórych przypadkach można nawet wykazać, że zaoszczędzenie wielu godzin dzięki użyciu konsoli Windows PowerShell pozwoliło odłożyć konieczność zatrudnienia dodatkowych pracowników.

Kiedys pomagałem pewnemu klientowi, który musiał wyznaczyć jednego pracownika do robienia kopii zapasowej i czyszczenia dzienników zabezpieczeń co trzy godziny każdego dnia na wielu zdalnych serwerach. Dzięki poświęceniu trzech godzin na napisanie i przetestowanie skryptu klient ten poświęca na to samo zadanie mniej niż godzinę rocznie — kiedyś potrzebował 1095 godzin. Nie trzeba być geniuszem finansowym, aby dostrzec, jak dużo można potencjalnie zaoszczędzić. Pamiętam też pewnego użytkownika, który co tydzień musiał wykonywać pewne zadanie związane z używaniem Active Directory i Excela. Zajmowało mu to dwie godziny. Dzięki użyciu konsoli Windows PowerShell zadanie jest wykonywane automatycznie, a osoba, o której piszę, ma gotowy produkt, gdy przychodzi do pracy w poniedziałek — oszczędza 104 godziny pracy.

Pytanie, jakie należy sobie zadać, nie brzmi: „Co mogę zrobić przy użyciu konsoli Windows PowerShell?”, tylko: „Co **chcę** zrobić przy użyciu konsoli Windows PowerShell?”. Gdy odpowiesz sobie na pytanie i wprowadzisz swoją wizję w życie, nie zapomnij oszacować efektów. Może staniesz się w firmie informatyczną gwiazdą i nowym cennym pracownikiem.

Powtarzalność

Jeśli jakieś zadanie trzeba powtórzyć wiele razy, to jest to dobra okazja do napisania skryptu (choć oczywiście nie zawsze). Informatycy często potrzebują informacji o stanie różnych usług, który bez problemu można sprawdzić za pomocą polecenia `Get-Service`. Aby sprawdzić status wybranego procesu, należy użyć parametru `Name`, jak pokazano poniżej:

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

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
661	9	42616	46024	202	3.61	880	powershell

Jeśli trzeba odczytać najnowszy wpis w dzienniku aplikacji, należy użyć dwóch parametrów, jak pokazano poniżej:

```
PS C:\> Get-WinEvent -LogName application -MaxEvents 1
```

TimeCreated	ProviderName	Id	Message
1/26/2009 10:47:...	VSS	8193	Volume Shadow Co...

W tych przypadkach pisanie skryptu nie ma sensu, ponieważ składnia poleceń jest prosta, łatwa w użyciu i bez trudu można ją sobie przypomnieć, jeśli się zapomni. Wystarczy w razie potrzeby użyć polecenia `Get-Help`.

Jeśli jakąś czynność wykonuje się rutynowo na grupie komputerów, to warto zastanowić się nad napisaniem skryptu. Załóżmy, że musimy sprawdzać poziom fragmentacji na kilku komputerach. Pewnie uda się znaleźć sposób na uruchomienie polecenia bezpośrednio z konsoli Windows PowerShell, ale następnym razem, gdy będzie trzeba to zrobić, znowu spędzisz kilkadziesiąt minut na przypominaniu sobie składni. W takim przypadku lepiej napisać skrypt, np. o nazwie *DefragAnalysisReport.ps1*, co powinno zająć nie więcej niż godzinę. W skrypcie tym można użyć klasy `WMI Win32_Volume`, wywołać metodę `DefragAnalysis` dla każdego dysku komputera i zapisać wyniki w pliku tekstowym.

W skrypcie *DefragAnalysisReport.ps1* najpierw należy utworzyć tablicę nazw komputerów i przypisać ją do zmiennej `$arycomputer`. Można to zrobić zarówno przez wpisanie wartości bezpośrednio w kodzie, jak w tym przykładzie, lub odczytanie wartości z pliku tekstowego za pomocą polecenia `Get-Content`. Następnie należy przypisać wartość ścieżki wyjściowej do zapisania raportu. Ścieżka ta wskazuje folder istniejący na komputerze, na którym zostanie uruchomiony skrypt. Folderu tego nie musi być na komputerze docelowym, ponieważ wszystkie raporty będą przechowywane lokalnie. Poniżej znajduje się opisywana część kodu:

```
$arycomputer = "Windows 8","Berlin"
$FilePath = "C:\fso"
```

Teraz za pomocą instrukcji `Foreach` trzeba przejrzeć tablicę nazw komputerów zapisaną w zmiennej `$arycomputer`, jak pokazano poniżej:

```
Foreach($Computer in $aryComputer)
{
```

Do klasy `WMI Win32_Volume` można wysłać zapytania za pomocą polecenia `Get-WmiObject`. Klasa ta jest dostępna od systemu Windows 2003. Jeśli planowane jest uruchamianie skryptu w starszych systemach, dobrym zwyczajem jest dodanie mechanizmu obsługi błędów wykrywającego wersję systemu operacyjnego i w razie potrzeby elegancko przechodzącego do następnego komputera. Opis tej techniki znajduje się w rozdziale 6. Poniżej przedstawiono opisywane zapytanie:

```
Get-WmiObject -Class win32_volume -Filter "DriveType = 3" '
-ComputerName $computer |
```

Wyniki zapytania WMI są przekazywane do polecenia `ForEach-Object`. Technika polegająca na przekazywaniu danych przez potok jest nieco bardziej efektywna niż zapisywanie wyników w zmiennej i iterowanie po nich, ponieważ przetwarzanie rozpoczyna się od razu po otrzymaniu pierwszego obiektu. Pierwszą czynnością w poleceniu `ForEach-Object` jest wydrukowanie za pomocą parametru `-Begin` wiadomości informującej o tym, który komputer jest aktualnie sprawdzany, jak pokazano poniżej:

```
ForEach-Object '
-Begin { "Sprawdzanie komputera $computer" } '
```

Rzeczywistą analizę defragmentacji, która ma miejsce raz dla każdego dysku, można wykonać w bloku `Process`. Metoda `DefragAnalysis` jest wywoływana dla obiektu znajdującego się aktualnie w potoku. Zmienna `$_` to zmienna automatyczna odnosząca się do tego obiektu. Metoda `DefragAnalysis` zwraca zarówno raport o błędzie, jak i egzemplarz klasy `WMI Win32_DefragAnalysis`. Obie te informacje są zapisywane w zmiennej `$rtn`, jak pokazano poniżej:

```
-Process {
    "Sprawdzanie poziomu fragmentacji dysku $($_.name). Czekaj..."
    $RTN = $_.DefragAnalysis()
```

W celu utworzenia raportu o poziomie defragmentacji można użyć przekierowania. Pojedynczy znak nawiasu trójkątnego skierowany w prawo (`>`) nadpisuje poprzednie raporty. Jako że istnieje duża szansa, że serwer będzie zawierał więcej niż jeden dysk, lepiej użyć podwójnego nawiasu (`>>`). Inną możliwością jest użycie polecenia `Out-File`, którego zaletą jest to, że pozwala

określić sposób kodowania pliku. Ponadto polecenie to jest bardziej czytelne niż strzałki przekierowania. Dlatego z reguły używam polecenia `Out-File`. Poniżej znajduje się sekcja nagłówka raportu:

```
"Raport nt. defragmentacji dla komputera $computer" >> "$FilePath\Defrag$computer.txt"
"Raport dla dysku $(_.Name)" >> "$FilePath\Defrag$computer.txt"
"Data sporządzenia raportu: $(Get-Date)" >> "$FilePath\Defrag$computer.txt"
"-----" >> "$FilePath\Defrag$computer.txt"
```

Jedną z wielkich zalet konsoli Windows PowerShell jest sposób, w jaki automatycznie wyświetla własności i wartości obiektów. Aby w języku VBScript wydrukować wartość każdej własności, trzeba napisać kilkanaście wierszy kodu. Jak widać tutaj, obiekt zarządzania `Win32_DefragAnalysis`, który jest zapisany we własności `DefragAnalysis`, zostaje przekazany do polecenia `Format-List` w celu usunięcia własności systemowych klasy WMI. Nazwy wszystkich własności systemowych zaczynają się od dwóch znaków podkreślenia (`_`), dzięki czemu można je łatwo wyeliminować za pomocą wyrażenia regularnego wyszukującego tylko własności o nazwach zaczynających się od liter, po których występuje dowolna liczba innych znaków. Otrzymana lista własności z wartościami zostaje przekazana do pliku znajdującego się w lokalizacji określonej przez własność `$filePath`, jak pokazano poniżej:

```
$RTN.DefragAnalysis |
Format-List -Property [a-z]* >> "$FilePath\Defrag$computer.txt"
}
```

Na koniec za pomocą parametru `End` można wydrukować informację, że testowanie na danym komputerze zostało zakończone, jak pokazano poniżej:

```
-END { "Zakończono sprawdzanie komputera $computer" }
} #end foreach computer
```

Poniżej znajduje się cały skrypt *DefragAnalysisReport.ps1*.

DefragAnalysisReport.ps1

```
$arycomputer = "Windows 8","Berlin"
$FilePath = "C:\fso"
Foreach($Computer in $aryComputer)
{
  Get-WmiObject -Class win32_volume -Filter "DriveType = 3" `
    -ComputerName $computer |
  ForEach-Object {
    -Begin { "Sprawdzanie komputera $computer" } `
    -Process {
      "Sprawdzanie poziomu defragmentacji dysku $(_.name). Czekaj..."
      $RTN = $_.DefragAnalysis()
      "Raport nt. defragmentacji dla komputera $computer" >> "$FilePath\Defrag$computer.txt"
      "Raport dla dysku $(_.Name)" >> "$FilePath\Defrag$computer.txt"
      "Data sporządzenia raportu: $(Get-Date)" >> "$FilePath\Defrag$computer.txt"
      "-----" >> "$FilePath\Defrag$computer.txt"

      $RTN.DefragAnalysis |
      Format-List -Property [a-z]* >> "$FilePath\Defrag$computer.txt"
    }
  }
  -END { "Completed testing $computer" }
} #end foreach computer
```

Wiedza tajemna

Stefan Stranger, Senior Premier Field Engineer

Microsoft Corporation

Program System Center Operations Manager zaczął odbierać pakiety zbiorcze aktualizacji przy użyciu usługi Windows Update. Jako że nie wszystkie moje komputery demonstracyjne i testowe są połączone z internetem, potrzebowałem sposobu na odszukanie łączy używanych przez program Operations Manager do pobierania danych z tej usługi.

Usługa Windows Update używa pliku *.cab* znajdującego się pod adresem <http://go.microsoft.com/fwlink/?LinkId=76054>. Jest w nim plik *Package.xml* zawierający łączy, przy użyciu których Operations Manager (i inne produkty) pobiera pliki aktualizacyjne.

