Wprowadzenie

Ta książka zawiera mnóstwo gotowych rozwiązań, związanych z zarządzaniem i konserwacją środowiska Microsoft Exchange 2016 za pomocą programu Windows PowerShell 5.0 oraz Exchange Management Shell. Książka koncentruje się na automatyzacji rutynowych zadań i rozwiązywaniu typowych problemów. Chociaż w powłoce Exchange Management dostępne są setki cmdletów, nie będziemy szczegółowo omawiać każdego z nich. Zamiast tego, skoncentrujemy się na typowych rzeczywistych scenariuszach. Zaprezentowane tu przepisy można natychmiast wykorzystać, aby przyspieszyć wykonywanie swoich zadań. Ponadto, omówione tu techniki pozwolą z łatwością pisać własne wspaniałe jednowierszowe polecenia i skrypty.

Zawartość książki

Rozdział 1., *Kluczowe koncepcje PowerShell*, stanowi wprowadzenie do kilku podstawowych koncepcji środowiska PowerShell, takich jak składnia poleceń i parametrów, korzystanie z potoku, pętle i logika warunkowa. Zagadnienia omówione w tym rozdziale stanowią fundament do zrozumienia przykładowego kodu z kolejnych rozdziałów.

Rozdział 2., *Typowe zadania powłoki Exchange Management*, omawia codzienne zadania oraz ogólne techniki zarządzania środowiskiem Exchange z poziomu wiersza poleceń. W tym rozdziale dowiemy się, jak konfigurować ręczne zdalne połączenia z powłoką, eksportować raporty do plików zewnętrznych, wysyłać wiadomości e-mail za pomocą skryptów oraz planować wykonywanie skryptów za pomocą programu Task Scheduler.

Rozdział 3., *Zarządzanie odbiorcami*, przedstawia niektóre z najbardziej typowych zadań zarządzania odbiorcami, takie jak tworzenie skrzynek pocztowych, grup dystrybucyjnych oraz kontaktów. Nauczymy się zarządzać regułami poczty przychodzącej po stronie serwera oraz ustawieniami nieobecności w pracy, a także importować zdjęcia użytkowników.

Rozdział 4., *Zarządzanie skrzynkami pocztowymi*, prezentuje wykonywanie różnych zadań zarządzania skrzynkami pocztowymi, włącznie z przenoszeniem skrzynek pocztowych, importowaniem i eksportowaniem danych skrzynek pocztowych, a także wykrywaniem i naprawą uszkodzonych skrzynek pocztowych. Ponadto, nauczymy się usuwać i przywracać elementy ze skrzynki pocztowej oraz zarządzać folderami publicznymi.

Rozdział 5., *Grupy dystrybucyjne i listy adresowe*, omawia bardziej zaawansowane aspekty zarządzania grupami dystrybucyjnymi. Nauczymy się tworzyć raporty dotyczące grup dystrybucyjnych oraz zasady nazewnictwa grup dystrybucyjnych, a także umożliwiać użytkownikom na zarządzanie członkostwem w grupach dystrybucyjnych. Dowiemy się też, jak tworzyć listy adresowe i hierarchiczne książki adresowe.

Rozdział 6., *Zarządzanie bazą danych skrzynek pocztowych*, prezentuje konfigurowanie ustawień i limitów baz danych. Ponadto, zawiera przepisy dotyczące generowania raportów o rozmiarach baz danych skrzynek pocztowych, średniej wielkości skrzynki pocztowej w bazie danych oraz stanu kopii zapasowej.

Rozdział 7., *Zarządzanie dostępem klienta*, obejmuje zarządzanie usługami ActiveSync, OWA, POP oraz IMAP, a także konfiguracje tych komponentów w środowisku Exchange 2016. Zajmiemy się także kontrolowaniem połączeń z różnych klientów, włącznie z urządzeniami ActiveSync.

Rozdział 8., *Zarządzanie serwerami transportu*, objaśnia różne metody kontrolowania przepływu poczty w organizacji Exchange. Nauczymy się tworzyć łączniki wysyłania i otrzymywania, umożliwiać przekazywanie poczty przez serwery aplikacji, wyszukiwać dzienniki śledzenia wiadomości oraz zarządzać kolejkami transportu.

Rozdział 9., *Bezpieczeństwo w środowisku Exchange*, wprowadza nowy model uprawnień, czyli kontrolę dostępu opartą na rolach (RBAC). Nauczymy się tworzyć własne role RBAC dla administratorów oraz użytkowników końcowych, zarządzać uprawnieniami do skrzynek pocztowych, a także implementować certyfikaty SSL.

Rozdział 10., *Zgodność i rejestrowanie inspekcji*, obejmuje funkcje zgodności i inspekcji wprowadzone w programie Exchange 2016. Omówione zostały takie zagadnienia, jak rejestrowanie w dzienniku, zapobieganie utracie danych, archiwizowanie skrzynek pocztowych oraz zbieranie elektronicznych materiałów dowodowych, a także rejestrowanie inspekcji administratora i skrzynek pocztowych.

Rozdział 11., *Wysoka dostępność*, obejmuje zadania implementacji i zarządzania grup dostępności baz danych (DAG). Omówiono tworzenie grup DAG, dodawanie kopii baz danych skrzynek pocztowych oraz konserwację członków DAG. Opisano także nową funkcję automatycznej ponownej inicjalizacji.

Rozdział 12., *Monitorowanie kondycji programu Exchange*, omawia sprawdzanie i monitorowanie kondycji środowiska Exchange za pomocą wbudowanych cmdletów

testowych i badania kondycji. Rozdział ten zawiera także informacje o tworzeniu różnych raportów, służących na przykład do monitorowania kolejek poczty oraz nadmiarowości baz danych.

Rozdział 13., *Integracja*, omawia techniki integracji między serwerem Exchange, programem Skype dla firm oraz serwerem Office Online. Ponadto, zawiera informacje o weryfikacji konfiguracji trybu hybrydowego Exchange.

Rozdział 14., *Skrypty wykorzystujące zarządzane API Exchange Web Services*, opisuje zaawansowane techniki tworzenia skryptów wykorzystujących Exchange Web Services. W tym rozdziale nauczymy się pisać skrypty i funkcje wykraczające poza korzystanie z cmdletów powłoki Exchange Management.

Dodatek A, *Ogólne informacje dotyczące powłoki*, zawiera zbiór często wykorzystywanych automatycznych zmiennych powłoki oraz akceleratorów typu, a także listę skryptów wbudowanych w środowisko Exchange 2016. Ponadto, omawia szczegółowo typowe filtrowalne właściwości wspierane przez cmdlety powłoki, które wykorzystują właściwości filtra.

Dodatek B, *Składnie zapytań*, można wykorzystywać jako przewodnik po składni Keyword Query Language (KQL). Można tu znaleźć wiele różnych przykładów, które można wykorzystać w rzeczywistych rozwiązaniach.

Co będzie potrzebne

Aby wykonać ćwiczenia zaprezentowane w tej książce, należy spełnić następujące wymagania:

- Zalecany jest program PowerShell v5, który jest domyślnie zainstalowany w systemach Windows 10 i Windows Server 2016. Większość przepisów można jednak wykonać w programie PowerShell v4.
- Najlepiej, aby serwery Exchange były uruchomione w systemie Windows Server 2016, ale można je też uruchomić w systemie Windows Server 2012 R2.
- W pełni działające środowisko laboratoryjne z lasem Active Directory oraz organizacją Exchange.
- Potrzebny jest co najmniej jeden serwer Microsoft Exchange 2016, ale niektóre zagadnienia, na przykład związane z grupami dostępności baz danych, wymagają dwóch serwerów.
- Zakładamy, że wykorzystywane konto należy do grupy roli Organization Management. Do tej grupy automatycznie dodawane jest konto użytkownika, który zainstalował program Exchange 2016.

- Jeśli to możliwe, polecenia, skrypty i funkcje zaprezentowane w tej książce należy uruchamiać na komputerze klienckim. Warto wybrać 64-bitową wersję systemu Windows 10 z zainstalowanymi narzędziami do zarządzania programem Exchange 2016. Narzędzia te można również uruchomić w systemie Windows 8.1. Każdy klient ma inne wymagania odnośnie uruchamiania tych narzędzi, dlatego szczegółowe informacje na ten temat należy sprawdzić w witrynie TechNet firmy Microsoft.
- Jeśli nie dysponujemy komputerem klienckim, powłokę zarządzania można uruchomić na serwerze Exchange 2016.
- Rozdział 14., *Skrypty wykorzystujące zarządzane API Exchange Web Services*, wymaga zainstalowania pakietu Exchange Web Services Managed API w wersji 2.2, który można pobrać pod adresem `http://www.microsoft.com/en-us/download/details.aspx?id=42951`.

Przykładowy kod zaprezentowany w tej książce należy uruchamiać w środowisku laboratoryjnym i dokładnie przetestować przed wdrożeniem w środowisku produkcyjnym. Jeśli nie mamy skonfigurowanego środowiska laboratoryjnego, możemy pobrać oprogramowanie ze strony `http://technet.microsoft.com/en-us/exchange` i skonfigurować serwery w dowolnym środowisku wirtualnym.

Dla kogo jest ta książka

Ta książka jest przeznaczona dla profesjonalistów, zajmujących się zagadnieniami komunikacji, którzy chcą się nauczyć tworzenia rzeczywistych skryptów w środowisku Windows PowerShell 5.0 oraz Exchange Management Shell. Jeśli jesteś administratorem sieci lub systemów, odpowiedzialnym za zarządzanie i utrzymanie lokalnej wersji programu Exchange Server 2016, ta książka jest dla Ciebie.

Przepisy zaprezentowane w tej książce wykorzystują wszystkie główne role serwera Exchange 2016 i wymagają praktycznej wiedzy o dodatkowych technologiach, takich jak Windows Server 2012 R2 lub 2016, Active Directory i DNS.

Wszystkie zagadnienia poruszane w tej książce koncentrują się na lokalnych wersjach programu Exchange 2016. Jedynym wyjątkiem jest weryfikacja konfiguracji trybu hybrydowego programu Exchange. W tej książce nie będziemy się zajmować wersją programu Exchange Online w pakiecie Office 365, hostowaną przez Microsoft. Jednakże koncepcje poznane w tej książce umożliwią rozpoczęcie pracy z tą platformą, ponieważ dzięki nim zrozumiemy składnię poleceń PowerShell oraz ich obiektową naturę. Ponadto, wiele skryptów i zadań zaprezentowanych w tej książce można także zastosować w środowisku Exchange Online.

Podział na podrozdziały

W tej książce regularnie pojawiają się określone podrozdziały (Przygotowanie, Jak to zrobić, Jak to działa, To nie wszystko i Zobacz też). Są one wykorzystywane w opisany poniżej sposób, aby jak najlepiej przekazać instrukcje dotyczące wykonania ćwiczenia.

