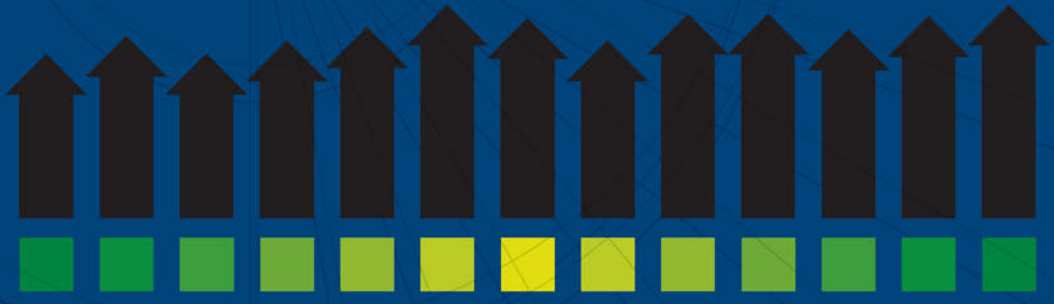
Microsoft

Windows PowerShell



NAJLEPSZE PRAKTYKI

Ed Wilson

Helion

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.

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.

Spis treści

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 VBScrip 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 }
>>
```

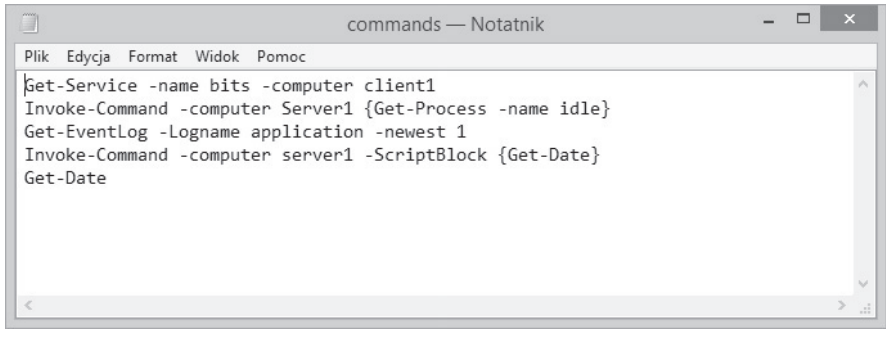
```
Address    statusCode ResponseTime
-------    ---------- ------------
192.168.2.1         0            1
Address    statusCode ResponseTime
-------    ---------- ------------
192.168.2.3         0            2
Address    statusCode ResponseTime
-------    ---------- ------------
192.168.2.5         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.

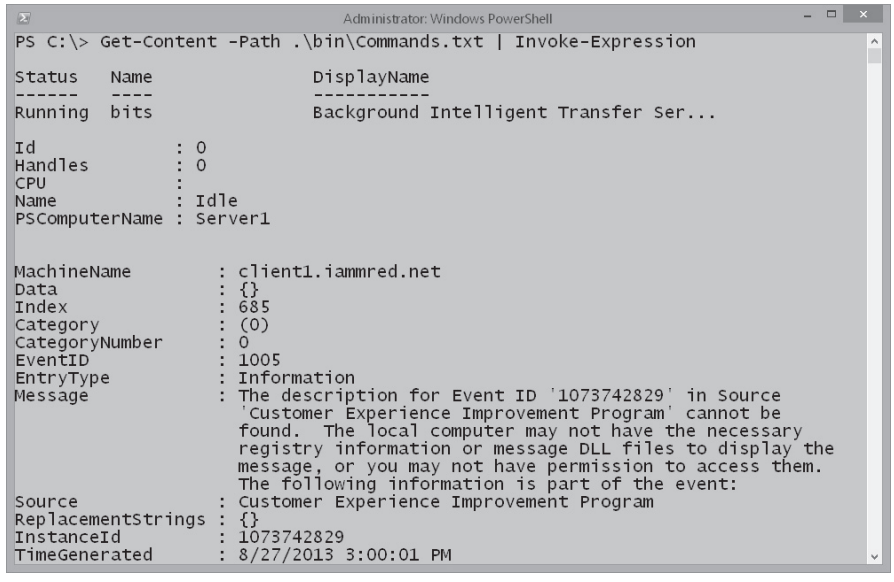


RYSUNEK 7.1. Plik zawierający zbiór poleceń Windows PowerShell

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.