Utworzyłem skrypt pobierający ten plik *.cab*, wypakowujący z niego potrzebne pliki i wyszukujący w pliku *.xml* łączy do plików aktualizacji, co pozwoliło mi na aktualizowanie moich środowisk testowych bez podłączania ich do internetu.

Agent aktualizacji pobiera te informacje z archiwum *wsusscn2.cab* udostępnianego do pobrania przez firmę Microsoft pod stałym adresem URL <http://go.microsoft.com/fwlink/?linkid=76054>. Plik ten pobiera także program Microsoft Baseline Security Analyzer (MBSA), który przy jego użyciu sprawdza, czy system ma zainstalowane wszystkie poprawki.

Archiwum zawiera plik katalogowy o nazwie *package.xml*, w którym firma Microsoft indeksuje wszystkie poprawki dotyczące bezpieczeństwa (wraz z zależnościami) dla wszystkich swoich systemów operacyjnych. W pliku tym znajdują się też adresy URL do plików do pobrania, za pomocą których można pobrać wybrane poprawki bezpośrednio z serwerów Microsoftu.

Najpierw należy za pomocą konsoli Windows PowerShell pobrać plik *wsusscn2.cab*:

```
# Pobiera plik .cab Windows Update, aby znaleźć w nim łączy do aktualizacji.
```

```
$download = "http://go.microsoft.com/fwlink/?LinkId=76054"
```

```
# Pobiera plik .cab.
```

```
Start-BitsTransfer -Source "http://go.microsoft.com/fwlink/?LinkId=76054" -Destination  
"$env:temp\wsusscn2.cab"
```

Kolejną czynnością jest wydobycie pliku *package.xml* z właśnie pobranego pliku *wsusscn2.cab*.

Użyłem funkcji COM Expand-Cab:

```
Function Expand-Cab ($SourceFile, $TargetFolder, $Item)  
{  
  
    $ShellObject = new-object -com shell.application  
    # Plik ZIP do rozpakowania:  
    $zipfolder = $ShellObject.namespace($sourceFile) # miejsce przechowywania pliku ZIP  
    $item = $zipfolder.parse($Item) # element w pliku ZIP  
    $targetfolder = $ShellObject.namespace("$targetFolder")  
    $targetfolder.copyhere($item)  
}
```

Za pomocą tej funkcji można wypakować zawartość pliku *wsussnc2.cab* do folderu tymczasowego użytkownika:

```
# Wypakowuje plik Package.cab z pliku WSUSsnc2.cab
```

```
Expand-Cab -SourceFile "$env:temp\wsussnc2.cab" -TargetFolder "$env:temp" -Item "Package.cab"
```

Po rozpakowaniu pliku *wsussnc2.cab* mamy plik *Package.cab*, który też trzeba rozpakować, aby wydobyć z niego plik *Package.xml*:

```
# Wypakowuje plik Package.xml z pliku Package.cab.
```

```
Expand-Cab -SourceFile "$env:temp\Package.cab" -TargetFolder "$env:temp" -Item "Package.xml"
```

Ostatnią czynnością jest odczytanie zawartości pliku *Package.xml* za pomocą konsoli Windows PowerShell:

```
[xml]$wsusupdate = get-content -path "$env:temp\package.xml"
```

```
$urls = $wsusupdate.OfflineSyncPackage.FileLocations.FileLocation
```

Stosując filtrowanie przy użyciu nazw artykułów Bazy wiedzy, można znaleźć pliki pakietu zbiorczego aktualizacji dla każdego takiego artykułu:

```
$KBArticle = "KB2750631"
```

```
$urls | ? { $_.Url -like "$KBArticle*" } | ft -Property Url -wrap
```

Teraz pobieramy pliki *.cab* dla wybranego artykułu Bazy wiedzy za pomocą polecenia

```
Start-BitsTransfer.
```

```
$urls | ? { $_.Url -like "*KB2750631*" } | select @{L="Source";E={$_.Url}},
```

```
@{L="Destination";E={"$env:temp\"+($_.Url) -split "/" }[(($_.Url) -split "/").count -1]}} |
```

```
Start-BitsTransfer
```

Możliwość opisanania w dokumentacji

Skrypt zapewnia, że pewne czynności zawsze będą wykonywane w ten sam sposób. Jest to bardzo ważne przy wykonywaniu skomplikowanych zadań konfiguracyjnych oraz przy dokonywaniu prostych zmian w rejestrze. Skrypt dokładnie dokumentuje, co wydarzyło się podczas zmieniania konfiguracji. Jeśli później zostanie w niej znaleziony błąd, można sprawdzić dokumentację wykonanych przez skrypt poleceń i go odpowiednio zmodyfikować, aby cofnąć wprowadzone zmiany.

W poniższym przykładzie w gałęzi rejestru *HKEY_CURRENT_USER* tworzony jest nowy klucz o nazwie *Scripting* oraz kolejny o nazwie *Logon*. Po utworzeniu tych kluczy zostaje utworzona własność o nazwie *ScriptName* i wartości *temp*. Powstałe w ten sposób klucze rejestru widać na rysunku 7.9, a kod użyty do ich utworzenia pokazano poniżej:

```
PS C:\> New-Item -Path HKCU:\Scripting\Logon -Force
```

```
Hive: HKEY_CURRENT_USER\Scripting
```

Name	Property
----	-----
Logon	

```
PS C:\> New-ItemProperty -Path HKCU:\Scripting\Logon -Name ScriptName -Value "Temp"
```

```
ScriptName : Temp
```

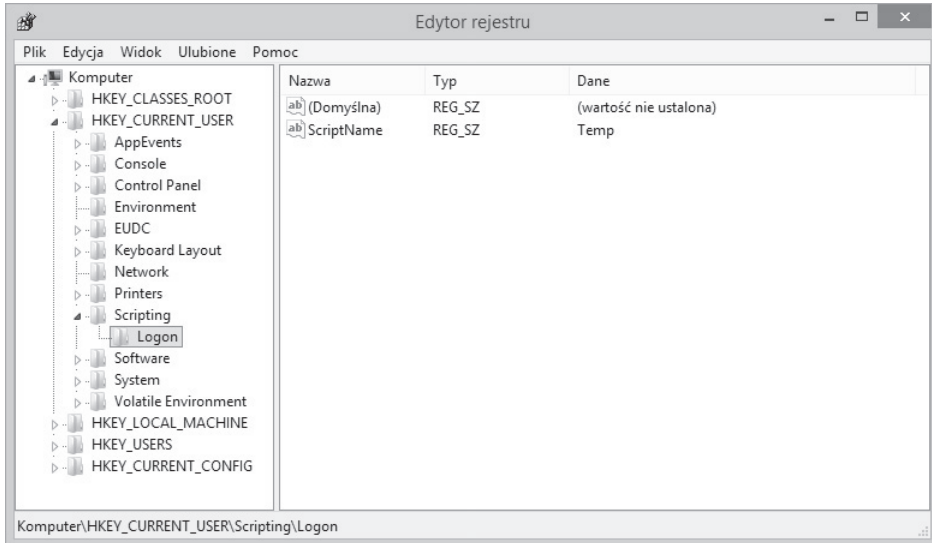
```
PSPath : Microsoft.PowerShell.Core\Registry::HKEY_CURRENT_USER\Scripting\Logon
```

```
PSParentPath : Microsoft.PowerShell.Core\Registry::HKEY_CURRENT_USER\Scripting
```

```
PSChildName : Logon
```

```
PSDrive : HKCU
```

```
PSProvider : Microsoft.PowerShell.Core\Registry
```



RYSUNEK 7.9. Własność ScriptName w kluczu rejestru Logon

Jeśli podczas tworzenia kluczy rejestru i ich wartości wystąpi problem, należy uruchomić Edytor rejestru lub wpisać w konsoli Windows PowerShell parę poleceń. Skrypt z zasady łatwiej się modyfikuje, ponieważ widać w całości wykonywany kod. Polecenia, które wpisano wcześniej w konsoli Windows PowerShell, są pokazane w skrypcie *CreateScriptingRegistryKey.ps1*.

CreateScriptingRegistryKey.ps1

```
New-Item -Path HKCU:\Scripting\Logon -Value "Temp" -Force
New-ItemProperty -Path HKCU:\Scripting\Logon -Name ScriptName -Value "Temp"
```

Jeśli problem wystąpił z wcześniejszym poleceniem, to na podstawie pierwszego skryptu bez problemu można utworzyć nowy skrypt cofający zmiany, jak widać w skrypcie *DeleteScriptingRegistryKey.ps1*. Drugi wiersz tego skryptu jest wyłączony za pomocą komentarza, a w pierwszym zmieniono polecenie `New-Item` na `Remove-Item`. Parametr `-force` zmieniono na `Recurse`, a parametr `value` w poleceniu `Remove-Item` jest niepotrzebny. Zawartość zmodyfikowanego skryptu *DeleteScriptingRegistryKey.ps1* pokazano poniżej:

DeleteScriptingRegistryKey.ps1

```
Remove-Item -Path HKCU:\Scripting -Recurse
#New-ItemProperty -Path HKCU:\Scripting\Logon -Name ScriptName -Value "Temp"
```

Zdolność do adaptacji

W zależności od struktury skryptu można go wykorzystać do wykonywania różnych innych zadań. Jeśli skrypt ma budowę modułową oraz użyto w nim funkcji i argumentów wiersza poleceń, to może być używany do wykonywania rozmaitych czynności. Funkcje można zaimportować przez dołączenie skryptu do innego skryptu. Sam skrypt można też przekształcić w moduł, który później można zaimportować do sesji za pomocą polecenia `Import-Module`.

Przykładem skryptu o budowie modułowej jest skrypt *SaveWmiInformationAsDocument.ps1*. Jego najważniejsze składniki są funkcjami, które można łatwo wykorzystać w innych skryptach.

Wielokrotne wykorzystanie kodu

Możliwość przystosowywania funkcji z jednego skryptu do użycia w innych skryptach często uzasadnia koszt finansowy i czas, jaki należy włożyć w napisanie tego skryptu. Niemniej jednak możliwość wielokrotnego wykorzystania kodu nie powinna być podstawowym celem programisty. Pisanie skryptów o modułowej budowie trwa znacznie dłużej niż zwykłych. Ponadto inwestowanie czasu w możliwość ponownego użycia kodu w bliżej nieokreślonej przyszłości nie zawsze jest dobrym rozwiązaniem. Oczywiście budowanie kodu modułowego jest bardzo dobrym podejściem do programowania, które pozwala uzyskać czytelny i łatwy w modyfikacji kod. Warto dążyć do tych celów projektowych, ale sam potencjał ponownego użycia kodu w przyszłości nie powinien być ostatecznym argumentem za.