Przygotowanie

Ten podrozdział informuje o zagadnieniach poruszanych w przepisie i opisuje konfigurację dodatkowego oprogramowania lub innych ustawień, wymaganych do wykonania ćwiczenia.

Jak to zrobić

Ten podrozdział zawiera kolejne kroki, niezbędne do wykonania ćwiczenia.

Jak to działa

Ten podrozdział zwykle zawiera szczegółowe wyjaśnienia zadań wykonanych w poprzednim podrozdziale.

To nie wszystko

Ten podrozdział zawiera dodatkowe informacje związane z przepisem, aby zaprezentować czytelnikowi więcej szczegółów na dany temat.

Zobacz też

Ten podrozdział zawiera odwołania do innych przydatnych informacji związanych z tym przepisem.

Konwencje

W tej książce stosujemy różne style tekstowe, które umożliwiają odróżnienie różnych rodzajów informacji. Oto kilka przykładów stosowanych stylów oraz wyjaśnienie ich znaczenia.

Fragmenty kodu w tekście wyglądają następująco: „Treść pliku można wczytać do powłoki za pomocą cmdletu `Get-Content`”.

Polecenia i bloki kodu wyglądają następująco:

```
Get-Mailbox -ResultSize Unlimited | Out-File C:\report.txt
```

Tego typu polecenia można wywołać interaktywnie w powłoce lub z poziomu skryptu lub funkcji.

Większość poleceń, jakie prezentujemy w tej książce, jest bardzo długa. Aby zmieścić je na stronach tej książki, korzystamy ze znaku kontynuacji wiersza. Na przykład poniżej przedstawione jest polecenie służące do utworzenia skrzynki pocztowej:

```
New-Mailbox -UserPrincipalName jsmith@contoso.com `
-FirstName John `
-LastName Smith `
-Alias jsmith `
-Database DB1 `
-Password $password
```

Zauważmy, że ostatnim znakiem w każdym wierszu jest grawis (`), zwany również akcentem słabym lub ciężkim. W środowisku PowerShell jest to znak kontynuacji wiersza. Powyższe polecenie można uruchomić bez żadnych modyfikacji, ale należy się upewnić, że na końcu każdego wiersza nie ma żadnych spacji. Można też usunąć grawisy oraz znaki powrotu karetki i uruchomić jednowierszowe polecenie. Należy się jednak upewnić, że zachowane zostaną spacje między parametrami i argumentami.

Niektóre polecenia, wykorzystujące potoki są sformatowane następująco:

```
Get-Mailbox -ResultSize Unlimited |
Select-Object DisplayName, ServerName, Database |
Export-Csv c:\mbreport.csv -NoTypeInformation
```

Znak potoku (|) w środowisku PowerShel służy do wysyłania danych wyjściowych polecenia w potoku, aby można je było wykorzystać jako dane wejściowe innego polecenia. W przypadku znaku potoku nie ma potrzeby stosowania sekwencji ucieczki. Powyższe polecenie można wpisać bez żadnych modyfikacji albo sformatować je do postaci jednego wiersza.

Wszystkie dane wejściowe wiersza poleceń lub dane wyjściowe, które należy uzyskać w sposób interaktywny w konsoli powłoki, są zaprezentowane następująco:

```
[PS] C:\>Get-Mailbox administrator | FT Servername, Database -Auto

ServerName Database
---------- --------
mbx1       DB01
```

Nowa terminologia i **ważne słowa** są oznaczone pogrubioną czcionką. Słowa widoczne na ekranie, na przykład w menu lub oknach dialogowych, są zaprezentowane

w tekście w następujący sposób: „Po kliknięciu przycisku **Next**, zostaniemy przeniesieni do kolejnego ekranu”.

Ostrzeżenia i ważne uwagi są oznaczone w taki sposób.

Wskazówki są oznaczone w ten sposób.

Opinie czytelników

Zawsze jesteśmy wdzięczni za opinie naszych czytelników. Chcemy wiedzieć, co czytelnicy myślą o książce, czy im się podobała, czy nie. Opinie czytelników są dla nas ważne, ponieważ dzięki nim opracowujemy książki, które będą zawierać możliwie najbardziej przydatne treści. Aby wysłać nam ogólną opinię, wystarczy napisać wiadomość e-mail na adres *feedback@packtpub.com* oraz podać tytuł książki w temacie wiadomości. Jeśli czytelnik jest ekspertem w jakiejś dziedzinie i chciałby napisać lub wziąć udział w pisaniu książki, zachęcamy do zapoznania się z przewodnikiem dla autorów, pod adresem *www.packtpub.com/autors*.

Wsparcie klienta

Dla wszystkich dumnych posiadaczy książki wydanej przez Packt przygotowaliśmy różne materiały, które umożliwią w pełni wykorzystać ten zakup.

Pobieranie przykładowego kodu

Pliki przykładowego kodu dla tej książki można pobrać ze strony *http://www.ksiazki.promise.pl/aspx/produkt.aspx?pid=112118*. Po pobraniu plików należy je rozpakować za pomocą najnowszej wersji jednego z poniższych programów:

- WinRAR lub 7-Zip w systemie Windows
- Zipeg, iZip lub UnRarX w systemie Mac
- 7-Zip lub PeaZip w systemie Linux

Przygotowaliśmy także plik PDF, zawierający kolorowe zrzuty ekranowe i diagramy, wykorzystywane w tej książce. Kolorowe ilustracje ułatwiają zrozumienie zmian w danych wyjściowych. Plik ten jest dołączony do kodu przykładowego.

Na stronie *https://github.com/PacktPublishing/* można znaleźć inne pakiety z kodem dla naszego bogatego zbioru książek i wideo. Warto je sprawdzić!

Errata

Chociaż dołożyliśmy wszelkich starań, aby zapewnić dokładność naszych treści, czasem popełniamy błędy. Jeśli znajdziesz błąd w jednej z naszych książek, na przykład literówkę w tekście lub kodzie, będziemy wdzięczni za informację o tym. Dzięki temu zapobiegniesz frustracji innych czytelników i ułatwisz nam ulepszanie kolejnych wersji tej książki. Jeśli znajdziesz jakiś błąd, przejdź na stronę *http://www.packtpub.com/submit-errata*, wybierz tytuł swojej książki, kliknij łącze **Errata Submission Form** i wpisz szczegóły swojej poprawki. Po zweryfikowaniu poprawki i jej zaakceptowaniu zostanie ona opublikowana w naszej witrynie lub dodana do listy dotychczasowych poprawek w sekcji Errata tej książki. Opublikowane wcześniej poprawki można sprawdzić na stronie *http://www.packtoub.com/books/content/support*, na której należy wpisać tytuł książki lub numer ISBN w polu wyszukiwania*. Szukane przez nas informacje będą dostępne w sekcji **Errata**.

Piractwo

Nielegalne użycie materiałów objętych prawami autorskimi w Internecie jest nieustannym problemem we wszystkich mediach. W wydawnictwie Packt bardzo poważnie podchodzimy do kwestii praw autorskich i licencji. Jeśli natkniesz się na nielegalne kopie naszych publikacji w Internecie, prosimy o natychmiastowe poinformowanie nas o adresie lub nazwie witryny, abyśmy mogli podjąć odpowiednie kroki. Prosimy o kontakt pod adresem *copyright@packtpub.com* i podanie łącza do podejrzanych nielegalnych materiałów. Doceniamy pomoc w ochronie naszych autorów i jesteśmy dumni z naszych możliwości w dostarczaniu cennych treści.

Pytania

Jeśli masz dowolne problemy związane z dowolnym aspektem tej książki, możesz skontaktować się z nami pod adresem *questions@packtpub.com*, a dołożymy wszelkich starań, aby rozwiązać ten problem.

---

* Należy użyć tytułu lub numeru ISBN oryginalnego anglojęzycznego wydania książki. Informacje te są dostępne na stronie redakcyjnej (przyp. tłum.).

1

Kluczowe koncepcje PowerShell

W tym rozdziale zostaną omówione następujące zagadnienia:

- Korzystanie z systemu pomocy
- Składnia i parametry poleceń
- Aliasy poleceń
- Konfigurowanie profilu w środowisku PowerShell
- Koncepcja potoku
- Korzystanie ze zmiennych i obiektów
- Korzystanie z tablic i tablic skrótów
- Przetwarzanie elementów w pętli
- Tworzenie niestandardowych obiektów
- Korzystanie z funkcji debugera
- Nowe zasady wykonywania
- Korzystanie z funkcji Save-Help
- Korzystanie z repozytoriów skryptów

Wstęp

W naszej firmie zapadła decyzja o przeniesieniu się na serwer Exchange Server 2016, który oferuje wiele nowych ciekawych funkcji, takich jak zintegrowana archiwizacja poczty elektronicznej, możliwości wyszukiwania oraz wysokiej dostępności. Czy się to nam podoba, czy nie, zdajemy sobie sprawę, że PowerShell jest niezbędnym narzędziem do zarządzania serwerem Exchange Server i chcielibyśmy poznać podstawy jego obsługi, a także uzyskać materiały pomocne podczas pisania własnych skryptów. Do tego służy ta książka. W tym rozdziale zostaną omówione niektóre najważniejsze koncepcje dotyczące środowiska PowerShell. Będziemy z nich korzystać podczas wykonywania pozostałych ćwiczeń zawartych w tej książce. Nawet doświadczeni użytkownicy środowiska PowerShell mogą potraktować ten rozdział jako przegląd zagadnień lub materiał, do którego można się odwoływać podczas samodzielnego pisania skryptów.

Osoby, które dopiero rozpoczynają przygodę z programem PowerShell, mogą zauważyć, że środowisko to przypomina powłoki poleceń systemu UNIX. Podobnie jak powłoki oparte na systemie UNIX, program PowerShell umożliwia łączenie wielu poleceń w jednym wierszu, za pomocą tzw. potoków. W tej technice dane wyjściowe jednego polecenia są przekazywane jako dane wejściowe do kolejnego polecenia. Jednak w przeciwieństwie do powłok wykorzystywanych w systemie UNIX, w których między poleceniami przekazywane są dane tekstowe, w programie PowerShell wykorzystywany jest model obiektowy, oparty na platformie .NET Framework. W tym przypadku w potoku przekazywane są obiekty, a nie zwykły tekst. Z perspektywy programu Exchange, korzystając z obiektów mamy możliwość uzyskania bardzo szczegółowych informacji o serwerach, skrzynkach pocztowych, bazach danych i innych. Posłużmy się przykładem skrzynki pocztowej. Każda skrzynka zarządzana z poziomu powłoki jest obiektem z kilkoma właściwościami, takimi jak adres e-mail, lokalizacja bazy danych lub ograniczenia ilości wysyłanych i otrzymywanych danych. Dzięki możliwości uzyskania tego typu informacji za pomocą prostych poleceń, możemy z łatwością tworzyć potężne skrypty generujące raporty, modyfikujące konfigurację i wykonujące zadania związane z utrzymaniem.