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:
```

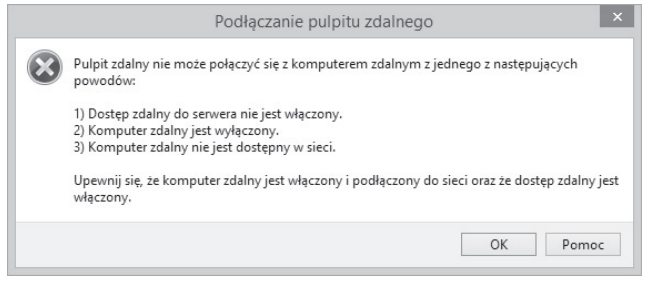
UWAGA

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:

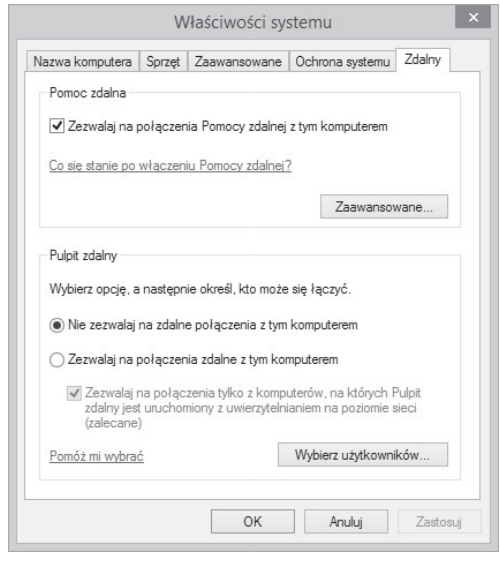
```
PS C:\> invoke-command -ComputerName Sydney.WoodbridgeBank.com -Credential
administrator@WoodBridgeBank.com -ScriptBlock { Get-Content -Path C:\fso\Commands.txt |
Invoke-Expression }
```

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

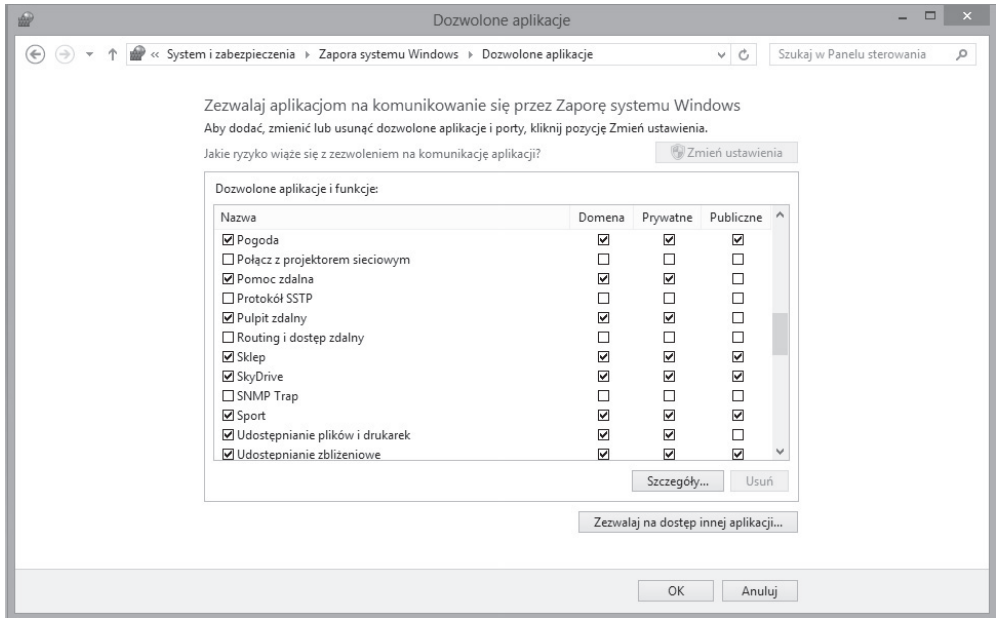
Aby włączyć zdalny pulpit w systemach Microsoft Windows 2012 i Windows 2012 R2, należy w menedżerze serwera w węźle *Serwer lokalny* wybrać opcję *Pulpit zdalny*. W systemach Windows 8 i Windows 8.1 należy w Panelu sterowania otworzyć aplet *System*, kliknąć odnośnik *Ustawienia zdalne*. Pojawi się okno 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 $_}
```