Pierwsza funkcja w skrypcie *SaveWmiInformationAsDocument.ps1* nazywa się `CreateWordDoc`. Tworzy ona egzemplarz klasy `Word.Application`, który zapisuje w zmiennej skryptowej `$word`. Następnie funkcja ta sprawia, że aplikacja Microsoft Office Word staje się widoczna, i dodaje dokument do kolekcji dokumentów, jak pokazano poniżej:

```
Function CreateWordDoc()
{
    $script:word = New-Object -ComObject word.application
    $word.visible = $true
    $Script:doc = $word.documents.add()
} #end CreateWordDoc
```

Kolejna funkcja nazywa się `CreateSelection` i przyjmuje łańcuch, który powinien zostać użyty jako nagłówek dokumentu Office Word. Do utworzenia zaznaczenia w programie Word potrzebny jest egzemplarz klasy `Word.Application`. Jako że zmienna `$word` należy do zakresu skryptowego, jest dostępna także w funkcji `CreateSelection`. Obiekt `selection` zostaje utworzony przez wysłanie zapytania do własności `selection`. Do wpisania nagłówka do dokumentu Word użyto metody `TypeText`. Następnie tworzony jest pusty akapit i funkcja kończy działanie.

```
Function CreateSelection($Heading)
{
    $script:selection = $word.selection
    $selection.typeText($Heading)
    $selection.TypeParagraph()
} #end CreateSelection
```

Funkcja `GetWmiData` odpytuje klasę WMI, wynik przekształca na łańcuch i zapisuje informacje w dokumencie Word jako zaznaczenie:

```
Function GetWmiData($WmiClass)
{
    Get-WmiObject -class $wmiClass | Out-String |
    ForEach-Object {$selection.typeText($_)}
} #end GetWmiData
```

Podczas pobierania informacji WMI należy utworzyć ścieżkę do pliku za pomocą funkcji `CreateFilePath`, aby można było gdzieś zapisać dokument Word. Funkcja ta pobiera nazwę klasy WMI przez zmienną `$WmiClass`. Następnie za pomocą metody `substring` z klasy `System.String` usuwa sześć pierwszych znaków z nazwy klasy WMI. Te sześć znaków to przedrostek `Win32_` obecny w nazwach prawie wszystkich klas WMI. Aby być dokładniejszym, powinno się sprawdzić jeszcze inne wzorce nazw klas WMI i odpowiednio dostosować polecenie `substring` do znalezionej nazwy klasy. Następnie polecenie `Join-Path` buduje ścieżkę do pliku, która zostaje użyta przy zapisywaniu dokumentacji WMI. Poniżej znajduje się kod źródłowy tej funkcji:

```
Function CreateFilePath($wmiClass)
{
    $script:filename = $wmiClass.substring(6)
    $script:path = Join-Path -Path $folder -childpath $filename
} #end CreateFilePath
```

Następnie trzeba zapisać dokument Word. W tym celu najpierw należy utworzyć egzemplarz wyliczenia `Microsoft.Office.Interop.Word.WdSaveFormat` poprzez dokonanie rzutowania reprezentacji łańcuchowej tego wyliczenia jako typu. Musi to być typ referencyjny, dlatego dodano `[ref]`. Metoda `saveas` z obiektu `Word.Document` wymaga, aby zarówno ścieżka, jak i format zapisu były typami referencyjnymi. Po zapisaniu dokumentu obiekt `Word.Application` można usunąć z pamięci za pomocą metody `quit`. Poniżej znajduje się kompletny kod źródłowy funkcji `SaveWordData`:

```
Function SaveWordData($path)
{
    [ref]$SaveFormat = "microsoft.office.interop.word.WdSaveFormat" -as [type]
    $doc.saveas([ref]$path, [ref]$saveFormat::wdFormatDocument)
    $word.quit()
} #end SaveWordData
```

Punkt początkowy skryptu tworzy parę zmiennych i wywołuje odpowiednie funkcje, jak pokazano poniżej:

```
$folder = "C:\fso"
$wmiClass = "Win32_Bios"
$heading = "$wmiClass information:"
CreateWordDoc
CreateSelection($Heading)
GetWmiData($wmiClass)
CreateFilePath($wmiClass)
SaveWordData($path)
```

Poniżej znajduje się kompletny kod źródłowy opisywanego skryptu *SaveWmiInformation-AsDocument.ps1*:

SaveWmiInformationAsDocument.ps1

```
Function CreateWordDoc()
{
    $script:word = New-Object -ComObject word.application
    $word.visible = $true
    $Script:doc = $word.documents.add()
} #end CreateWordDoc
Function CreateSelection($Heading)
```

```

{
    $script:selection = $word.selection
    $selection.typeText($Heading)
    $selection.TypeParagraph()
} #end CreateSelection

Function GetWmiData($WmiClass)
{
    Get-WmiObject -class $WmiClass | Out-String |
    ForEach-Object {$selection.typeText($_)}
} #end GetWmiData

Function CreateFilePath($WmiClass)
{
    $script:filename = $WmiClass.substring(6)
    $script:path = Join-Path -Path $folder -childpath $filename
} #end CreateFilePath

Function SaveWordData($path)
{
    [ref]$SaveFormat = "microsoft.office.interop.word.WdSaveFormat" -as [type]
    $doc.saveas([ref]$path, [ref]$SaveFormat::wdFormatDocument)
    $word.quit()
} #end SaveWordData

# *** punkt początkowy ***
$folder = "C:\fso"
$WmiClass = "Win32_Bios"
$Heading = "$WmiClass information:"
CreateWordDoc
CreateSelection($Heading)
GetWmiData($WmiClass)
CreateFilePath($WmiClass)
SaveWordData($path)

```

Wiedza tajemna

Rejestrowanie pomysłów na skrypty

Chris Bellée, Premier Field Engineer

Microsoft Corporation, Australia

Klienci często proszą mnie o przykładowe skrypty. Często podsuwają dobre pomysły, więc piszę skrypty, które zachowuję do użycia kiedy indziej. Używam do tego Notatnika, który jest łatwy w obsłudze i szybki. Pliki tekstowe odczytują programy wszelkiego typu, więc nie muszę się martwić, czy mam zainstalowany odpowiedni program z pakietu Microsoft Office. Podczas pisania skryptów często odkrywam nowe techniki i technologie. Wówczas od razu piszę przykładowy skrypt z ich użyciem, aby mieć jakiś przykład na przyszłość. Nie jest to formalna technika, ale jej zaletą jest prostota.

Doskonałym pomysłem jest utworzenie bazy danych skryptów, np. w programie Microsoft Office Access. Skrypty można podzielić na kategorie tematyczne według dowolnego kryterium, a następnie można sporządzać raporty wskazujące, w jakich dziedzinach brakuje jeszcze określonych rodzajów skryptów. Później wystarczy tylko przejrzeć raport i uzupełnić braki. Baza danych może służyć nie tylko do przechowywania, ale również do wyszukiwania skryptów. Bardzo ciekawym pomysłem jest utworzenie konstruktora skryptów bazującego na ogólnych procedurach zapisanych w bazie danych. Oczywiście jest to przede wszystkim metoda tworzenia nowych skryptów na podstawie zapisanych pomysłów, a nie wyszukiwania kolejnych okazji do napisania skryptu. Konstruktor skryptów może być rozszerzeniem przenośnego centrum skryptowego dostępnego z Microsoft Script Center.

Gdy zabieram się do napisania nowego skryptu, z reguły używam klas platformy Microsoft .NET, jeśli są dostępne, zamiast funkcji VBScript czy nawet klas WMI, ponieważ lepiej znam platformę .NET i uważam, że jest to najlepsze rozwiązanie. Platforma .NET jest technologią macierzystą dla konsoli Windows PowerShell i o wiele łatwiej jest wywołać interfejsy API Win32 w Windows PowerShell 2.0. Używam tych API, gdy nie mam dostępu do jakiejś klasy w bibliotece .NET, np. do tworzenia udziałów sieciowych i ustawiania ich uprawnień.

Współpraca nad skryptami

Podczas gdy pisanie skryptów może być przyjemne i chyba rzeczywiście wielu administratorów sieci czerpie z tego przyjemność, bywa też czasochłonne. Dlatego proces tworzenia skryptów powinien być tak zaplanowany, aby korzyści z niego miała cała organizacja. Należy pamiętać, że istnieje różnica między nauką pisania skryptów a pisanem skryptów. To prawda, że administratorzy sieci często uczą się poprzez pisanie skryptów, ale mimo wszystko należy rozdzielać te dwie czynności. Nauka pisania skryptów to działalność szkoleniowa i należy ją traktować jako obciążenie budżetu na szkolenia. Natomiast pisanie skryptów to działalność operacyjna. Jeśli administrator sieci osiem godzin pisze skrypt sprawdzający, ile jest wolnego miejsca na dysku serwera, to znaczy, że nie umie pisać skryptów. W związku z tym te osiem godzin należy zaliczyć do obciążeń szkoleniowych, a nie produkcyjnych. Posiadanie przez firmę kilkunastu różnych skryptów sprawdzających ilość wolnego miejsca na dysku komputera nie jest efektywnym ekonomicznie rozwiązaniem.

Jeśli pracownicy różnych działów piszą takie same skrypty, marnuje się czas oraz wysiłek i trzeba pomyśleć o wdrożeniu jakichś narzędzi do współpracy. Przy ich użyciu pracownicy mogą udostępniać swoje skrypty innym oraz zgłaszać zapotrzebowanie na konkretne skrypty. W ten sposób oddziela się działalność szkoleniową od produkcyjnej, a więc oszczędza czas i wysiłek.

Dodatkowe źródła informacji

- Centrum skryptów portalu TechNet pod adresem <http://www.microsoft.com/technet/scriptcenter> zawiera wiele przykładowych skryptów z filtrami wyszukiwania LDAP.

- W artykule <http://support.microsoft.com/kb/947709> w Bazie wiedzy znajdują się informacje na temat sposobów użycia poleceń kontekstu NetSh Advanced Firewall.
- W portalu MSDN na stronie [http://msdn.microsoft.com/en-us/library/aa746385\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa746385(VS.85).aspx) znajduje się dokumentacja składni filtrów wyszukiwania LDAP.
- Wszystkie skrypty opisane w tym rozdziale są dostępne do pobrania w repozytorium centrum skryptów portalu TechNet pod adresem <http://gallery.technet.microsoft.com/scriptcenter/PowerShell-40-Best-d9e16039>.