W tej książce wykorzystujemy wersję 5.1 środowiska PowerShell oraz system Windows 2016 Server w wersji 5.1 i kompilacji 14393.

Podstawowe kroki

Aby wykonać przykłady opisane w tym rozdziale, wykonaj poniższe czynności, służące do uruchomienia programu Exchange Management Shell:

1. Zaloguj się na stacji roboczej lub na serwerze, na którym zainstalowane są narzędzia zarządzania w programie Exchange.
2. Jeśli okaże się, że narzędzia te nie są zainstalowane, możesz uzyskać połączenie korzystając ze zdalnej wersji programu PowerShell. W tym celu wykonaj poniższe polecenie:

```
$Session = New-PSSession -ConfigurationName Microsoft.Exchange `
-ConnectionUri http://nazwa_serwera/PowerShell/ `
-Authentication Kerberos
Import-PSSession $Session
```

3. Exchange Management Shell można też uruchomić klikając przycisk Windows i wybierając opcję **Microsoft Exchange Server 2016/Exchange Management Shell**.

Pamiętajmy, że aby uniknąć problemów związanych z uprawnieniami, program Exchange Management Shell należy uruchamiać korzystając z funkcji **Run as Administrator** (Uruchom jako administrator). W przykładowych cmdletach, zaprezentowanych w tym rozdziale, użyto znaku `, aby podzielić długie polecenia na kilka wierszy. Ma to na celu zwiększenie czytelności poleceń. Używanie znaków ` nie jest konieczne i należy z nich korzystać tylko wtedy, gdy są niezbędne. Zauważmy, że w przypadku korzystania z tej techniki nie są dostępne zmienne programu Exchange, takie jak `$exscript`.

Korzystanie z systemu pomocy

W programie Exchange Management Shell mamy do dyspozycji ponad 830 cmdletów, z których każdy przyjmuje zestaw wielu parametrów. Na przykład cmdlet `New-Mailbox` przyjmuje ponad 60 parametrów, natomiast `Set-Mailbox` ma około 200 parametrów. Nie będzie przesadą stwierdzenie, że nawet najbardziej zaawansowany ekspert programu PowerShell byłby na straconej pozycji, gdyby nie miał do dyspozycji porządnego systemu pomocy. W tym ćwiczeniu dowiemy się, jak uzyskać pomoc dotyczącą programu Exchange Management Shell.

Jak to zrobić

Aby uzyskać informacje dotyczące cmdletu, należy wpisać polecenie `Get-Help`, a następnie nazwę cmdletu. Wykonaj poniższe polecenie, aby uzyskać pomoc dotyczącą cmdletu `Get-Mailbox`:

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

Jak to działa

Po uruchomieniu polecenia `Get-Help` z nazwą cmdletu, w powłoce zostanie wyświetlone streszczenie i opis cmdletu. Cmdlet `Get-Help` jest jednym z najlepszych narzędzi umożliwiających poznawanie tajników programu PowerShell. Można z niego korzystać w razie wątpliwości, dotyczących działania lub parametrów cmdletu.

Informacje konkretnego typu można uzyskać za pomocą poniższych przełączników cmdletu `Get-Help`:

- **`Detailed`** – umożliwia uzyskanie opisów parametrów wraz z przykładami. Jego składnia jest następująca:

  ```
  Get-Help <nazwa_cmdletu> -Detailed
  ```

- **`Examples`** – umożliwia uzyskanie wielu przykładów użycia cmdletu. Jego składnia jest następująca:

  ```
  Get-Help <nazwa_cmdletu> -Examples
  ```

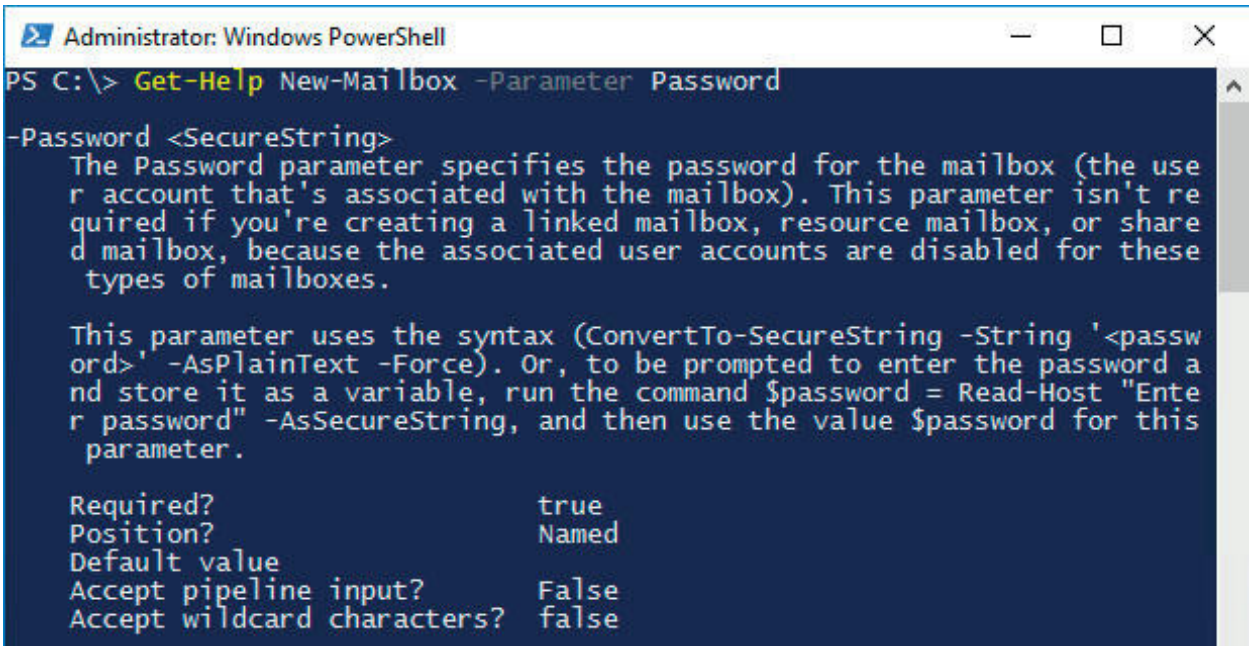
- **`Full`** – umożliwia wyświetlenie całej treści pliku pomocy dotyczącej określonego cmdletu, na przykład:

  ```
  Get-Help <nazwa_cmdletu> -Full
  ```

- **`Online`** – umożliwia wyświetlenie pliku pomocy online, dotyczącej określonego cmdletu. Jego składnia jest następująca:

  ```
  Get-Help <nazwa_cmdletu> -Online
  ```

Niektóre parametry przyjmują dane wejściowe w postaci zwykłych łańcuchów tekstowych, natomiast do innych należy przekazać obiekt. Jeśli tworzymy skrzynkę pocztową za pomocą cmdletu `New-Mailbox`, do parametru `-Password` musimy przekazać obiekt bezpiecznego łańcucha. Typ danych wymagany przez parametr można sprawdzić za pomocą cmdletu `Get-Help`:



W danych wyjściowych powyższego polecenia znajduje się kilka kluczowych informacji dotyczących parametru `-Password`. Oprócz wymaganego typu danych `<SecureString>`, możemy zauważyć, że jest to parametr nazwany. Jest on wymagany podczas uruchomienia cmdletu `New-Mailbox` i nie przyjmuje symboli wieloznacznych. Za pomocą cmdletu `Get-Help` możemy sprawdzić, czy ustawienia te są wspierane przez parametry dowolnego cmdletu.

W czasie pisania tej książki w środowisku Exchange Management Shell występowały problemy z wykonywaniem polecenia `Get-Help New-Mailbox`. Istnieje jednak nieoficjalne obejście tego problemu. W zwykłym środowisku PowerShell systemu Windows należy najpierw wykonać polecenie `Add-PSSnapin -Name Microsoft.Exchange.Management.PowerShell.E2010`, a następnie można już uruchomić cmdlet `Get-Help New-Mailbox`. Jest to znany problem, który zostanie rozwiązany w przyszłości.

Za pomocą polecenia `Get-Help New-Mailbox -Examples` można określić składnię służącą do utworzenia obiektu bezpiecznego łańcucha hasła oraz uzyskać informacje o jego użyciu w celu utworzenia skrzynki pocztowej. Ta technika jest szczegółowo opisana w podrozdziale *Tworzenie, modyfikowanie i usuwanie skrzynek pocztowych*, w rozdziale 3., *Zarządzanie odbiorcami*.

To nie wszystko

Czasem potrzebny jest nam cmdlet, którego pełnej nazwy nie znamy. W tej sytuacji można skorzystać z kilku poleceń, które umożliwią znalezienie potrzebnego cmdletu.

Aby znaleźć wszystkie cmdlety, których nazwa zawiera słowo „mailbox", można skorzystać z symbolu wieloznacznego, jak w poniższym poleceniu:

```
Get-Command *Mailbox*
```

Za pomocą parametru `-Verb` można znaleźć wszystkie cmdlety, których nazwa rozpoczyna się od określonego czasownika:

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

Za pomocą parametru `-Noun` można znaleźć wszystkie cmdlety, których nazwa zawiera określony rzeczownik:

```
Get-Command -Noun Mailbox
```

Cmdlet `Get-Command` jest poleceniem wbudowanym w program PowerShell, które zwraca polecenia dostępne zarówno w programie Windows PowerShell, jak i Exchange Management Shell. W powłoce Exchange Management Shell dostępne jest też specjalne polecenie `Get-Ex`, które zwraca polecenia dostępne jedynie w środowisku Exchange.

Polecenie `Get-Help` nie tylko umożliwia uzyskanie pomocy na temat cmdletów, ale również pozwala na przeglądanie dodatkowych plików pomocy, które opisują ogólne koncepcje powłoki PowerShell, dotyczące głównie tworzenia skryptów. Aby wyświetlić plik pomocy, dotyczący określonego zagadnienia, należy wpisać polecenie `Get-Help about_`, a następnie nazwę zagadnienia. Na przykład, aby uzyskać informacje dotyczące podstawowych poleceń dostępnych w powłoce PowerShell, należy wykonać następujące polecenie:

```
Get-Help about_Core_Commands
```

Listę wszystkich plików pomocy, opisujących koncepcje powłoki można uzyskać za pomocą następującego polecenia:

```
Get-Help about_*
```

Nie ma potrzeby zapamiętywania nazw wszystkich cmdletów programu Exchange lub PowerShell. Wystarczy pamiętać o poleceniach `Get-Command` i `Get-Help`, za pomocą których można wyszukać nazwy i uzyskać informacje o składni poleceń służących do wykonania dowolnych zadań.