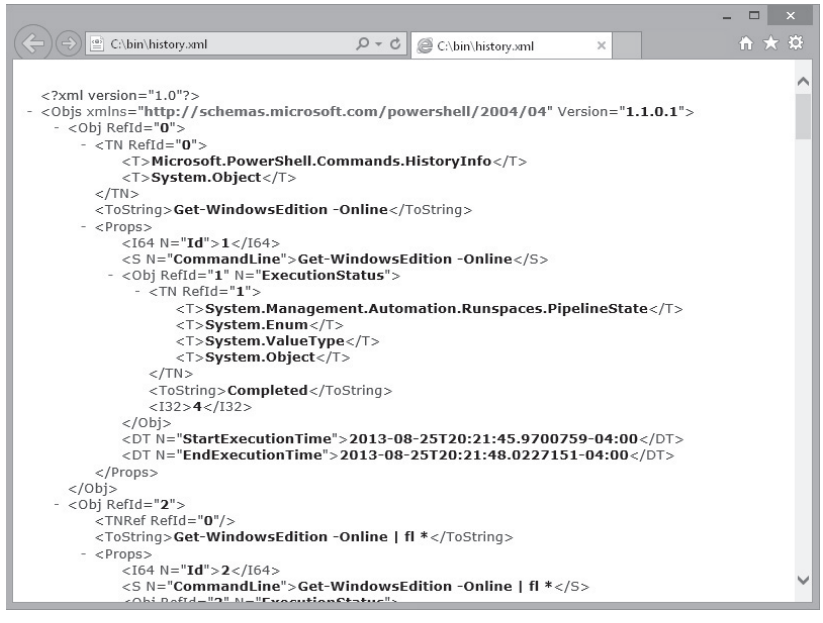
Z własnego doświadczenia wiem, że w takich sytuacjach jak ta, gdy komputerów są setki lub nawet tysiące, bardziej niezawodnym rozwiązaniem jest przekazanie nazw komputerów do polecenia `ForEach-Object` zamiast przekazywania całej listy do parametru `ComputerName`. Oba przedstawione rozwiązania powinny 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:



RYSUNEK 7.6. Plik .xml z historią poleceń

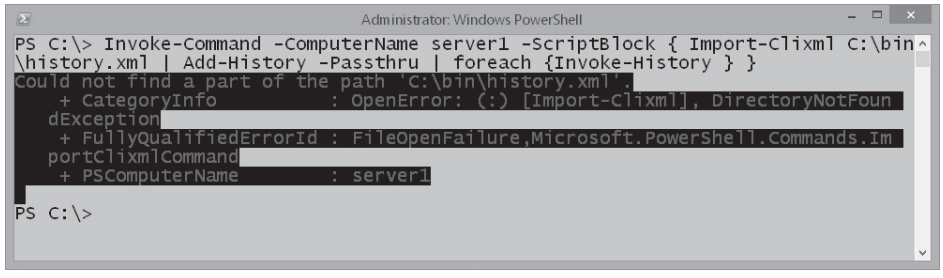
Polecenia z tego pliku można importować za pomocą polecenia `Import-Clixml`. Jego wynik przekazuje się do polecenia `Add-History`, aby polecenia zawarte w zaimportowanym pliku dodać do historii konsoli. Należy użyć przełącznika `-passthru`, aby polecenia przekazać zarówno do polecenia `Add-History`, 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.



RYSUNEK 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 pomieszane 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-vmsrvc           Virtual Machine Additions Services ...
Running  1-vmsrvc           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

dojdziesz 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 Add        Running Windows 8
                    itions Services App
                    lication
1-vmsrvc            Virtual Machine Add        Running berlin
                    itions Services App
                    lication
AeLookupSvc         Application Experie        Running berlin
                    nce
AeLookupSvc         Application Experie        Stopped Windows 8
                    nce
```

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 *DSQeury.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 [ADSISearcher]

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 `[ADSISearcher]`, który jest skrótem nazwy klasy `System.DirectoryServices.DirectorySearcher`. 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 `[ADSISearcher]`, 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 `DirectorySearcher`. 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.DirectorySearcher "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 `[ADSISearcher]`, 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
`ObjectClass=Computer`	Wszystkie obiekty komputerów
`ObjectClass=OrganizationalUnit`	Wszystkie obiekty jednostek organizacyjnych
`ObjectClass=User`	Wszystkie obiekty użytkowników oraz wszystkie obiekty komputerów
`ObjectCategory=User`	Wszystkie obiekty użytkowników
`(&(ObjectCategory=User)(ObjectClass=Person))`	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.

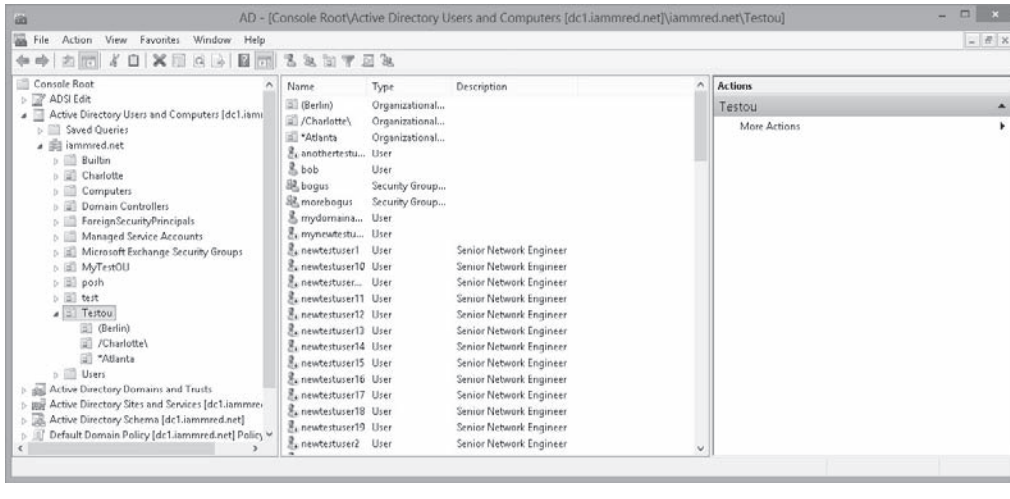
Jak widać na rysunku 7.8, znaki specjalne mogą być używane w nazwach jednostek organizacyjnych w Active Directory.

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:\> ([ADSISearcher]"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 [ADSISearcher], należy użyć poniższego zapytania:

```
PS C:\> ([ADSISearcher]"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 [ADSISearcher] do wysyłania zapytań do katalogu Active Directory niezaufanego lasu lub niezaufanej 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 {([ADSISearcher]"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 zwalniam 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   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   dssenh.dll                C:\Windows\system32\dssenh.dll
Application   dssite.msc                C:\Windows\system32\dssite.msc
Application   dsuiext.dll               C:\Windows\system32\dsuiext.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   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   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   dssenh.dll                     C:\Windows\system32\dssenh.dll
Application   dssite.msc                     C:\Windows\system32\dssite.msc
Application   dsuiext.dll                    C:\Windows\system32\dsuiext.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 `[ADSISearcher]` 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 `[ADSISearcher]`:

```
PS C:\> ([ADSISearcher]"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 `[ADSISearcher]` 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 obarczone 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 `[ADSISearcher]`. 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 jakiekolwiek 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.

Kiedyś 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 to 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 `DerfagAnalysis` 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 wysyłać 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 łącza, 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* łącza 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 łącza 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\wsussnc2.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 = $zipfolder.parsename("$Item") # element w pliku ZIP
    $targetfolder = $ShellObject.namespace("$targetFolder")
    $targetfolder.copyhere($item)
}
```

Za pomocą tej funkcji można wypakować zawartość pliku *wsusscn2.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 *wsusscn2.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 opisania 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:

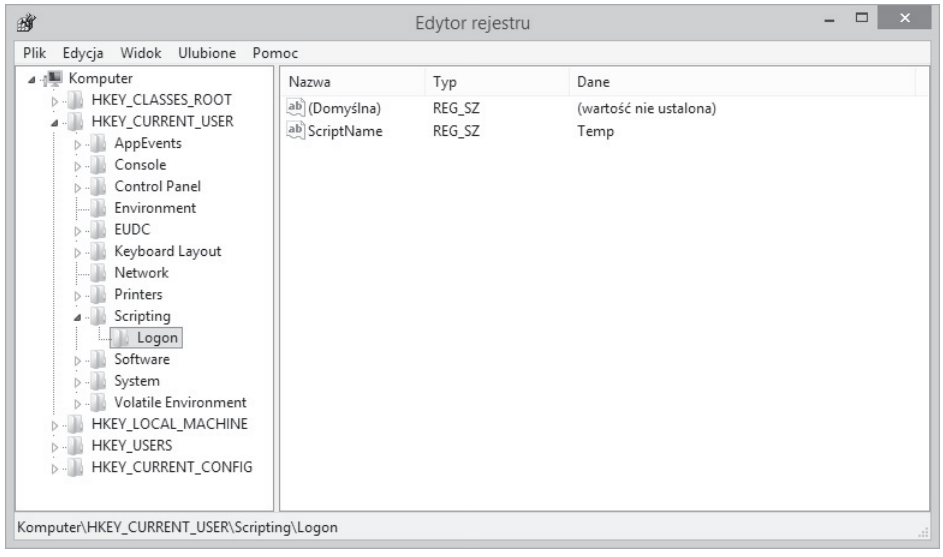
```
PS C:\> New-Item -Path HKCU:\Scripting\Logon -Force

    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.Offie.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 pisaniem 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* 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

C

D

E

F

G

H

I

J

K

N

O

P

R

S

Ś

T

U

V

W

Y

Z

Ź

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

Informatyka w najlepszym wydaniu

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

cena: 99,00 zł