Skorowidz

.NET, 92, 111
.NET Framework, 88

A

abstrakcyjne drzewa składni, 451
Active Directory, 444, 445, 481
ADO, ActiveX Data Object, 88, 208
adres IP, 256
ADSI, Active Directory Services Interface, 88
agent aktualizacji, 221
akcelerator
 [ADSISeacher], 209
 [WMICLASS], 275, 403
aktualizowanie
 dokumentacji, 287
 pomocy, 35–37
aktywności
 nieautomatyczne, 604
 przepływów pracy, 602, 603
 równoległe, 605
 sekwencyjne, 609
aktywność
 Checkpoint-Workflow, 603, 609
 ForEach -Parallel, 603
 InlineScript, 604, 611
 Parallel, 603
 Sequence, 603
 Suspend-Workflow, 603
algorytm MD5, 27
alias, 122
 cat, 366
 ft, 51
aliasy
 parametru, 341
 stałe, 125
 tylko do odczytu, 124
 nazw funkcji, 127
 typów danych, 249
AllUsersAllHosts, 148
AllUsersCurrentHost, 148
analiza
 defragmentacji, 219
 składniowa skryptów, 451
 składniowa skryptu, 455
API, pplication programming interface, 88
API C Windows, 433
aplikacja, 117, *Patrz także* narzędzie
aplikacje IIS, 96
argumenty pozycyjne, 236
AST, abstract syntax tree, 451
atrybut
 [ordered], 529
 mandatory, 337
 ValidateRange, 409
atrybuty
 parametrów, 339, 340
 weryfikacyjne, 346, 451
automatyzacja, 88, 90
 testów interfejsów, 434
 zadań, 87
AWDS, Active Directory Web Service, 62

B

baza danych skryptów, 227
Bellée Chris, 226, 454
bezpieczeństwo, 29, 72, 96
bezpieczeństwo danych, 492
blok
 Catch, 392, 407
 Finally, 392
 Try, 392

błąd, 135, 147, 160, 170, 199, 205, 281, 330, 336, 338, 391, 403, 407, 520, 582, 585, 620
 błędy
 logiczne, 441, 527
 składniowe, 443, 524
 wykonawcze, 524
 zakresu, 408
 brak
 dostawców WMI, 396
 uprawnień, 389
 znaku dolara, 342
 Brasser Jaap, 341
 Brundage James, 151, 391, 417, 430, 432

C

Canastreiro Luís, 112, 164
 Carlos Ruiz Lopez Juan, 234, 251
 Carter Marc, 317
 Cedenro Enrique, 440, 444
 certyfikat, 472
 chmura, 144
 Christopher Jim, 307
 CLR, Common Language Runtime, 112
 COM, Component Object Model, 88, 136, 185
 Costantini Peter, 299
 Craig Burley James, 261
 CurrentUserAllHosts, 148
 CurrentUserCurrentHost, 148
 czyszczenie zawartości pliku, 444

D

dane
 dziennika, 516
 konfiguracji, 619
 wejściowe parametru, 342
 WMI, 247
 XML, 431
 DCOM, Distributed Component Object Model, 95
 debugger aliasu, 552
 debugowanie, 483, 523, 554
 debugowanie skryptów, 515, 545, 558, 560
 definiowanie
 funkcji, 240, 244
 komentarzy, 281
 logiki biznesowej, 254
 defragmentacja, 219

Dekens Luc, 529
 diagnostyka, 546
 diagnostyka szczegółowa, 527
 dodawanie
 aktywności sekwencyjnej, 609
 dokumentacji, 271
 komentarza, 275, 288
 punktów kontrolnych, 607
 dokumentacja, 222, 271, 287
 Active Directory, 70, 73
 SDK, 185
 dokumentowanie
 przebiegu testowania, 419
 skryptów, 447
 dołączanie
 dwukropka, 409
 pliku, 154
 dopisywanie danych, 504
 dostawca
 CIMWin32, 396
 Group, 621
 MSIProv, 398
 User, 621
 WMI, 172, 396
 zasobów DSC, 614
 dostęp do
 folderu, 103, 137
 konfiguracji sesji, 481
 obiektu, 338
 obiektu COM, 136
 punktów wstrzymania, 549
 dryf konfiguracji, 623
 DSC, Desired State Configuration, 17, 613
 dysk
 HKCR, 400, 401
 PowerShell, 141, 142
 sieciowy, 513
 dziennik, 431, 441, 443, 502, 504
 aplikacji, 519
 śledzenia, 506
 Windowsupdate.log, 516
 zdarzeń, 478, 518, 519

E

edytor skryptów, 490
 edytowanie, 484
 EFS, Encrypting File System, 349
 egzemplarz klasy, 454

eksportowanie
 danych, 445
 historii poleceń, 204
 poświadczeń, 354
 ETS, Extended Type System, 143
 ewaluacja zmiennej, 405

F

Farr Ian, 492, 508
 filtr, 263, 266
 filtr HasMessage, 266
 filtrowanie klas, 48
 filtry wyszukiwania LDAP, 209–213
 Finke Douglas, 240
 folder
 %username%, 135
 Backups, 34
 Dokumenty, 29
 foldery
 skryptów, 482
 tymczasowe, 277
 wyjściowe, 511
 fragmenty kodu, 569, 570
 funkcja, 231, 242, 427
 AddOne, 264, 265
 Add-Registry, 531
 Add-RegistryValue, 533
 add-two, 543
 bletch, 262
 Check-AllowedValue, 408
 Check-Number, 343
 ConvertFrom-Cab, 188
 CreateFilePath, 225
 CreateSelection, 224
 DivideNum, 556
 Enable-PSRemoting, 583–585
 Expand-Cab, 221
 Format-NonIPOutput, 260
 Get-AllowedComputer, 386–388
 Get-Bios, 442
 Get-Change, 427, 428
 Get-Choice, 383
 Get-Comments, 279
 Get-ComputerInfo, 321, 322
 Get-CountryByIP, 510
 Get-Discount, 254
 Get-FileName, 453
 Get-Folder, 511
 Get-ieStartPage, 275, 276

Get-MemberOf, 103
 Get-MoreHelp, 126, 128
 Get-MyModule, 315–317
 Get-OsVersion, 114
 Get-TempFile, 428
 Get-TextStatistics, 235–239
 Get-Type, 152
 Get-ValidWmiClass, 403–405
 Get-Version, 165
 Get-Volume, 163, 164
 Get-WmiClass, 131–133
 Get-WmiClasses, 283, 284
 Get-WmiInformation, 405
 Get-WmiProvider, 398, 402
 IntelliSense, 567
 Kontrola konta użytkownika, 96
 New-Cab, 186, 187
 New-DDF, 192
 New-LocalUser, 435
 New-ModulesDrive, 313
 New-TempFile, 514
 New-TestConnection, 344
 ParseAction, 154
 Remove-OutPutFile, 278
 Set-LocalGroup, 437
 Set-ScreenSaverTimeout, 178
 Start-Transcript, 441
 Test-ComputerPath, 384
 Test-IsAdmin, 108
 Test-IsAdministrator, 518
 Test-ModulePath, 309
 Test-Scripts, 427
 Write-Path, 238

funkcje
 definiowanie logiki biznesowej, 254
 przenośność, 244
 rozwijania nazw, 525
 samodzielność, 244
 spójność wyników, 243
 z innych skryptów, 152
 z wieloma parametrami, 244, 252

G

gałąź
 CLSID, 401
 Current_User, 522
 HKEY_Classes_Root, 400
 Hkey_Current_User, 520

- generowanie
 - dokumentacji, 450
 - pliku konfiguracyjnego, 481
 - raportów, 503
- gip, 24
- gotowe fragmenty kodu, 569
- Goude Niklas, 480
- GPO, 469
- grupa, 103, 104
- grupa zabezpieczeń
 - dodawanie użytkownika, 76
 - nazwa, 75
 - ścieżka, 75
 - usuwanie użytkownika, 76
 - zakres, 75
- Gusev Vasily, 558

H

- hasło, 77, 348, 620, 621
- Helmick Jason, 24
- Hicks Jeffery, 450
- hierarchiczna przestrzeń nazw, 167
- Hill Keith, 304
- historia poleceń, 204
- Hofferle Jason, 87, 248
- Holmes Lee, 354
- Huffman Clint, 173

I

- identyfikacja wersji, 89
- identyfikator
 - GUID, 203
 - klasy dostawcy, 401
 - OID, 351
 - ObjectGUID, 81
 - RID, 71
 - SID, 54, 81, 99, 103, 444
- IIS, Internet Information Services, 96
- importowanie
 - pliku, 445
 - poświadczeń, 354
- indeksowanie zmiennej \$args, 331
- informacje
 - diagnostyczne, 436, 526
 - dotyczące domeny, 71
 - o błędach, 37, 125, 133, 147, 161, 517, 593
 - o czasie logowania, 502

- o dostawcy klasy, 396
- o karcie sieciowej, 595
- o komputerach, 602
- o odmowie dostępu, 97
- o postępie testów, 428
- o typie obiektu, 403
- o Usłudze konfiguracji, 613
- o wersji platformy .NET, 112
- o wykonywaniu funkcji, 438
- w nagłówku, 290
- zbędne, 293
- infrastruktura WMI, 167
- instalowanie
 - konsoli, 25
 - modułów, 308
 - modułu Active Directory, 62
- instrukcja
 - #Requires, 97, 314, 393
 - [cmdletbinding()], 434
 - exit, 438
 - Foreach, 174, 187
 - if, 388, 397
 - On Error Resume Next, 406
 - Param, 335, 381, 409, 416
 - Return, 261, 438
 - Switch, 357
 - Throw, 333
 - Trap, 391, 407

- instrumentacja, 515
- instrumentacja zarządzania Windows, 167
- integralność, 492
- interfejs
 - API, 433
 - automatyzacji, 88
 - programistyczny, 88
 - usług Active Directory, 88
- ISE, Integrated Scripting Environment, 317

J

- jednostka organizacyjna, 74, 211
- język VBScript, 40, 91, 393, 406
- Jones Don, 43, 483, 489, 545