Uzyskiwanie pomocy na temat cmdletów i funkcji

Początkujący użytkownik może mieć trudności w odróżnieniu cmdletów od funkcji. Podczas uruchamiania programu Exchange Management Shell, na serwerze Exchange uruchamiana jest zdalna sesja programu PowerShell, a do lokalnej sesji powłoki importowane są specyficzne polecenia, zwane funkcjami proxy. Ogólnie rzecz biorąc, są to bloki kodu, które mają nazwę, na przykład `Get-Mailbox`. Odpowiadają one skompilowanym cmdletom zainstalowanym na serwerze. Dotyczy to również sytuacji, gdy mamy do dyspozycji jeden serwer i korzystamy z lokalnej sesji powłoki uruchomionej na tym samym serwerze.

Gdy uruchomimy funkcję `Get-Mailbox` z poziomu powłoki, poprzez zdalną sesję PowerShell przekazywane są dane między komputerem lokalnym a serwerem Exchange. Cmdlet `Get-Mailbox` jest w rzeczywistości uruchamiany na zdalnym serwerze Exchange, a uzyskane dane wyjściowe są przekazywane do komputera lokalnego. Zaletą takiego rozwiązania jest możliwość zdalnego uruchamiania cmdletów, niezależnie od tego, czy serwery znajdują się na miejscu, czy też w chmurze.

W pozostałych rozdziałach tej książki poznamy więcej szczegółów tej techniki. Na razie należy zapamiętać, że podczas pracy z systemem pomocy cmdlety programu Exchange 2016 będą opisywane jako funkcje, a nie jako cmdlety.

Rozważmy następujące polecenie oraz uzyskane dane wyjściowe:

```
Machine: tlex01.testlabs.se
[PS] C:\>Get-Command Get-Service

CommandType     Name                                               Version
-----------     ----                                               -------
Cmdlet          Get-Service                                        3.1.0.0
```

W tym przypadku polecenie `Get-Command` dotyczy cmdletu powłoki PowerShell w wersji 5. Zauważmy, że w polu `CommandType` widoczny jest typ `Cmdlet`.

Wykonajmy tę samą operację w przypadku cmdletu `Get-Mailbox`:

```
Machine: tlex01.testlabs.se
[PS] C:\>Get-Command Get-Mailbox

CommandType     Name                                               Version
-----------     ----                                               -------
Function        Get-Mailbox                                        1.0
```

Jak widać, w polu `CommandType` dla cmdletu `Get-Mailbox` widnieje typ `Function`, czyli jest to funkcja. Na podstawie tych przykładów można wysnuć kilka wniosków. Po pierwsze, w tej książce będziemy nazywać cmdlety programu Exchange 2016

cmdletami, nawet jeśli po wykonaniu polecenia `Get-Command` ich typ zostanie określony jako funkcja. Po drugie, pamiętajmy, że możemy wykonać polecenie `Get-Help` dla dowolnej nazwy funkcji, na przykład `Get-Mailbox`, aby uzyskać plik pomocy dla odpowiedniego cmdletu. Jeśli nie mamy pewności odnośnie dokładnej nazwy cmdletu, za pomocą polecenia `Get-Command` możemy ułatwić sobie zadanie i przeprowadzić wyszukiwanie z użyciem symbolu wieloznacznego. Gdy już określimy nazwę szukanego cmdletu, możemy wykonać dla niego polecenie `Get-Help`, aby uzyskać szczegółowe informacje o jego użyciu.

Zanim zaczniemy szukać odpowiedzi w Internecie, spróbujmy najpierw skorzystać z systemu pomocy. Okaże się, że odpowiedzi na większość pytań zostały opisane we wbudowanej pomocy cmdletu.

Zobacz też

- Przepis *Składnia i parametry poleceń* w tym rozdziale.
- Przepis *Ręczne konfigurowanie zdalnych połączeń PowerShell* w rozdziale 2., *Typowe zadania powłoki Exchange Management*.
- Przepis *Korzystanie z kontroli dostępu opartej na rolach* w rozdziale 9., *Bezpieczeństwo w środowisku Exchange*.

Składnia i parametry poleceń

W programie Windows PowerShell mamy do dyspozycji ogromną liczbę wbudowanych cmdletów, służących do wykonania określonych zadań. W programie Exchange Management Shell dostępne są dodatkowe cmdlety, przeznaczone do zarządzania programem Exchange. Można je uruchamiać zarówno z poziomu powłoki, jak i używać w zautomatyzowanych skryptach. Podczas uruchamiania cmdletu można skorzystać z parametrów i przekazać za ich pośrednictwem dodatkowe informacje, takie jak skrzynka pocztowa lub serwer, z którego należy skorzystać. W ten sposób można też określić, które atrybuty obiektów należy zmodyfikować. W tym ćwiczeniu poznamy podstawową składnię polecenia powłoki PowerShell oraz sposób używania parametrów z cmdletami.

Jak to zrobić

Aby uruchomić polecenie PowerShell, należy najpierw wpisać nazwę cmdletu, a następnie potrzebne parametry. Nazwy parametrów są poprzedzone kreską (`-`). Następnie podawana jest wartość parametru. Zacznijmy od prostego przykładu. Aby

uzyskać informacje o skrzynce pocztowej użytkownika o nazwie `testuser`, należy wykonać następujące polecenie:

```
Get-Mailbox -Identity testuser
```

Te same informacje można uzyskać również za pomocą poniższego polecenia, ponieważ parametr `-Identity` ma charakter pozycyjny:

```
Get-Mailbox testuser
```

Większość cmdletów przyjmuje wiele parametrów, które można użyć jednocześnie w jednym poleceniu.

Za pomocą poniższego przykładowego polecenia można zmodyfikować dwa różne ustawienia skrzynki pocztowej użytkownika `testuser`:

```
Set-Mailbox testuser -MaxSendSize 50Mb -MaxReceiveSize 50Mb
```

Jak to działa

Nazwy wszystkich cmdletów są zgodne ze standardową konwencją czasownik-rzeczownik. Aby na przykład uzyskać listę wszystkich skrzynek pocztowych, należy skorzystać z cmdletu `Get-Mailbox`. Konfigurację skrzynki pocztowej można zmienić za pomocą cmdletu `Set-Mailbox`. W obydwu przykładach czasownik (`Get` lub `Set`) określa akcję, którą chcemy wykonać na rzeczowniku (`Mailbox`). Czasownik jest zawsze połączony z rzeczownikiem za pomocą znaku kreski (`-`). Z wyjątkiem kilku cmdletów programu Exchange Management Shell, rzeczowniki zawsze występują w liczbie pojedynczej.

Wielkość liter w nazwach cmdletów i parametrów nie ma znaczenia. Aby zwiększyć czytelność skryptów można używać kombinacji wielkich i małych liter, ale nie jest to wymagane.

Dane wejściowe parametru są opcjonalne lub wymagane, w zależności od parametru i cmdletu. Nie ma konieczności podawania wartości do parametru `-Identity`, ponieważ w przypadku cmdletu `Get-Mailbox` nie jest ona wymagana. Jeśli po prostu uruchomimy cmdlet `Get-Mailbox` bez żadnych argumentów, uzyskamy listę pierwszych 1000 skrzynek pocztowych w firmie.

Jeśli w naszym środowisku zdefiniowanych jest ponad 1000 skrzynek pocztowych, można uruchomić cmdlet `Get-Mailbox` z parametrem `-ResultSize` o wartości `Unlimited`. W ten sposób uzyskamy listę wszystkich skrzynek pocztowych w firmie.

Zauważmy, że w dwóch pierwszych przykładach wykonaliśmy polecenie `Get-Mailbox` dla jednego użytkownika. W pierwszym przykładzie użyliśmy parametru `-Identity`, natomiast w drugim nie. Ponieważ `-Identity` jest parametrem pozycyjnym, nie musimy go wpisywać. W tym przypadku parametr `-Identity` znajduje się na pierwszej pozycji, dlatego pierwszy argument przekazany do tego cmdletu jest automatycznie przypisywany do tego parametru. Cmdlet może wspierać kilka parametrów pozycyjnych, które są numerowane od 1. Parametry, które nie są pozycyjne to parametry nazwane. W tym przypadku, aby przekazać do nich wartość, najpierw należy podać ich nazwę.

Parametr `-Identity` jest rozpoznawany przez większość cmdletów programu Exchange Management Shell. Umożliwia zaklasyfikowanie obiektu, na którym należy wykonać określoną operację.

Parametr `-Identity`, wykorzystywany przez cmdlety programu Exchange Management Shell, może przyjmować różne typy wartości. Oprócz aliasu, można skorzystać z następujących typów: ADObjectID, nazwa wyróżniająca, domena\nazwa_użytkownika, GUID, LegacyExchangeDN, SmtpAddress i UserPrincipalName (UPN).

Jeśli uruchomimy cmdlet bez podania wartości wymaganego parametru, zostaniemy poproszeni o podanie niezbędnych informacji przed wykonaniem cmdletu. Wynika to z tego, że cmdlet musi wiedzieć, którą skrzynkę pocztową chcemy zmodyfikować.

Aby określić, które parametry są wymagane, nazwane lub pozycyjne, które wspierają symbole wieloznaczne lub akceptują dane wejściowe z potoku, można wykonać cmdlet `Get-Help`, który zostanie omówiony w kolejnym podrozdziale. Jako dane wejściowe można wykorzystać wiele typów danych, w zależności od parametru. Niektóre parametry przyjmują łańcuchy, natomiast inne wymagają liczb całkowitych lub wartości logicznych. Parametry przyjmujące wartości logiczne są używane, gdy chcemy przekazać wartość prawda lub fałsz. PowerShell udostępnia dla tych wartości wbudowane zmienne powłoki. Są to odpowiednio zmienne automatyczne `$true` i `$false`. Listę wszystkich zmiennych automatycznych dostępnych w programie PowerShell w wersji 5.x można uzyskać wykonując polecenie `Get- Helpabout_automatic_variables`. W *Dodatku A* dostępna jest lista dodatkowych zmiennych automatycznych, dostępnych w programie Exchange Management Shell.