K

- katalog LDAP, 167
- Kearney Sean, 40
- klamra, 126

klasa

- __provider, 397, 398
- _Namespace, 168
- _provider, 172
- Enum, 110
- ErrorRecord, 251
- InvocationInfo, 180
- Io.Path, 513
- Microsoft.Win32.Registry, 92
- PromptForChoice, 358
- PSObject, 511
- PSParser, 456
- Security.Principal.WindowsBuiltInRole, 109
- Security.Principal.WindowsIdentity, 518
- SecurityIdentifier, 100
- System.Enum, 109, 179
- System.Environment, 113, 115
- System.IO.FileInfo, 311
- System.IO.Path, 412
- System.Management.Automation.LineBreak, 547
- System.Math, 182
- System.Random, 160
- System.String, 403
- System.TimeSpan, 424, 428
- System.Version, 113
- Win32_Bios, 50, 565
- Win32_ComputerSystem, 412
- Win32_Desktop, 176
- Win32_LogicalDisk, 245
- Win32_OperatorSystem, 165
- Win32_Process, 52
- Win32_Product, 398
- Win32_Service, 51
- Win32_UserAccount, 54
- Win32_Volume, 218
- WindowsIdentity, 98
- WMI Win32_NetworkAdapterConfiguration, 256
- Word.Application, 224

klasy

- abstrakcyjne, 49
- Association, 52
- CIM, 45
- dynamiczne, 173
- pospolite, 173
- rdzenne, 173
- WMI, 45, 47, 49, 173

klauzula WHERE, 118

klawisz Tab, 567

Klindt Todd, 119

klucz \$scriptRoot, 520

klucze rejestru, 222

- ForScripting, 520, 617

kod ADSI, 65

kolejność wykonywania aktywności, 610

kolekcja, 174

komentarze, 275, 452

- efektywne, 286

- jednowierszowe, 271, 280, 294

- na końcu wiersza, 295

- opisywanie struktur, 295

- wielowierszowe, 279

komentarzowy blok nagłówkowy, 449

komunikacja, 299

konfiguracje DSC, 623

konfigurowanie, 616, 619

- konsoli, 33

- profilu, 121

- przycisków, 139

- środowiska skryptowego, 121

- węzła, 625

- żądanego stanu programu, 613

konkatenacja, 278, 405

konsola

- MMC, 207

- PWA, 43

- Windows PowerShell, 21, 23

- Windows PowerShell ISE, 94

konstrukcja

- Begin-Process-End, 338

- Try-Catch, 407

- Try-Catch-Finally, 334, 391

konstruktor, 159, 182

konsumenci WMI, 167

kontener users, 77

konto

- komputera, 74

- użytkownika, 78

kontrola

- dryfu konfiguracji, 623

- konta użytkownika, 96, 389

- wersji, 490, 496

- wersji skryptów, 489

- wykonywania poleceń, 30

konwencje nazewnictwa, 151

konwersja łańcucha, 403, 405

krokowe wykonywanie skryptu, 541

- kwalifikator
 - abstract, 49
 - association, 48
 - deprecated, 48
 - dynamic, 49
 - singleton, 48
 - supportsupdate, 48

L

- LDAP, 167, 208
- liczba
 - błędów, 413
 - opcji, 382
 - poleceń, 24, 158
 - własności i egzemplarzy, 50
- liczenie testów, 428
- lista
 - czasowników, 235
 - dostawców WMI, 172
 - dostępnych modułów, 302
 - gotowych fragmentów kodu, 569
 - jednostek organizacyjnych, 215
 - przestrzeni nazw, 170
 - punktów wstrzymania, 556
 - skryptów, 491
- logika
 - biznesowa, 254, 255
 - programu, 254
- logowanie, 467, 475–480
- lokalizacje sieciowe, 513
- Lopez Juan Carlos Ruiz, 554

Ł

- ładowanie modułów, 305
- łańcuch
 - miejscowy, 192
 - połączenia, 356
- łączenie zmiennych, 405

M

- magazyn
 - certyfikatów użytkownika, 473
 - skryptów, 492
- Maheu Georges, 166
- mapowanie dysków, 201
- maszyna wirtualna, 445
- Mayer Keith, 90

- McGlone Ashley, 61
- mechanizm potwierdzania, 76
- Mell Bill, 32
- Menedżer certyfikatów, 472, 473
- menu

- Debug, 546, 560
 - opcji, 582

- metoda
 - AddDays, 422
 - ChangeStartMode(), 175, 176
 - Connect(), 455
 - create, 47
 - createcab, 186
 - DerfagAnalysis, 218
 - DownloadString, 431
 - FindAll, 209
 - GetCurrent, 97, 518
 - GetNames, 109, 179
 - GetStringValue, 92
 - GetTempFileName, 412, 513
 - GetType, 403
 - GetValues, 110
 - IsInRole, 108
 - Namespace, 190
 - popup, 138–141
 - PromptForChoice, 382, 383
 - RegRead, 91
 - saveas, 225
 - split, 310
 - Tokenize, 456
 - ToString, 99, 353, 520
 - WshShell.popup, 139

- metody
 - klas WMI, 47
 - pisania skryptów, 164
 - pobierania danych, 328
 - statyczne, 182
 - zwracania danych, 358

- Minasi Mark, 364

- model COM, 88

- moduł, 118, 301, 395

- Active Directory, 61
 - dokumentacja, 70, 73
 - instalowanie, 62
 - nazwa lokalizacji, 73
 - zastosowanie, 63
 - znajdowanie kont użytkowników, 78, 81
 - znajdowanie wyłączonych użytkowników, 80
- BasicFunctions.psm1, 319

- CimCmdlets, 46
- DotNet, 152
- PSCX, 144
- moduły
 - instalowanie, 308
 - ładowanie, 305
 - sprawdzanie zależności, 314
 - tworzenie, 319
 - tworzenie folderu, 309
 - z udziałów, 318
- modyfikowanie
 - rejestr, 465, 466
 - skryptów, 256
 - wartości, 178
 - zmiennej path, 286
- monitowanie o informacje, 357
- Moravec David, 27
- możliwości konsoli, 23
- Muscetta Daniele, 470

N

- nadpisywanie dziennika, 500
- nagłówki, 290
 - komentarzowy, 449
 - skryptu, 493
- narzędzia
 - do pracy zdalnej, 94–96, 573
 - RSAT, 63
 - wiersza poleceń, 28
- narzędzie
 - CMD.exe, 142, 376
 - CSVDE, 444
 - DSMode.exe, 217
 - DSQuery.exe, 215, 216
 - fsutil, 24
 - IPConfig.exe, 24
 - Kinect, 390
 - LDIFDE, 444
 - MakeCab.exe, 192
 - NetDom, 88
 - NetSH, 88, 191
 - PoshPAIG, 324
 - SolarWinds Network Configuration Manager, 33
 - Streams.exe, 464
 - System Center Operations Manager, 221
 - VersionRecall, 497
 - VSS, 496

- WbemTest, 396
- WSH, 496
- nawias
 - okrągły, 248
 - trójkątny, 219
- nazwa
 - dostawcy WMI, 397, 399
 - główna użytkownika, 212
 - jednostki organizacyjnej, 211
 - lokalizacji, 73
 - modułu, 395
- niedozwolone polecenia rdzenne, 604
- niepoprawna wersja skryptu, 495
- niepoprawne typy danych, 403
- nieprzechwycone wyniki, 263
- Norman Richard, 375
- notacja
 - funkcyjna, 543
 - metodowa, 543
- numer
 - poprawki, 113
 - wersji, 493, 494
 - głównej, 113
 - kompilacji, 113
 - pomocniczej, 113

O

- obiekt, 454
 - \$wshShell, 137
 - COM, 143
 - COM WshShell, 93
 - DateTime, 413, 422
 - DirectoryEntry, 216
 - GPO, 470, 475
 - makecab.makecab, 188
 - MSGraph.Application, 505
 - poświadczeń, 355
 - PSCredential, 354, 355, 621
 - ScriptInfo, 180, 181
 - selection, 224
 - shell, 188
 - Shell.Application, 188, 190
 - Win32_OperatingSystem, 165
 - WindowsIdentity, 98, 105
 - Word.Document, 225
 - WshShell, 136–141
 - WshSpecialFolders, 137
 - XML, 431

obsługa

- aplikacji zewnętrznych, 191
- błędów, 379, 404
- brakujących parametrów, 380
- COM, 185
- hasel, 445
- IP, 259
- parametrów, 129
- parametrów nazwanych, 131
- platformy .NET, 182
- poleceń, 157
- skryptów, 146
- wejścia i wyjścia, 327
- WMI, 167

odblokowywanie kont użytkowników, 78

odczytywanie

- pliku tekstowego, 198
- rejestr, 91, 92

odmowa dostępu, 97, 525

ograniczanie

- liczby opcji, 382
- możliwości wyboru, 382
- wartości parametru, 409

ograniczenia typów parametrów funkcji, 249, 131

ogranicznik [Object[]], 339

ograniczona zasada wykonywania, 147

okienko

- Command, 565, 567
- skryptu, 566

okno

- poświadczeń, 583
- Testera, 396
- Windows PowerShell ISE, 563, 564

opcja

- Step Into, 560
- Step Over, 560

opcje pomocy, 35

operator

- contains, 101, 385, 387
- like, 101
- match, 101, 130
- [], 337
- przekierowania, 499, 501
- zakresu, 123

operatory filtrów, 211

opisywanie struktur, 295

P

pakiet

- MSI, 466, 482
- VSS, 496
- Windows Management Framework 4.0, 25

parametr, 243

- \$baseLineScript, 427
- action, 551
- append, 413, 515
- autosize, 208
- baseLineScript, 426
- class, 366
- classname, 45, 50
- commandtype, 215
- computer, 335
- computername, 117, 165, 206, 330, 573
- confirm, 76
- credential, 355, 581
- debug, 186, 434, 526
- description, 136, 569
- destination, 191
- discover, 67
- Encoding, 507
- Expression, 427
- filepath, 367
- filter, 50, 55, 68, 178, 213
- folder, 510
- force, 81, 125, 223, 583
- groupScope, 75
- identity, 80
- inputobject, 55
- keep, 595
- LDAPFilter, 213
- log, 427
- members, 76
- membertype, 591
- mode, 561
- modifiedScript, 426
- namespace, 45
- nazwany, 131
- numberOfTests, 427
- parent, 277
- password, 357
- path, 199
- PipelineVariable, 341
- Process, 413
- property, 50
- PSComputerName, 588, 600, 601
- qualifier, 48