Aby na przykład włączyć lub wyłączyć łącznik wysyłania, należy użyć cmdletu `Set-SendConnector` z parametrem `-Enabled`:

```
Set-SendConnector Internet -Enabled $false
```

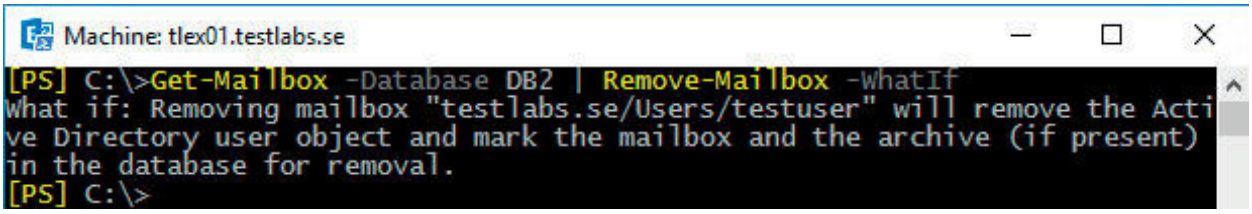
Parametry przełącznika nie wymagają podawania wartości. Są one wykorzystywane do włączania lub wyłączania funkcji lub ustawienia. Jednym z typowych przykładów użycia przełącznika jest tworzenie archiwum skrzynki pocztowej użytkownika:

```
Enable-Mailbox testuser -Archive
```

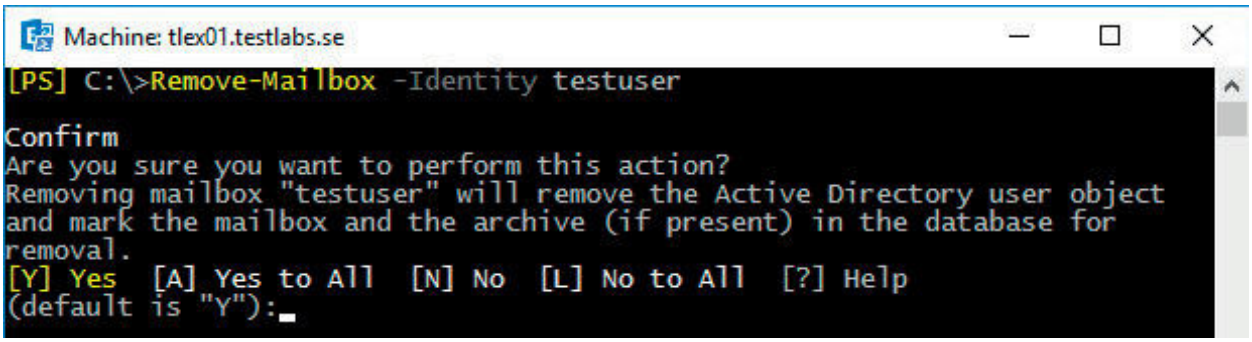
W programie PowerShell mamy do dyspozycji zestaw wspólnych parametrów, które można używać z każdym cmdletem. Niektóre wspólne parametry, takie jak parametry służące do ograniczenia ryzyka (`-Confirm` i `-Whatif`), działają tylko z cmdletami, które dokonują zmian.

Pełna lista wspólnych parametrów jest dostępna za pośrednictwem polecenia `Get-Help about_CommonParameters`.

Parametry mające na celu ograniczenie ryzyka umożliwiają uzyskanie podglądu wprowadzonych zmian lub wymagają potwierdzenia zmiany, która może mieć destrukcyjne działanie. Jeśli przed wykonaniem polecenia chcemy sprawdzić, co się stanie po jego wykonaniu, możemy skorzystać z parametru `-WhatIf`:



Podczas wprowadzania zmian, takich jak usuwanie skrzynki pocztowej, zostaniemy poproszeni o potwierdzenie naszych zamiarów, jak na poniższym rysunku:



Aby pominąć konieczność potwierdzenia, należy ustawić parametr `-Confirm` na `false`:

```
Remove-Mailbox testuser -Confirm:$false
```

Warto zauważyć, że jeśli do parametru `-Confirm` przekazujemy zmienną `$false`, tuż po nazwie parametru musimy wpisać dwukropek, a następnie wartość logiczną. Różni się to od przekazywania wartości za pomocą parametru `-Enabled` i cmdletu `Set-SendConnector`. Pamiętajmy, że parametr `-Confirm` zawsze wymaga użycia tej specjalnej składni. Większość parametrów przyjmujących wartości logiczne nie ma takich wymagań, chociaż zależy to od uruchamianego cmdletu. Na szczęście w programie PowerShell dostępny jest wspaniały wbudowany system pomocy, z którego można skorzystać, aby rozwiać podobne wątpliwości.

Cmdlety i parametry wspierają uzupełnianie po naciśnięciu tabulatora. Wystarczy rozpocząć wpisywanie kilku pierwszych znaków nazwy cmdletu lub parametru, a następnie nacisnąć tabulator, aby automatycznie uzupełnić nazwę lub wybrać nazwę z listy. Ta technika świetnie sprawdza się podczas odkrywania nowych poleceń i pozwala zaoszczędzić nieco czasu.

Ponadto, wystarczy wpisać tylko tyle znaków nazwy parametru, aby odróżnić ją od nazwy innego parametru. W poniższym poleceniu częściowa nazwa parametru jest całkowicie poprawna:

```
Set-User -id testuser -Office Sales
```

W tym przykładzie ciąg `id` jest skrótem parametru `-Identity`. Powyższy cmdlet nie przyjmuje innych parametrów rozpoczynających się od znaków `id`, dlatego automatycznie zakłada, że chodzi nam o parametr `-Identity`.

Inną przydatną cechą niektórych parametrów jest wsparcie dla symboli wieloznacznych. Podczas uruchamiania cmdletu `Get-Mailbox` parametr `-Identity` może przyjmować symbole wieloznaczne. Dzięki temu możemy uzyskać listę wielu skrzynek pocztowych, których nazwy pasują do podanego wzorca:

```
Get-Mailbox -id t*
```

W powyższym przykładzie uzyskamy listę wszystkich skrzynek pocztowych, których nazwy rozpoczynają się od litery `t`. Chociaż ta technika wydaje się dość prosta, szczegółowe informacje dotyczące symboli wieloznacznych stosowanych w powłoce PowerShell można uzyskać korzystając z systemu pomocy. W tym celu należy wykonać polecenie `Get-Helpabout_Wildcards`.

Skrótowy zapis cmdletów może się świetnie sprawdzić podczas wykonywania interaktywnych zadań administracyjnych. Jednak by zapewnić bezproblemowe działanie skryptów w przyszłości, zaleca się korzystanie z pełnej składni cmdletów.

To nie wszystko

Jeśli chcemy przekazać do parametru wartość, która zawiera spacje, musimy ją ująć w cudzysłowy lub apostrofy. W poniższym przykładzie chcemy uzyskać listę wszystkich skrzynek pocztowych, należących do jednostki organizacyjnej `Sales Users` w usłudze Active Directory. Należy zauważyć, że nazwa jednostki organizacyjnej została umieszczona między apostrofami, ponieważ zawiera spację:

```
Get-Mailbox -OrganizationalUnit 'testlabs.se/Sales Users/Seattle'
```

Aby rozwinąć zmienną znajdującą się w łańcuchu, należy skorzystać z cudzysłowów:

```
$City = 'Seattle'
Get-Mailbox -OrganizationalUnit "testlabs.se/Sales Users/$City"
```

W powyższym przykładzie najpierw tworzymy zmienną zawierającą nazwę miasta, które reprezentuje podjednostkę organizacyjną należącą do jednostki `Sales Users`. Następnie, w cmdlecie `Get-Mailbox` umieszczamy tę zmienną w łańcuchu określającym nazwę jednostki organizacyjnej. PowerShell automatycznie rozwinie nazwę zmiennej umieszczonej w łańcuchu ujętym w cudzysłowy i umieści ją w odpowiednim miejscu. Dzięki temu uzyskamy listę wszystkich skrzynek pocztowych, przydzielonych do jednostki organizacyjnej `Seattle`.

Reguły cytowania są szczegółowo udokumentowane w systemie pomocy programu PowerShell. Można je uzyskać wykonując polecenie `Get-Help about_Quoting_Rules`.

Zobacz też

- Przepis *Korzystanie z systemu pomocy* w tym rozdziale.
- Przepis *Korzystanie ze zmiennych i obiektów* w tym rozdziale.

Aliasy poleceń

W tej i w poprzednich książkach pisaliśmy o aliasach poleceń lub cmdletów, na przykład o skrótowej nazwie cmdletu.

Aliasy mogą być bardzo przydatne podczas pisania skryptów, które chcemy zoptymalizować i możliwie skrócić ich zawartość.

Przyjrzyjmy się kilku przykładom wbudowanych aliasów. Na początek zajmijmy się często używanym cmdletem `Where-Object`. Dla tego cmdletu zdefiniowano dwa aliasy, `?` oraz `where`. Innym przykładem często używanego cmdletu jest `ForEach-Object`; dla tego cmdletu zdefiniowano dwa aliasy, `%` oraz `foreach`. Ostatnim przykładem wbudowanego aliasu jest `select`, zdefiniowany dla cmdletu `Select-Object`.

Jak to zrobić

Przy uruchamianiu polecenia PowerShell, zamiast wpisywać pełną nazwę cmdletu, można skorzystać z jego aliasu. Rozważmy kilka przykładów korzystania z aliasów:

```
Get-Alias
New-Alias -Name wh -Value Write-Host
wh "Testowanie aliasu"
New-Alias -Name list -Value Get-ChildItem
list -Path "C:\Scripts"
New-Alias -Name npp -Value "C:\Program Files `
(x86)\Notepad++\notepad++.exe"
npp
```

Jak to działa

Zaczęliśmy od cmdletu `Get-Alias`, za pomocą którego można uzyskać listę wszystkich zdefiniowanych aliasów. Jeśli nie zostały jeszcze zdefiniowane żadne niestandardowe aliasy, lista będzie zawierać tylko aliasy wbudowane.

W powyższych przykładach tworzymy trzy różne aliasy. Aliasy można zdefiniować dla cmdletów, z których często korzystamy lub jeśli chcemy uzyskać możliwość uruchamiania aplikacji za pomocą aliasu, bezpośrednio z poziomu wiersza poleceń PowerShell.

W pierwszym przykładzie definiujemy alias `wh` dla polecenia `Write-Host`. Tego typu alias można użyć w poleceniu `wh "Testowanie aliasu"`, które uruchomi cmdlet `Write-Host` i wyświetli tekst na ekranie.

W drugim przykładzie utworzyliśmy alias o nazwie `list` dla polecenia `Get-ChildItem`. Przykładem użycia jest polecenie `list -Path "C:\Scripts"`, które zwraca listę wszystkich plików i folderów znajdujących się w folderze `C:\Scripts`.

W trzecim i ostatnim przykładzie tworzymy alias służący do uruchamiania aplikacji Notepad++. W tym celu przypisujemy do aliasu pełną ścieżkę do pliku .exe aplikacji. Po zdefiniowaniu aliasu możemy uruchomić program Notepad++ wykonując polecenie `npp` w wierszu poleceń PowerShell.

Zobacz też

- Przepis *Konfigurowanie profilu w środowisku PowerShell* w tym rozdziale.
- Przepis *Korzystanie ze zmiennych i obiektów* w tym rozdziale.
- Przepis *Przetwarzanie elementów w pętli* w tym rozdziale.

Konfigurowanie profilu w środowisku PowerShell

Za pomocą profilu PowerShell można dostosować środowisko powłoki, wczytać funkcje, moduły, aliasy i zmienne podczas uruchamiania sesji programu Exchange Management Shell. W tym przykładzie dowiemy się, jak utworzyć profil.

Jak to zrobić

Profile nie są tworzone domyślnie, ale warto zweryfikować, czy profil nie został już utworzony.

1. Zacznijmy od uruchomienia cmdletu `Test-Path`.

   ```
   Test-Path $profile
   ```

2. Jeśli cmdlet `Test-Path` zwróci wartość `$true`, oznacza to, że dla bieżącego użytkownika został już utworzony profil. Istniejący profil można otworzyć uruchamiając `notepad.exe` z poziomu powłoki.

   ```
   notepad $profile
   ```

3. Jeśli cmdlet `Test-Path` zwróci wartość `$false`, można utworzyć nowy profil dla bieżącego użytkownika, wykonując następujące polecenie:

   ```
   New-Item -type file -path $profile -force
   ```

Jak to działa

Profil PowerShell jest zwykłym skryptem z rozszerzeniem `.ps1`, który jest wykonywany zawsze podczas uruchamiania powłoki. Możemy przyjąć, że profil jest czymś w rodzaju skryptu logowania dla sesji programu PowerShell lub Exchange Management Shell. W profilu można dodać własne aliasy, zdefiniować zmienne, wczytać moduły

lub dodać własne funkcje, które będą dostępne zawsze po uruchomieniu powłoki. W poprzednim przykładzie użyliśmy automatycznej zmiennej powłoki `$profile`, aby utworzyć skrypt profilu dla bieżącego użytkownika, który w tym przypadku zostanie umieszczony w katalogu `$env:UserProfile\Documents\WindowsPowerShell\`.

Ponieważ powłoka PowerShell wczytuje profil użytkownika po prostu wykonując skrypt `.ps1`, zasady wykonywania muszą zezwalać na wykonywanie skryptów na tym komputerze. W przeciwnym razie profil nie zostanie wczytany podczas uruchamiania powłoki i zostanie zwrócony błąd.

W powłoce PowerShell można używać czterech rodzajów profili:

- **`$Profile.AllUsersAllHosts`**: Ten profil dotyczy wszystkich użytkowników i powłok. Znajduje się w pliku `$env:Windir\system32\WindowsPowerShell\v1.0\profile.ps1`
- **`$Profile.AllUsersCurrentHost`**: Ten profil dotyczy wszystkich użytkowników oraz tylko bieżącego hosta programu PowerShell.exe. Znajduje się w pliku `$env:Windir\system32\WindowsPowerShell\v1.0\Microsoft.PowerShell_profile.ps1`
- **`$Profile.CurrentUserAllHosts`**: Ten profil dotyczy bieżącego użytkownika i wszystkich powłok. Znajduje się w pliku `$env:UserProfile\Documents\WindowsPowerShell\profile.ps1`
- **`$Profile.CurrentUserCurrentHost`**: Ten profil dotyczy bieżącego użytkownika i hosta programu PowerShell.exe. Znajduje się w pliku `$env:UserProfile\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1`

Jeśli do utworzenia profilu użyjemy tylko zmiennej `$profile`, profil zostanie domyślnie umieszczony w miejscu wskazywanym przez zmienną `CurrentUserCurrentHost`. Prawdopodobnie jest to najczęściej wykorzystywany typ profilu. Jeśli chcemy utworzyć profil dla wszystkich użytkowników na komputerze, należyć skorzystać z jednego z typów *AllUsers*.

Być może niektórzy zastanawiają się teraz, na czym polega różnica między typami profilu `CurrentHost` i `AllHosts`. Środowisko uruchomieniowe PowerShell może się znajdować w aplikacjach innych firm, dlatego profil `AllHosts` dotyczy także tych instancji programu PowerShell. Typy profilu `CurrentHost` można używać z programem PowerShell.exe oraz Exchange Management Shell.

Oprócz definiowania własnych aliasów lub funkcji w profilu, można rozważyć wczytanie innych przydatnych modułów. Można na przykład wczytać moduł Active Directory w powłoce PowerShell, aby po uruchomieniu powłoki mieć pod ręką cmdlety z tego modułu.

Po wprowadzeniu zmian w profilu, należy go zapisać i zamknąć plik. Aby zmiany zaczęły obowiązywać, należy ponownie uruchomić powłokę, lub wczytać skrypt za pomocą notacji kropki, aby ponownie załadować profil:

```
.$profile
```

Można też utworzyć wiele plików `.ps1`, zawierających aliasy, funkcje i zmienne, a następnie wczytać je za pomocą kropki w skrypcie profilu. Dzięki temu wszystkie zostaną załadowane podczas każdego uruchomienia sesji PowerShell.

Odpowiednim przykładem może być użycie cmdletu `Import-Module` w skrypcie profilu, aby zaimportować moduły programu PowerShell w momencie uruchamiania powłoki Windows PowerShell lub Exchange Management Shell. W skrypcie profilu można też wczytać funkcje, moduły i inne skrypty PS1.

Za pomocą polecenia `Get-Helpabout_profiles` można uzyskać szczegółowe informacje dotyczące omawianego zagadnienia.

To nie wszystko

Próba zapamiętania wszystkich typów profili i ścieżek odpowiadających im skryptów może być dość trudna. Dlatego warto skorzystać z dość ciekawej sztuczki, wykorzystującej zmienną `$profile` do wyświetlenia wszystkich typów profili i ścieżek do plików. W tym celu należy uzyskać dostęp do właściwości `psextended` obiektu `$profile`:

```
$profile.psextended | Format-List
```

W ten sposób uzyskamy listę każdego typu profilu oraz ścieżkę skryptu `.ps1`, z którego należy skorzystać podczas tworzenia profilu.

Zobacz też

- Przepis *Aliasy poleceń* w tym rozdziale
- Przepis *Korzystanie ze zmiennych i obiektów* w tym rozdziale

Koncepcja potoku

Najważniejszą koncepcją w programie PowerShell jest użycie elastycznego, obiektowego potoku. Niektórzy zapewne używali już potoków w powłokach opartych na systemie UNIX lub podczas korzystania z wiersza poleceń programu `cmd.exe`. Koncepcja potoków jest podobna, ponieważ we wszystkich wspomnianych przypadkach polega na przekazywaniu danych wyjściowych jednego polecenia do kolejnego. Ale zamiast przekazywania zwykłego tekstu, PowerShell wykorzystuje obiekty, dzięki czemu

za pomocą jednego wiersza kodu możemy wykonać niekiedy bardzo skomplikowane zadania. W tym ćwiczeniu nauczymy się używać potoków w celu łączenia ze sobą wielu poleceń i tworzenia potężnych jednowierszowych poleceń.

Jak to zrobić

Za pomocą poniższego potoku można skonfigurować lokalizację biura każdej skrzynki pocztowej w bazie danych DB2:

```
Get-Mailbox -Database DB2 | Set-User -Office "Headquarters"
```

Jak to działa

Serię poleceń w potoku oddzielamy od siebie operatorem potoku, czyli znakiem `|`. W powyższym przykładzie cmdlet `Get-Mailbox` zwraca kolekcję obiektów skrzynek pocztowych. Każdy obiekt skrzynki pocztowej zawiera kilka właściwości, które przechowują takie informacje jak nazwa i inne. Cmdlet `Set-User` został opracowany w taki sposób, że może w potoku przyjmować dane zwracane przez cmdlet `Get-Mailbox`. Dzięki temu jednym prostym poleceniem możemy przekazać całą kolekcję skrzynek pocztowych, które następnie możemy zmodyfikować za pomocą jednej operacji.

Można też przekierować dane wyjściowe do poleceń filtrujących, takich jak cmdlet `Where-Object`. W tym przykładzie polecenie umożliwia uzyskanie tylko skrzynek pocztowych, których właściwość `aMaxSendSize` ma wartość 50 megabajtów:

```
Get-Mailbox | Where-Object{$_.MaxSendSize -eq 50mb}
```

Kod wykorzystywany przez cmdlet `Where-Object` do przeprowadzenia filtrowania jest ujęty w nawiasy klamrowe (`{}`). Taka konstrukcja nosi nazwę bloku skryptu, natomiast zawarty w tym bloku kod jest wykonywany na każdym obiekcie przekazanym w potoku. Jeśli wynik przeprowadzonej operacji ma wartość `true`, wówczas obiekt jest zwracany; w przeciwnym razie obiekt zostanie zignorowany. W tym przykładzie dostęp do właściwości `MaxSendSize` każdej skrzynki pocztowej uzyskujemy za pomocą `obiektu $_`, który jest automatyczną zmienną, zawierającą bieżący obiekt w potoku. Za pomocą operatora przyrównania (`-eq`) sprawdzamy, czy właściwość `MaxSendSize` każdej skrzynki pocztowej jest równa 50 megabajtów. Jeśli tak, polecenie zwróci tylko te skrzynki pocztowe.

Operatory porównania umożliwiają porównywanie wyników i znajdowanie wartości pasujących do wzorca. Pełną listę operatorów porównania można uzyskać za pomocą polecenia `Get-Help about_Comparison_Operators`.

Gdy uruchomimy to jednowierszowe polecenie, każdy obiekt skrzynki pocztowej zostanie przetworzony po kolei, w procesie przetwarzania strumieniowego. Oznacza to, że gdy tylko zostanie znalezione dopasowanie, informacje o skrzynce pocztowej zostaną wyświetlone na ekranie. Gdyby nie takie zachowanie, wyniki pojawiłyby się na ekranie dopiero po dłuższym czasie, po znalezieniu wszystkich skrzynek pocztowych. Nie ma to większego znaczenia, jeśli nasze środowisko jest bardzo małe, ale jeśli pracujemy w ogromnym przedsiębiorstwie, w którym zdefiniowane są dziesiątki tysięcy skrzynek pocztowych, musielibyśmy czekać bardzo długo na przetworzenie wszystkich obiektów i zwrócenie przefiltrowanej kolekcji.