- recurse, 412
- reset, 77
- script, 558
- step, 539
- strict, 542
- Text, 569
- Title, 569
- trace, 534
- TypeName, 454
- ValidatePattern, 344
- variable, 368
- Verbose, 179, 436
- whatif, 434, 437–440
- width, 507
- Wrap, 208
- parametry
 - instrukcji #Requires, 394
 - konfiguracji, 617
 - obowiązkowe, 337, 381
 - polecień, 34
 - standardowe, 434
 - wiersza poleceń, 426
- parser AST, 455
- pasek stanu, 532
- pętla, 128
- pętla foreach, 171, 339
- Pfeiffer Mike, 502
- pisanie
 - funkcji, 242
 - skryptów, 157
- planowanie skryptów, 59
- platforma .NET, 92, 111
- plik
 - \$files, 422
 - Autoexec.bat, 148
 - dotnettypes.format.ps1xml, 362
 - konfiguracji, 481
 - package.xml, 221, 222
 - passwordHash.txt, 354
 - TroubleShoot.bat, 29
 - WindowsUpdate.log, 134
 - wsusscn2.cab, 222
- pliki
 - .bat, 465
 - .cab, 186, 187, 222
 - .ddf, 192
 - .ps1, 143
 - .psm1, 319
 - .xml, 204
 - ADM, 471
 - cabinet, 185
 - CSV, 502
 - MOF, 614, 616
 - pomocy, 36, 37
 - tekstowe, 198, 386, 509, 516
 - wsadowe, 29
- pobieranie
 - danych, 328
 - danych WMI, 247
 - dokumentacji
 - z komentarzy, 452
 - z pomocy, 447
 - hasel, 348
 - informacji, 93, 95
 - łańcuchów połączenia, 356
- podpisywanie kodu, 472
- polecenia, 24
 - Active Directory, 213
 - CIM, 45
 - cmdlet, 27, 30, 31, 573
 - diagnostyczne, 392, 546, 555
 - New-Aduser, 77
 - promieniste, 205
 - rdzenne, 604
- polecenie
 - Add-Content, 509
 - Add-Member, 511
 - cd, 23
 - Clear-Host, 541
 - CLS, 376
 - Complete-Transaction, 179
 - Continue, 558
 - ConvertFrom-SecureString, 355
 - ConvertTo-Html, 370, 502, 509
 - Copy-Item, 200, 205
 - dir, 23
 - Disable-PSBreakpoint, 561
 - Enable-PSRemoting, 583
 - Enter-PSSession, 70, 587
 - Export-Clixml, 359, 509
 - Export-Csv, 370, 502, 509
 - ForEach-Object, 193, 204, 332, 423
 - Format-List, 81
 - Format-Table, 51, 207, 557, 566
 - Get-ADDefaultDomainPasswordPolicy, 72
 - Get-ADDomain, 71
 - Get-ADDomainController, 67, 72
 - Get-ADForest, 71
 - Get-ADOrganizationalUnit, 216
 - Get-ADRootDSE, 73

polecenie

- Get-Aduser, 77
- Get-ADUser, 77–83
- Get-Alias, 122, 133
- Get-ChildItem, 187, 193, 342, 412, 422, 456
- Get-CimAssociatedInstance, 52–56
- Get-CimClass, 45–48
- Get-CimInstance, 46–51, 600
- Get-Command, 66, 602
- Get-Content, 198, 200, 219, 366, 525
- Get-Credential, 481
- Get-Date, 267, 422, 442
- Get-EventLog, 520, 605
- Get-ExecutionPolicy, 147, 468, 569
- Get-FileHash, 27
- Get-Help, 38, 125, 286, 323, 484, 573
- Get-History, 204
- Get-IseSnippet, 571
- Get-Item, 374, 407
- Get-Job, 591, 594
- Get-Member, 52, 177, 454, 507, 591
- Get-Module, 63, 65, 321
- Get-NetAdapter, 595
- Get-NetIPConfiguration, 24
- Get-Process, 118, 125, 157, 360, 561, 591
- Get-PSBreakpoint, 556, 560
- Get-PSDrive, 401
- Get-PSSession, 213, 587
- Get-Service, 117, 157, 206, 218
- Get-Variable, 329
- Get-WebServiceProxy, 510
- Get-WindowsFeature, 611
- Get-WmiClass, 130
- Get-WmiObject, 172, 219, 330, 405, 442, 595
- help, 42
- Import-Clixml, 359
- Import-Module, 213, 223
- Import-Module ActiveDirectory, 62
- Import-PSSession, 107
- Invoke-Command, 65, 94, 205, 477, 588
- Invoke-Expression, 198, 413
- Invoke-History, 204
- ipconfig, 28, 365
- Join-Path, 134, 192, 225, 401
- Measure-Command, 424, 426, 430
- Move-Item, 514
- New-ADGroup, 75
- New-ADOrganizationalUnit, 74
- New-Alias, 123
- New-DDF, 194
- New-EventLog, 517
- New-IseSnippet, 570
- New-Item, 123, 134, 150, 533
- New-ItemProperty, 521
- New-Object, 185, 430, 454
- New-PSDrive, 313
- New-PSSession, 586, 587
- New-Variable, 135, 342
- Out-File, 219, 366, 413, 506, 513
- Out-Host, 364
- Out-Null, 401, 521
- Out-String, 520
- ping, 197, 384
- Quit, 558
- Read-Host, 352, 412, 434
- Receive-Job, 593
- Register-PSSessionConfiguration, 481
- Remove-Item, 125, 571
- Remove-Job, 591
- Remove-PSBreakpoint, 560
- Remove-PSDrive, 402
- Remove-PSSession, 213, 587
- Rename-ADObject, 74
- Save-Help, 37
- Search-ADAccount, 78
- Select-Object, 209, 361, 529
- Select-String, 240
- Set-ADAccountPassword, 77
- Set-Alias, 123, 124
- Set-Content, 509
- Set-ExecutionPolicy, 146, 466, 471
- Set-Item, 134
- Set-ItemProperty, 521
- Set-Location, 142, 586
- Set-PSBreakpoint, 547, 558
- Set-PSDebug, 529–534, 539–542, 548
- Set-StrictMode, 543
- Split-Path, 187
- Start-DscConfiguration, 623
- Start-Job, 594
- Start-Transaction, 178
- Start-Transcript, 441, 515, 587
- Step-into, 558
- Stop-Job, 594
- Stop-Process, 157
- Stop-Transcript, 442, 515
- substring, 225
- Tee-Object, 367, 509, 515
- Test-Connection, 526
- Test-Path, 154, 342, 400, 533

Test-WsMan, 584
 Update-Help, 36
 Wait-Job, 593
 Where-Object, 117, 124, 425, 509, 580
 Write-Debug, 179, 180, 192, 441, 525
 Write-Error, 509
 Write-Host, 434, 551
 Write-Verbose, 178, 400, 438, 545
 Write-Warning, 509
 pomoc, 35, 38, 41, 447, 484
 do skryptu, 271
 komentarzowa, 281
 techniczna, 484
 porównanie szybkości działania, 424
 potok, 423
 potwierdzenie wykonania poleceń, 31
 powtarzalność, 218
 poziom
 drugi śledzenia, 533
 pierwszy śledzenia, 532
 zasady wykonywania, 146
 praca zdalna, 94, 95, 573
 proces DSC, 614
 profil, 144, 148–152
 AllUsersAllHosts, 150
 CurrentUserAllHosts, 150
 CurrentUserCurrentHost, 149, 150
 program, *Patrz* narzędzie
 programowanie, 234
 programy do kontroli wersji, 496
 projektowanie
 modułów, 301
 skryptów, 159, 229, 231
 protokół XMPP, 143
 Prox Boe, 324
 przechowywanie
 dzienników, 513
 informacji, 522
 skryptów, 492
 tekstu, 509
 przechwytywanie błędów, 391
 przedrostek ! CALL, 534
 przeglądanie
 danych, 421
 tablicy, 128
 przeglądarka Internet Explorer, 465
 przekazywanie
 opcji do poleceń, 34
 wartości, 330
 wielu parametrów, 129

przekierowanie, 219
 przełącznik
 bypass, 462, 468
 debug, 526, 528
 forceDiscover, 67
 -UseTransaction, 178
 whatif, 438
 wrap, 566
 przepływ pracy, workflow, 94, 483, 597
 przesłanie istniejących poleceń, 126
 przestrzenie nazw WMI, 45, 168
 przestrzeń nazw, 167, 374
 przetwarzanie
 łańcuchów, 99
 tokenów, 456
 przyciski metody popup, 139
 przypisywanie
 wartości domyślnej, 380
 zmiennych globalnych, 374
 przywracanie danych, 492
 pulpit zdalny, 201
 punkt kontrolny, 606
 na poziomie aktywności, 607
 na poziomie przepływu pracy, 607
 punkt wstrzymania, 547–550
 PWA, PowerShell Web Access, 43

R

Rahim Ibrahim Abdul, 417, 430, 432
 raport, 415, 432, 457
 reagowanie na punkty wstrzymania, 555
 reguła zabezpieczeń, 393
 rejestr, 91, 178, 222, 465, 520
 rejestr błędów, 405
 rejestrowanie
 danych, 442, 508, 509
 wyników, 499, 500
 zdarzeń, 519
 repozytorium, 492
 Riedel Alexander, 496
 Ring Jan Egil, 37
 rodzaje błędów, 391
 rola FSMO, 65, 70
 role użytkownika, 107
 Rottenberg Hal, 142
 rozwiązywanie problemów, 504, 523
 rozwijanie nazw, 567
 równoległe wykonywanie poleceń, 599

RPC, remote procedure call, 95
 RSAT, Remote Server Administration Tools, 65
 rzutowanie wartości parametrów, 451