Warto zwrócić uwagę na operację porównywania wykonywaną przez filtr `Where-Object`, w której wykorzystywany jest przyrostek mnożnika `mb`. PowerShell natywnie wspiera tego typu mnożniki, które znacznie ułatwiają pracę z wielkimi liczbami. W tym przykładzie użyliśmy wartości `50mb`, która odpowiada wpisaniu wartości w megabajtach, ponieważ za kulisami PowerShell zastąpi tę wartość następującą `1024*1024*50`. PowerShell oferuje następujące mnożniki: `kb`, `mb`, `gb`, `tb` i `pb`.

To nie wszystko

Za pomocą zaawansowanych technik potoków można przesyłać obiekty do innych cmdletów, które nie oferują bezpośredniego wsparcia dla danych wejściowych przekazywanych w potoku. Na przykład, poniższe jednowierszowe polecenie umożliwia dodanie listy użytkowników do grupy:

```
Get-User |
  Where-Object{$_.title -eq "Exchange Admin"} | Foreach-Object{
      Add-RoleGroupMember -Identity "Organization Management" `
      -Member $_.name
  }
```

Powyższe polecenie rozpoczyna się od prostego filtra, który zwraca jedynie użytkowników, których właściwość `Title` ma wartość `Exchange Admin`. Dane wyjściowe tego polecenia są następnie przekazywane w potoku do cmdletu `ForEach-Object`, który przetwarza każdy obiekt znajdujący się w kolekcji. Podobnie do cmdletu `Where-Object`, cmdlet `ForEach-Object` przetwarza każdy element z potoku za pomocą bloku skryptu. Zamiast operacji filtrowania, tym razem wykonujemy polecenie na każdym obiekcie znajdującym się w kolekcji i dodajemy go do grupy `Organization Management`.

Użycie aliasów w potokach może się przydać, ponieważ zmniejsza liczbę znaków, które należy wpisać. Poniżej znajduje się poprzednie polecenie, jednak zapisane z użyciem aliasów:

```
Get-User |
  ?{$_.title -eq "Exchange Admin"} | %{
    Add-RoleGroupMember -Identity "Organization Management" `
    -Member $_.name
  }
```

Zwróćmy uwagę na znak zapytania (`?`) oraz znak procenta (`%`). Znak `?` jest aliasem cmdletu `Where-Object`, natomiast znak `%` jest aliasem cmdletu `ForEach-Object`. Wspomniane cmdlety są bardzo często wykorzystywane i zwykle w postaci aliasów, ponieważ ułatwia to ich wpisywanie.

Za pomocą cmdletu `Get-Alias` można znaleźć wszystkie aliasy, które są zdefiniowane w bieżącej sesji powłoki. Natomiast, aby utworzyć własne aliasy można wykorzystać cmdlet `New-Alias`.

Cmdlety `Where-Object` i `ForEach-Object` mają dodatkowe aliasy. Oto jeszcze jeden sposób zapisu poprzedniego polecenia:

```
Get-User |
  where{$_.title -eq "Exchange Admin"} | foreach{
    Add-RoleGroupMember -Identity "Organization Management" `
    -Member $_.name
  }
```

Korzystajmy z aliasów podczas interaktywnej sesji w powłoce, aby przyspieszyć pracę i zapewnić spójność poleceń. W skryptach produkcyjnych należy rozważyć użycie pełnych nazw cmdletów, aby uniknąć dezorientacji innych osób, które mogą mieć dostęp do kodu.

Zobacz też

- Przepis *Przetwarzanie elementów w pętli* w tym rozdziale
- Przepis *Tworzenie niestandardowych obiektów* w tym rozdziale

Korzystanie ze zmiennych i obiektów

Każdy język skryptowy wykorzystuje zmienne do przechowywania danych. PowerShell nie jest w tym przypadku wyjątkiem. Bardzo często musimy wykorzystywać zmienne w celu zapisania danych tymczasowych w obiekcie, aby można z nich było skorzystać w przyszłości. Program PowerShell znacznie różni się od innych powłok poleceń, ponieważ wszystko jest w rzeczywistości obiektem z wieloma właściwościami i metodami. W programie PowerShell każda zmienna jest po prostu instancją obiektu, tak

samo jak wszystkie inne elementy. Właściwości obiektu zawierają różnego rodzaju fragmenty informacji, w zależności od typu obiektu, z którym pracujemy. W tym przykładzie dowiemy się, jak tworzyć własne zmienne i przetwarzać obiekty w programie Exchange Management Shell.

Jak to zrobić

Aby utworzyć zmienną, przechowującą instancję skrzynki pocztowej `testuser`, wykonaj następujące polecenie:

```
$mailbox = Get-Mailbox testuser
```

Jak to działa

Aby utworzyć zmienną lub instancję obiektu, należy poprzedzić nazwę zmiennej znakiem dolara (`$`). Z prawej strony nazwy zmiennej umieszczamy znak równości (`=`), czyli operator przypisania. Następnie umieszczamy wartość lub obiekt, który chcemy przypisać do zmiennej. Pamiętajmy, że utworzone zmienne są dostępne tylko w bieżącej sesji powłoki i zostaną usunięte w momencie zakończenia sesji.

Rozważmy kolejny przykład. Aby utworzyć zmienną w postaci łańcucha, zawierającą adres e-mail, wykonaj następujące polecenie:

```
$email = testuser@contoso.com
```

Oprócz zmiennych zdefiniowanych przez użytkownika, w programie PowerShell dostępne są też zmienne automatyczne i preferencji. Więcej informacji na ich temat można uzyskać za pomocą poleceń `Get-Help about_Automatic_Variables` i `Get-Help about_Preference_Variables`.

Nawet zmienna zawierająca zwykły łańcuch jest obiektem z właściwościami i metodami. Na przykład, każdy łańcuch ma właściwość `Length`, która zwraca liczbę znaków w łańcuchu:

```
[PS] C:\>$email.length

20
```

Jeśli chcemy uzyskać dostęp do właściwości obiektu, możemy skorzystać z notacji kropki, która umożliwia odwoływanie się do potrzebnej właściwości. W tym celu należy wpisać nazwę obiektu, kropkę, a następnie nazwę właściwości, jak w poprzednim przykładzie. W ten sam sposób można uzyskać dostęp do metod, pamiętając jednak, że nazwy metod zawsze kończą się nawiasami `()`.

Typ danych string oferuje kilka metod, takich jak `Substring`, `Replace` i `Split`. W poniższym przykładzie zaprezentowano, jak podzielić łańcuch za pomocą metody `Split`:

```
[PS] C:\>$email.Split("@")

testuser
contoso.com
```

Jak widać, metoda `Split` wykorzystuje znak `@` jako separator i zwraca dwa łańcuchy powstałe w wyniku podziału.

W powłoce PowerShell dostępny jest też operator `-Split`, który umożliwia podział łańcucha na jedną lub kilka części. Szczegółowe informacje na ten temat można uzyskać wykonując polecenie `Get-Help about_Split`.

To nie wszystko

Wiemy już, jak uzyskać dostęp do właściwości i metod obiektu, ale musimy jeszcze wiedzieć, jak sprawdzać ich dostępność i jak z nich korzystać. Aby określić, które właściwości i metody są dostępne dla bieżącego obiektu, można wykorzystać cmdlet `Get-Member`, który obok poleceń `Get-Help` i `Get-Command` jest jednym z kluczowych narzędzi, umożliwiających odkrywanie możliwości programu PowerShell.

Aby uzyskać informacje o elementach obiektu, należy przekazać obiekt w potoku do cmdletu `Get-Member`. Poniższe polecenie umożliwia uzyskanie wszystkich elementów instancji obiektów `$mailbox`, który wcześniej utworzyliśmy:

```
$mailbox | Get-Member
```

Aby przefiltrować wyniki zwrócone przez polecenie `Get-Member`, skorzystajmy z parametru `-MemberType`, określając, czy chcemy uzyskać informacje o typie `Property` lub `Method`.

Rozważmy praktyczny przykład użycia cmdletu `Get-Member` w celu uzyskania informacji o metodach obiektu. Wyobraźmy sobie, że każda skrzynka pocztowa w naszym środowisku ma niestandardową wartość ustawienia `MaxSendSize` i chcemy zarejestrować tę wartość w celu utworzenia raportu. Gdy odczytamy wartość właściwości `MaxSendSize`, uzyskamy następujące informacje:

```
[PS] C:\>$mailbox.MaxSendSize
```

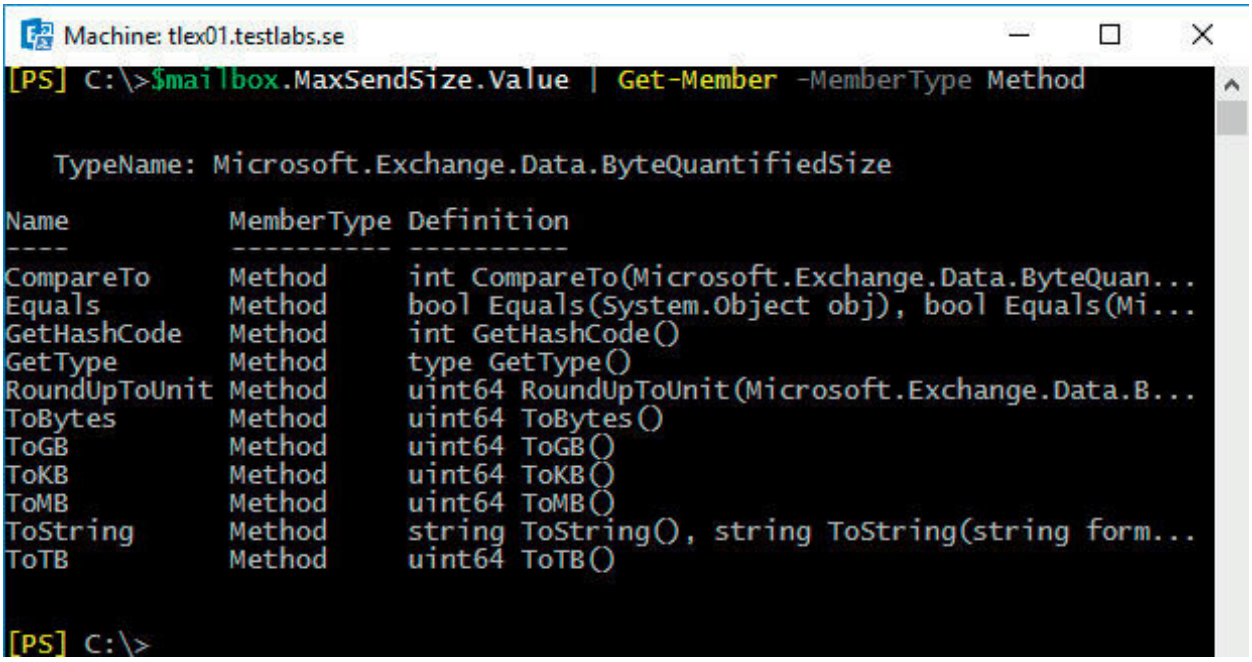
```
IsUnlimited Value
----------- -----
False       50 MB (52,428,800 bytes)
```

Widać, że właściwość `MaxSendSize` w rzeczywistości zawiera obiekt z dwiema właściwościami: `IsUnlimited` i `Value`. Dzięki posiadanej już wiedzy możemy za pomocą notacji z kropką uzyskać dostęp do informacji przechowywanych we właściwości `Value`:

```
[PS] C:\>$mailbox.MaxSendSize.Value

50 MB (52,428,800 bytes)
```

Ta technika działa, ale zwrócone informacje zawierają nie tylko wartość w megabajtach, ale też całkowitą liczbę bajtów przechowywaną we właściwości `MaxSendSize`. Nam w tym przykładzie potrzebna jest tylko całkowita wartość megabajtów. Sprawdźmy, czy ten obiekt oferuje metodę, która może się nam przydać. W tym celu skorzystajmy z polecenia `Get-Member`:



Na podstawie danych wyjściowych, widocznych na powyższej ilustracji, możemy zauważyć, że obiekt oferuje kilka metod, za pomocą których można dokonać konwersji wartości. Aby uzyskać wartość właściwości `MaxSendSize` w megabajtach, możemy wywołać metodę `ToMB`:

```
[PS] C:\>$mailbox.MaxSendSize.Value.ToMB()

50
```

W tradycyjnej, uruchomionej lokalnie powłoce programu Exchange, aby uzyskać tego typu informacje, konieczne byłoby wykonanie skomplikowanego przetwarzania łańcucha. Jednak w powłoce PowerShell oraz na platformie .NET Framework zadanie to jest proste. Z czasem przyznamy, że jest to jeden z powodów, dlaczego natura obiektowa powłoki PowerShell naprawdę przyćmiewa typowe powłoki poleceń oparte na przetwarzaniu tekstu.

W powyższym przykładzie należy zwrócić uwagę na ważną rzecz. Otóż polecenie to nie zadziała, jeśli dla skrzynki pocztowej nie zdefiniowano niestandardowego ograniczenia wartości `MaxSendSize`. Dotyczy to nowych skrzynek pocztowych utworzonych w programie Exchange 2016.

Niemniej jednak jest to świetny przykład procesu, jaki można wykorzystać podczas próby określenia właściwości lub metod obiektu.

Rozwijanie zmiennych w łańcuchach

Jak wspomniano w pierwszym podrozdziale, PowerShell wykorzystuje reguły cytowania, aby określić, jak należy traktować zmienne znajdujące się w cytowanym łańcuchu. Gdy prosta zmienna znajdzie się w łańcuchu ujętym w cudzysłowy, PowerShell rozwinie zmienną i zastąpi jej nazwę wartością łańcucha. Sprawdźmy jak to działa, na podstawie prostego przykładu:

```
[PS] C:\>$name = "Jan"
[PS] C:\> "Użytkownik ma na imię $name"

Użytkownik ma na imię Jan
```

To dość proste. Do zmiennej `$name` przypisaliśmy napis `Jan`. Następnie umieściliśmy zmienną `$name` w łańcuchu ujętym w cudzysłowy, który zawiera treść komunikatu. Gdy naciśniemy Enter, zmienna `$name` zostanie rozwinięta i na ekranie pojawi się oczekiwany komunikat.

Wypróbujmy tę technikę na bardziej złożonym obiekcie. Załóżmy, że chcemy przypisać instancję obiektu skrzynki pocztowej do zmiennej i uzyskać dostęp do właściwości `PrimarySmtpAddress` w cytowanym łańcuchu:

```
[PS] C:\>$mailbox = Get-Mailbox testuser
[PS] C:\>"Adres e-mail jest następujący: $mailbox.PrimarySmtpAddress"

Adres e-mail jest następujący: test user.PrimarySmtpAddress
```

Zauważmy, że gdy próbujemy uzyskać dostęp do właściwości `PrimarySmtpAddress` obiektu skrzynki pocztowej w łańcuchu ujętym w cudzysłowy, nie uzyskujemy pożądanej informacji. Bardzo często spotykamy się z tego typu problemem, kiedy próbujemy przetwarzać obiekty i właściwości w łańcuchach. Możemy temu zaradzić,

korzystając z notacji podwyrażenia. W tym celu należy ująć cały obiekt umieszczony w łańcuchu w znaki `$()`.

```
[PS] C:\>"Adres e-mail jest następujący: $($mailbox.PrimarySmtpAddress)"

Adres e-mail jest następujący: testuser@testlabs.se
```

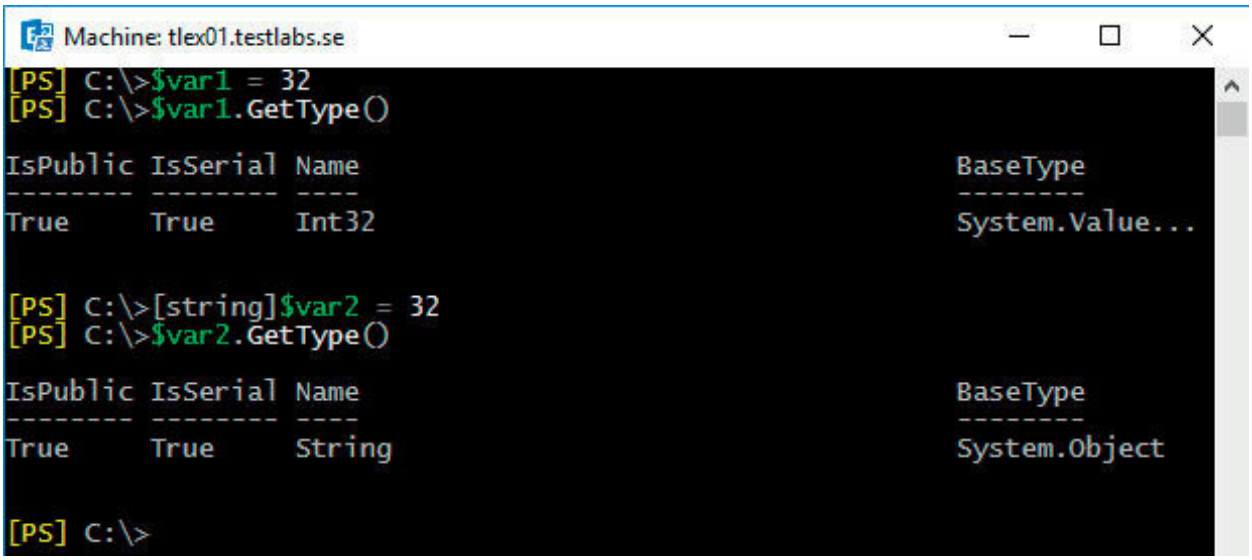
Po zastosowaniu powyższej składni, właściwość `PrimarySmtpAddress` obiektu `$mailbox` zostanie poprawnie rozwinięta i uzyskamy poprawne informacje. Ta technika przyda się nam także podczas wyodrębniania danych z obiektów i generowania raportów lub tworzenia plików dziennika.

Silnie typizowane zmienne

PowerShell automatycznie podejmie próbę wyboru poprawnego typu danych dla zmiennej, na podstawie przypisanej do niej wartości. Nie musimy się tym przejmować, ale możemy też w razie potrzeby samodzielnie jawnie określić typ zmiennej. W tym celu należy określić typ danych w nawiasach kwadratowych przed nazwą zmiennej:

```
[string]$var2 = 32
```

W powyższym przykładzie przydzielamy wartość `32` do zmiennej `&var2`. Gdybyśmy jawnie nie podali typu zmiennej za pomocą skrótu typu `[string]`, zmienna `$var2` uzyskałaby typ danych `Int32`, ponieważ przydzielamy do niej wartość liczbową, która nie została ujęta w apostrofy ani w cudzysłowy. Zerknijmy na poniższą ilustrację:



Jak widać, zmienna `$var1` została utworzona bez jawnego określania typu. Za pomocą metody `GetType()`, którą można wykonać na dowolnym obiekcie wykorzystywanym w powłoce, określamy typ zmiennej `$var1`. Ponieważ przydzielona wartość jest liczbą,

której nie ujęliśmy w cudzysłowy, jej typ został ustalony jako `Int32`. W przypadku zmiennej `$var2` użyliśmy skrótu typu `[string]`, przydzielając do niej tę samą wartość, dzięki czemu przypisane do niej dane mają typ łańcucha.

Warto poświęcić nieco czasu na zrozumienie typów danych, ponieważ podczas tworzenia skryptów zwracających obiekty możemy potrzebować pewnej kontroli nad typami. Możemy na przykład potrzebować informacji o ilości wolnego miejsca na dysku serwera Exchange. Jeśli przypiszemy tę wartość w postaci łańcucha do właściwości własnego obiektu, utracimy możliwość sortowania na jej podstawie. Z tej techniki korzystamy w kilku przykładach w tej książce.

W *Dodatku A* i *B* znajduje się opis skrótów dla często wykorzystywanych typów.

Korzystanie z tablic i tablic skrótów

Podobnie jak w wielu innych językach skryptowych i programowania, Windows PowerShell umożliwia korzystanie z tablic i tablic skrótów. Tablica jest kolekcją wartości, które można przechowywać w jednym obiekcie. Tablica skrótów jest również znana pod nazwą tablicy asocjacyjnej. Jest to słownik przechowujący zestaw par klucz-wartość. Warto dobrze opanować koncepcję tablic, ponieważ za ich pomocą można w wydajny sposób zarządzać wieloma obiektami jednocześnie i uzyskać maksymalną wydajność w korzystaniu z powłoki. W tym ćwiczeniu dowiemy się, jak przechowywać dane w obydwu typach tablic oraz jak je z nich wyodrębniać.

Jak to zrobić

Tablicę przechowującą zestaw elementów można zainicjalizować przypisując wiele wartości do zmiennej. Wystarczy rozdzielić każdą wartość za pomocą przecinka. Za pomocą poniższego polecenia można utworzyć tablicę zawierającą nazwy serwerów:

```
$servers = "EX1","EX2","EX3"
```

Aby utworzyć pustą tablicę skrótów, należy skorzystać z następującej składni:

```
$hashtable = @{}
```

Po utworzeniu pustej tablicy skrótów, można dodać pary klucz-wartość:

```
$hashtable["serwer1"] = 1
$hashtable["serwer2"] = 2
$hashtable["serwer3"] = 3
```