S

Sajid Osama, 560
 scenariusz używania, 379
 Schneider Andy, 546
 Schwinn Dave, 369
 SDK, Software Development Kit, 185
 sekcja
 Function, 416
 robocza skryptu, 279
 Shell Brandoe, 242
 Siddaway Richard, 468
 Siepser Gary, 338, 390
 składnia podstawowa, 411
 składowe
 klasy SecurityIdentifier, 100
 klasy System.Math, 183–185
 klasy Win32_LogicalDisk, 245–247
 obiektu Shell.Application, 188, 190
 obiektu WshShell, 137
 skompilowane pliki pomocy, 464
 skrót, hash, 353
 skrypt
 AddOne.ps1, 372
 AddTwoError.ps1, 543, 544
 BackUpFiles.ps1, 272
 BadScript.ps1, 535
 CheckForPdfAndCreateMarker.ps1, 288
 CheckNumberRange.ps1, 343
 CheckProviderThenQuery.ps1, 398
 CmdLineArgumentsTime.ps1, 292
 ConversionFunctions.ps1, 153, 241
 ConvertToFahrenheit_Include.ps1, 154, 291
 ConvertUseFunctions.ps1, 154
 Copy-Modules.ps1, 312, 321
 CreateCab.ps1, 187, 188
 CreateCab2.ps1, 194
 CreateFileNameFromDate.ps1, 272
 CreateRegistryKey.ps1, 521, 531, 534
 CreateScriptingRegistryKey.ps1, 223
 DebugRemoteWMI_Session.ps1, 526
 DefragAnalysisReport.ps1, 220
 DemoConsoleBeep.ps1, 293
 DemoTrapSystemException.ps1, 251
 DemoUserConfig.ps1, 619
 DisplayProcessor.ps1, 485
 DotSourceScripts.ps1, 258
 ExpandCab.ps1, 191
 ExportBiosToCsv.ps1, 158
 FilterHasMessage.ps1, 266
 FilterToday.ps1, 267
 FindDisabledUserAccounts.ps1, 287
 FindLargeDocs.ps1, 255
 FunctionGetIPDemo.ps1, 258
 GetAdminFunction.ps1, 110
 Get-AllowedComputer.ps1, 387, 388
 Get-AllowedComputerAndProperty.ps1, 389
 Get-Bios.ps1, 329, 332, 389
 Get-BiosArray1.ps1, 331
 Get-Biosarray2.ps1, 331
 Get-BiosInformation.ps1, 380
 Get-BiosInformationDefaultParam.ps1, 381
 GetBiosMandatoryParameter.ps1, 337
 Get-BiosMandatoryParameterWithAlias.ps1, 341
 Get-BiosParam.ps1, 336
 Get-ChoiceFunction.ps1, 383
 GetCmdletsWithMoreThanTwoAliases.ps1, 122
 GetCommentsFromScript.ps1, 452
 GetComputerInfoWorkFlow.ps1, 601
 Get-ComputerWmiInformation.ps1, 419
 Get-CountryByIP.ps1, 510, 511
 Get-DiskSpace.ps1, 247
 GetDrivesCheckAllowedValue.ps1, 408, 409
 GetDrivesValidRange.ps1, 410
 Get-EnabledBreakpointsFunction.ps1, 557
 Get-EventLogData.ps1, 605
 Get-IPObjektDefaultEnabled.ps1, 259
 Get-MemberOf.ps1, 103
 Get-ModifidFiles.ps1, 421, 422
 Get-ModifidFilesUsePipeline.ps1, 423–425
 Get-MoreHelpWithAlias.ps1, 127
 Get-OSVersion.ps1, 115
 Get-PowerShellRequirements.ps1, 26
 Get-PsVersionNet.ps1, 92
 Get-PsVersionRegistry.ps1, 89
 Get-PSVersionRemoting.ps1, 94
 Get-PsVersionWmi.ps1, 92
 Get-PSVersionWorkflw.ps1, 93
 GetRandomObject.ps1, 159
 GetRunningService.ps1, 117, 118
 Get-ScriptHelp.ps1, 447
 Get-ScriptVersion.ps1, 493
 GetServicesInSvchost.ps1, 294
 GetSetieStartPage.ps1, 276

- Get-ValidWmiClassFunction.ps1, 404, 405
- Get-Version.ps1, 165
- get-VM.ps1, 395
- Get-WindowsEdition.ps1, 495
- Get-WinFeatureServersWorkflow.ps1, 609
- Get-WmiClass.ps1, 130
- Get-WmiClass2.ps1, 132
- Get-WmiClass2WithAlias.ps1, 133
- GetWmiClassesFunction1.ps1, 283
- Get-WmiProviderFunction.ps1, 402
- Get-WmiProviders.ps1, 172
- InLineGetIPDemo.ps1, 257
- InternetScript.ps1, 464
- LogChartProcessWorkingSet.ps1, 505
- LogonScriptWithLogging.ps1, 501
- MandatoryParameter.ps1, 381
- MeasureAddOneFilter.ps1, 264
- MeasureAddOneFunction.ps1, 264
- My-Function.ps1, 528
- New-LocalGroupFunction.ps1, 438
- New-TempFile.ps1, 513, 514
- ParseScriptCommands.ps1, 455–457
- PingComputers.ps1, 344
- PingIpAddress.ps1, 345
- PinToStartAndTaskBar.ps1, 33
- PromptForChoice.ps1, 358
- RecursiveWmiNamespaceListing.ps1, 169
- RemoteWmiSessionNoDebug.ps1, 526
- RemoveUserFromGroup.ps1, 76
- RequireModuleVersion.ps1, 395
- RequiresModule.ps1, 118
- SaveWmiInformationAsDocument.ps1, 225
- ScriptFolderConf.ps1, 616
- ScriptFolderVersion.ps1, 617
- ScriptFolderVersionUnzip.ps1, 618
- ScriptFolderVersionUnzipCreateUsersAnd
 - ↳ Profile.ps1, 622
- SearchAllComputersInDomain.ps1, 295
- SearchForWordImages.ps1, 289
- Set-LocalGroupFunction.ps1, 436
- Set-SaverTimeOut.ps1, 181
- SetScriptExecutionPolicy.vbs, 467
- SetServicesConfig.ps1, 623
- SimpleTypingError.ps1, 542
- SimpleTypingErrorNotReported.ps1, 542
- skrypt StringArgsArray.ps1, 332
- skrypt StringArgsArray2.ps1, 332
- skrypt
 - Switch_DebugRemoteWmiSession.ps1, 526, 527
 - skrypt TestAdminCreateEventLog.ps1, 518
 - skrypt Test-ComputerPath.ps1, 384
 - skrypt Test-IsAdminFunction.ps1, 108, 394
 - Test-IsInRole.ps1, 110
 - Test-Script.ps1, 467
 - Test-ScriptHarness.ps1, 411–415
 - Test-TwoScripts.ps1, 426, 429
 - TranscriptBios.ps1, 441, 442
 - UpdatehelptrackErrors.ps1, 37
 - UseADCmdletsToCreateOuComputerAnd
 - ↳ User.ps1, 75
 - UseGetMemberOf.ps1, 103, 106
 - ValidateRange.ps1, 343
 - WriteBiosInfoToWord.ps1, 290
- skrypty, 157
 - analiza składowa, 455
 - brak obsługi aplikacji, 191
 - brak obsługi COM, 185
 - brak obsługi platformy, 182
 - brak obsługi poleceń, 157
 - brak obsługi WMI, 167
 - braki, 292
 - diagnostyczne, 483
 - dokumentowanie, 447
 - kontrola wersji, 489
 - kończenie pracy, 393
 - korzyści stosowania, 217
 - krok po kroku, 536, 541
 - lista warunków używania, 291
 - logowania, 475–480
 - obsługa błędów, 379
 - opisywanie struktur, 295
 - pomocy technicznej, 484
 - porównanie szybkości działania, 424
 - powody napisania, 293
 - projektowanie, 231
 - projektowanie pomocy, 271
 - raportujące, 483
 - rozwiązywanie problemów, 523
 - samodzielne, 483
 - skomplikowane konstruktory, 159
 - stosowanie, 197
 - testowanie, 411
 - testowanie wydajności, 421
 - uprawnienia, 393
 - uruchamianie, 417, 475
 - wdrażanie, 459
 - zasady wykonywania, 461
 - zdolność do adaptacji, 223

słowo kluczowe

- Configuration, 614
- constant, 125
- DependsOn, 618
- Filter, 255
- Function, 126, 192, 232, 239, 244
- Mandatory, 381
- Parallel, 605
- param, 526
- read-only, 124
- Workflow, 598

Snover Jeffrey, 95, 107, 116

spójność wyników, 243

sprawdzanie

- konfiguracji węzła, 625
- poprawności danych, 342
- wartości granicznych, 408
- wersji platformy .NET, 112
- wersji systemu, 165
- zależności modułów, 314
- zawartości tablicy, 385
- sprawdzenie wersji systemu, 165

Stahler Wes, 479

standard WS-Management Protocol, 583

standardowe

- czasowniki, 232
- konwencje nazewnictwa, 158

status usługi bits, 157

Stewart Bill, 406

stosowanie znaków specjalnych, 212

Stranger Stefan, 221

strefa internetowa, 463

struktura komentarza, 275

struktury zagnieżdżone, 295

strumień

- Debug, 393, 509
- Error, 509
- Success, 509
- Verbose, 509
- Warning, 509

strzałka przekierowująca, 366

sygnatura metody popup, 138

symbol |, 126

symbole wieloznaczne, 214, 565

system

- zabezpieczeń typów, 451
- ETS, 143
- operacyjny, 113, 115

szacowanie wydajności, 426

sztuczki, 163

szukanie błędów, 415

szyfrowanie plików, 349

Ś

ścieżka, 91, 141

do folderu, 616

do funkcji, 127

do konfiguracji, 616

lokalna, 205

UNC, 198, 200, 513

śledzenie

wykonywania skryptu, 531

zmian, 491, 495

środowisko

skryptowe, 121

Windows PowerShell ISE, 561, 563

T

Tabdilio Mark, 267

tablica, 128

\$Error, 252

\$servers, 388

techniki obsługi błędów, 406

technologia

ADO, 208

Adobe Flash, 433

DCOM, 95

Microsoft Silverlight, 433

RPC, 95

WMI, 95, 176

testowanie

aplikacji graficznych, 432

funkcji, 435

interfejsów API, 430

interfejsów użytkownika, 433

podstawowej składni, 411

skryptów, 411, 440

usług sieciowych, 430

własności, 387

wydajności skryptów, 421, 425

zaawansowane, 443

token, 455

transakcja, 178

Truman Jeff, 33

tryb

interaktywny, 584

Set-PSDebug -strict, 542

ściśły, 542

Tsaltas Deae, 273

tworzenie

- aliasów, 122
- aliasów nazw funkcji, 127
- aliasów stałych, 125
- aliasu parametru, 341
- biblioteki funkcji, 153
- dysków PowerShell, 141
- dysku modułu, 313
- dziennika zdarzeń, 519
- dzienników, 431, 441, 515
- egzemplarzy klas, 454
- fragmentów kodu, 569
- funkcji, 125
- grup, 621
- grup zabezpieczeń, 75
- jednostki organizacyjnej, 74
- jednowierszowych komentarzy, 280
- kluczy rejestru, 222
- komentarzy wielowierszowych, 279
- konta komputera, 74
- modułu, 319
- obowiązkowego parametru, 337
- parametrów obowiązkowych, 381
- plików cabinet, 185, 192
- pliku konfiguracji, 481
- pliku MOF, 617
- profilu, 148, 150
- punktów kontrolnych, 606
- punktu wstrzymania, 553
- ścieżki, 135
- tablicy, 386
- tablicy skrótów, 530
- użytkownika, 77, 621
- zadań, 590
- zdalnej sesji, 586
- zmiennych, 134

Tyler Jonathan, 111

typy danych, 403

U

UAC, User Account Control, 389

udziały UNC, 465, 525

umysł nadrzędny, 308

UNC, Universal Naming Convention, 198, 465, 513

unikanie wprowadzania błędów, 490

UPN, User Principal Name, 212

upraszczanie kodu, 425

uprawnienia, 389, 481

- administracyjne, 109, 394, 466
- skryptu, 393

uruchamianie

- przepływu pracy, 599
- skryptu, 417, 475
- zadania, 593

usługa

- Active Directory, 73
- AeLookupSvc, 206
- AWDS, 62
- bits, 157
- DSC, 613
- SkyDrive, 143, 144
- Windows Update, 221
- WinRM, 583, 585
- WMI, 45
- WSMAN, 580

usługi

- domenowe, 82
- sieciowe REST, 431
- sieciowe SOAP, 431

ustalanie punktów kontrolnych

- dla przepływów pracy, 606

ustawianie

- punktu wstrzymania, 550
 - na poleceniu, 552
 - na wierszu kodu, 547
 - na zmiennej, 549
- zabezpieczeń, 468
- zależności, 618

ustawienia pulpitu, 176

ustawienie

- AllSigned, 468
- Bypass, 472
- RemoteSigned, 469, 471
- Restricted, 470
- Undefined, 472
- Unrestricted, 468, 471

usuwanie

- błędów wykonawczych, 527
- fragmentów kodu, 570
- niepotrzebnych informacji, 51
- punktów wstrzymania, 559
- usterek, 491

użytkownik, 77

używanie

- atrybutów parametrów, 339
- CIM, 625
- dzienników, 504, 519

używanie

- filtrów, 266
- fragmentów kodu, 569
- parametrów standardowych, 434
- parsera AST, 455
- pomocy komentarzowej, 281
- potoku, 423
- profilu, 151
- przełącznika debug, 528
- wielu argumentów parametrów, 347

V

VSS, Visual SourceSafe, 496

W

Walker Jason, 170

wartości

- domyślne parametru, 380
- graniczne, 408
- logiczne, 388
- metody popup, 140
- parametrów instrukcji #Requires, 394
- parametru, 409
- reprezentujące ikony, 141
- WMI, 91
- wyliczeniowe, 404

wartość

- Continue, 404
- Inquire, 404
- null, 400, 501
- SilentlyContinue, 404
- Stop, 404

WASP, Web Application Services Platform, 434

WbemTest, 396

wczytywanie danych, 328

wdrażanie

- konsoli, 26
- lokalne, 482
- pakietu MSI, 482
- skryptu, 459
- zasady wykonywania, 465, 469

wersja

- modułu, 395
- platformy .NET, 111
- systemu operacyjnego, 115, 165, 235

weryfikowanie danych, 385

węzeł, node, 614

wielokrotne wykorzystanie kodu, 224, 240

wielowierszowe znaczniki komentarzowe, 279

wiersz poleceń, 28, 214, 328

Wilhite Brian, 46, 64, 202

Willett Andrew, 515

Wilson Ed, 627

Windows PowerShell ISE, 563

Windows Server 2012 R2, 583

WinRM, Windows Remote Management, 583

własności

- klasy __provider, 398, 399
- klasy InvocationInfo, 180
- obiektu ScriptInfo, 181
- statyczne, 182
- użytkownika, 83

własność

- \$filepath, 220
- CLSID, 400
- ProductType, 165
- ScreenSaverTimeout, 178
- ScriptName, 223

włączanie

- konsoli, 32
- obsługi skryptów, 146, 565
- punktów wstrzymania, 558
- trybu ścisłego, 542
- debugera, 550

WMI, Windows Management Instrumentation, 45, 88, 167, 235

Wouters Jeff, 84

wprowadzanie danych do zmiennej globalnej, 372

WSH, Windows Script Host, 406, 496

wstrzymywanie działania skryptu, 558

wybór

- profilu, 148
- zasady wykonywania, 461

wydajność skryptów, 421, 426, 432

wygaszacz ekranu, 177, 178

wyjątek ParameterBindingException, 334

wyjątki

- typu CommandNotFoundExceptions, 392
- w zaporze systemu, 202

wykonywanie

- aktywności, 610
- pojedynczego polecenia, 588
- przepływu pracy, 605, 610
- skryptu, 535, 539–541

wykrywanie

- bieżącego użytkownika, 97
- roli użytkownika, 107

wyłączanie punktów wstrzymania, 558
wymagania

- dotyczące
 - aplikacji, 117
 - bezpieczeństwa, 96
 - modułów, 118, 395
 - systemu operacyjnego, 113
 - uprawnień administratora, 394
 - wersji .NET, 111

przepływów pracy, 598
strukturalne, 96
systemowe, 18

wynik

- audytu, 477
- testu wydajności, 425

wysyłanie

- danych dziennika, 516
- plików tekstowych, 516
- wyników
 - do pliku, 366, 367
 - na adres e-mail, 371
 - na ekran, 360, 367
 - z funkcji, 371

zapytań, 208

wyszukiwanie

- danych, 267
- egzemplarzy klas, 49
- poleceń, 159, 565

wyszukiwarka Bing.com, 433

wyświetlanie

- błędów, 500
- informacji, 42
- monitu, 352
- punktów wstrzymania, 556
- składni polecenia, 568
- własności użytkownika, 83
- wyników, 431

wywołanie

- funkcji, 383
- skryptu logowania, 480

Y

Yoder Jason A., 217

Z

zabezpieczenia przeglądarki, 464

zachowanie zgodności, 491

zadania, 573, 590

zadanie Job10, 590

zakres

- CurrentUser, 472
- grupy, 75
- LocalMachine, 472
- Process, 471
- zasady wykonywania, 147

zalety przepływów pracy, 611

zapisywanie

- błędów, 517
- danych, 421
 - w dzienniku zdarzeń, 517
 - w pliku, 513
 - w pliku tekstowym, 499
 - w rejestrze, 520

hasła, 620

w pliku tekstowym, 350

w rejestrze, 351

w skrypcie, 349

w usługach domenowych, 351

nieprzechwyconych wyników, 262

zapytania, 208

zapytania LDAP, 208

zapytanie Get-EventLog, 477

zarządzanie użytkownikami, 74

zasady

- biznesowe, 528
- grupy, 469, 476
- wykonywania skryptów, 146, 461, 471

zasoby WMI, 167

zastosowania skryptów, 87

zastosowanie

- akceleratora [ADSISeacher], 209
- komentarzy jednowierszowych, 294
- modułu Active Directory, 63

zatrzymywanie usług, 175

zatwierdzanie poleceń, 30

zawieszenie polecenia, 31

zdalna sesja, 586

zdalne

- interaktywne sesje, 96
- komputery, 94
- zarządzanie systemem, 583

zdalny pulpit, 201

zdobywanie informacji, 38

zestaw słów kluczowych, 296

zgodność wersji, 160

zmienianie nazw lokalizacji, 73

zmienna, 134

\$args, 129, 328–335

\$baseLine, 428

\$bios, 368

\$changedFiles, 425

\$clsID, 401

\$computer, 203, 335

\$credential, 355

\$debug, 191

\$DebugPreference, 179, 194, 434, 526

\$driveData, 247

\$error, 251, 333, 413, 509

\$errorActionPreference, 404–407, 412, 504

\$etime, 414

\$files, 422

\$hklm, 91, 92

\$InputObject, 514

\$isAdmin, 108

\$key, 92

\$LastExitCode, 509

\$localappdata, 134

\$makecab, 186

\$MaximumErrorCount, 509

\$modulePath, 311

\$myInvocation.InvocationName, 441

\$noun, 385

\$null, 424

\$number, 372

\$numberOfTests, 428

\$outpath, 77

\$outputPath, 277

\$path, 91, 239, 412

\$ping, 369

\$pinToStart, 32

\$pinToTaskBar, 32

\$profile, 150

\$providerName, 399

\$proxy, 510

\$PSVersionTable, 93

\$report, 412

\$return, 139

\$sourceCab, 191

\$startTime, 413, 504

\$tmpFile, 514

\$trace, 505

\$value, 92

\$verbosePreference, 400

\$wshShell, 91

\$wuLog, 134

docs, 136

path, 286

zmienne globalne, 372

znacznik

[ordered], 530

czasu, 442

description, 510

example, 510

inputs, 510

komentarzowy, 279

notes, 510

outputs, 510

skryptu, 155

specjalny pomocy, 283

synopsis, 510

Zone.Identifier, 464

znaczniki

komentarzy wielowierszowych, 280

parametrów, 337

pomocy funkcji, 282, 283

znajdowanie

dostępnych komputerów, 384

kont użytkowników, 78

metod, 47

modułów, 302

nieużywanych kont użytkowników, 81

wyłączonych użytkowników, 80

znak

#, 419

@, 192

dolara, 247, 329

potoku (|), 295

't, 259

znaki

specjalne, 105

specjalne filtrów, 211, 212

zwracanie danych, 358

Ż

źródła

błędów wykonawczych, 526

informacji, 43, 57, 85, 119, 156, 195, 227, 268,

300, 325, 377, 410, 445, 457, 474, 487, 497,

522, 561, 571, 596, 611, 625

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA



Helion SA

Wykorzystaj potencjał konsoli Windows PowerShell!

PowerShell to naprawdę niezwykła konsola. Pozwala ona kontrolować system Windows i inne aplikacje oraz nimi zarządzać. Jest szczególnie doceniana przez zaawansowanych użytkowników i administratorów. Jeżeli chcesz poznać najlepsze sposoby wykorzystania konsoli PowerShell, ta książka jest dla Ciebie.

Sięgnij po nią i przekonaj się, jakie tajemnice kryje konsola Windows PowerShell oraz jak efektywnie skorzystać z jej możliwości. Poznaj moduł Active Directory, dzięki któremu przygotujesz skrypty pozwalające na wyszukiwanie kont. Naucz się konfigurować środowisko do wykonywania skryptów oraz zwinnie omiń typowe pułapki. Odkryj tajniki projektowania i wdrażania skryptów. Przekonaj się, jak przygotować pomoc do skryptu, opracować moduły oraz obsługiwać błędy. Dowiedz się, jak ustawić zasady wykonywania dla skryptu, uruchomić skrypt oraz kontrolować wersje. Książka ta będzie pożyteczną lekturą dla wszystkich użytkowników korzystających z Windows PowerShell każdego dnia!



Dzięki tej książce:

- poznasz możliwości konsoli Windows PowerShell
- skonfigurujesz środowisko skryptowe
- zaprojektujesz moduły
- przygotujesz system pomocy do skryptów
- wykorzystasz potencjał Windows PowerShell

Ed Wilson — ekspert w dziedzinie skryptów. Certyfikowany trener Microsoftu, prowadzący popularne warsztaty poświęcone Windows PowerShell. Autor książek wydawanych przez Microsoft Press, a poświęconych skryptom w środowisku Windows. Posiada ponad dwadzieścia certyfikatów z branży IT, w tym MCSE i CISSP.

Helion

31294

numer katalogowy

księgarnia Internetowa

<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

Sprawdź najnowsze promocje:
• <http://helion.pl/promocje>
Książki najchętniej czytane:
• <http://helion.pl/bestsellery>
Zamów informacje o nowościach:
• <http://helion.pl/nowosci>

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
<http://helion.pl>

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

ISBN 978-83-283-0478-9



9 788328 304789

Informatyka w najlepszym wydaniu

cena: 99,00 zł