



WPROWADZENIE

PowerShell to zaawansowane narzędzie do automatyzacji i zarządzania konfiguracją, łączące moc skryptów z interaktywnym środowiskiem wiersza poleceń. Bazuje na platformie **.NET**, co umożliwia administratorom i programistom efektywne zarządzanie systemem oraz automatyzację zadań za pomocą cmdletów i skryptów. Dzięki obsłudze obiektów PowerShell pozwala na łatwe przetwarzanie danych oraz integrację z różnymi aplikacjami i usługami. Jest to wszechstronne narzędzie, odpowiednie zarówno dla początkujących, jak i zaawansowanych użytkowników.

HISTORIA

W 2002 roku Microsoft rozpoczął prace nad nową powłoką systemową, początkowo nazywaną **Monad**. Celem projektu było połączenie programowania, automatyzacji i zarządzania systemem, umożliwiając administratorom efektywne wykonywanie zadań nawet bez zaawansowanej znajomości programowania. Dziś PowerShell jest jednym z najważniejszych narzędzi w nowoczesnych środowiskach IT. Wyróżniamy dwie główne wersje:

- ♦ **Windows PowerShell** – dostępny natywnie w systemie Windows, oparty na .NET
- ♦ **PowerShell 7** – nowoczesna, wieloplatformowa wersja (Windows, Linux, macOS), oparta na .NET Core i rozwijana jako otwarte oprogramowanie

WINDOWS POWERSHELL

Wersja	Wraz z systemem	Data wydania
1.0	Windows XP / Windows Server 2008	01.11.2006
2.0	Windows 7 / Windows Server 2008 R2	01.11.2009
3.0	Windows 8 / Windows Server 2012	01.08.2012
4.0	Windows 8.1 / Windows Server 2012 R2	01.11.2013
5.0	Windows 10 / Windows Server 2016	16.12.2015
5.1	Windows 10 Anniversary / Windows Server 2016	27.01.2017

POWERSHELL CORE / POWERSHELL 7

Wersja	Wraz z systemem	Data wydania
6.0 +	Windows 7 SP1 / Windows Server 2008 R2 Ubuntu 14.04 / Debian 8.7 / Red Hat 7 / OpenSUSE 42.2 macOS 10.12	20.01.2018
7.0 +	Windows 7 SP1 / Windows Server 2008 R2 Ubuntu 16.04+ / Debian 9+ / Red Hat 7+ / OpenSUSE 15+ macOS 10.13+	04.03.2020

PODSTAWOWE DEFINICJE

cmdlet	podstawowa jednostka funkcjonalna w PowerShellu
funkcja	zestaw poleceń dla PowerShella
skrypt	plik tekstowy zawierający polecenia PS z rozszerzeniem .ps1
parametr	argument przekazywany do cmdletu/funkcji/skryptu
alias	skrót dla cmdletu/funkcji/skryptu
obiekt	instancja klasy, zawiera właściwości oraz metody
pipeline (potok)	mechanizm przekazywania obiektów między cmdletami

SKRÓTY Klawiszowe

CTRL+C	przerywa aktualnie wykonywane polecenie
lewo/prawo	nawigacja kursorem w linii poleceń
CTRL+lewo/prawo	przechodzi do poprzedniego/następnego słowa
HOME/END	przesuwa kursor na początek/koniec wiersza
góra/dół	poruszanie po historii poleceń
SHIFT + TAB/TAB	autouzupełnianie poleceń
ESC	czyści bieżącą linię poleceń
ALT + ENTER	przełącza tryb pełnoekranowy
CTRL + SHIFT + plus/minus	zwiększa / zmniejsza rozmiar czcionki

SPRAWDZANIE WERSJI

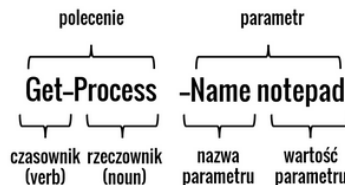
\$PSVersionTable	zmienna zawierająca dane dotyczące wersji PowerShella
rejestr	HKLM:\SOFTWARE\Microsoft\PowerShell\3\PowerShellEngine
rejestr (PS 7)	HKLM:\SOFTWARE\Microsoft\PowerShellCore\InstalledVersions
pwh -version (PS 7)	polecenie zwracające wersję PowerShella

CZYSZCZENIE OKNA KONSOLI

Clear-Host (cls/clear)	czyści ekran konsoli PowerShell
-------------------------------	---------------------------------

CMDLETY I PARAMETRY

Cmdlety (wymawiane 'command-lets') to specjalne polecenia w PowerShellu, które wykonują określone zadania. Są podstawowymi elementami składowymi skryptów i funkcji PowerShell. Zawsze składają się z **czasownika (verb)** i **rzeczownika (noun)**, oddzielonych łącznikiem, np. Get-Process, Set-Variable. Czasownik opisuje działanie, a rzeczownik określa obiekt, na którym to działanie jest wykonywane. Cmdlety mogą przyjmować **parametry**, które modyfikują ich zachowanie. Parametry poprzedzone są łącznikiem, który sygnalizuje konsoli, że słowo następujące po łączniku jest parametrem, a nie wartością przekazywaną do polecenia np. -Name, -Path.



Nie wszystkie parametry wymagają wartości i nie wszystkie nazwy parametrów muszą być określone. W niektórych przypadkach nazwa parametru jest domyślna i nie musi być jawnie podana. To znaczy, że niektóre polecenia będą działać doskonale bez podania parametru, inne tego parametru do działania będą wymagały, ale nie zawsze występuje konieczność podawania nazwy samego parametru.

POPULARNE CMDLETY

Get-Command	wyszukuje polecenia
Get-Help	wyświetla plik pomocy dla danego polecenia
Get-Service	wyświetla listę usług w systemie
Get-Process	wyświetla listę uruchomionych procesów
Get-Content	wyświetla zawartość pliku
Get-ChildItem	wyświetla zawartość katalogu
Get-Date	wyświetla aktualną datę
Set-Location	zmienia bieżącą lokalizację w drzewie katalogów
Show-Command	wyświetla graficzne okno do budowy poleceń

CZYTANIE SKŁADNI

Składnię dowolnego cmdletu można uzyskać, wykonując polecenie PowerShella: **Get-Command Get-EventLog -Syntax**. W wyniku otrzymasz:

```
Get-EventLog [-LogName <string>] [[-InstanceId <long>]] [-ComputerName <string>] [-Newest <int>] [-After <datetime>] [-Before <datetime>] [-UserName <string>] [-Index <int>] [-EntryType <string>] [-Source <string>] [-Message <string>] [-AsBaseObject] [<CommonParameters>]
```

```
Get-EventLog [-ComputerName <string>] [-List] [-AsString] [<CommonParameters>]
```

Przykład ilustruje dwa niezależne zestawy parametrów, z których można **korzystać wyłącznie rozdzielnie**, użycie elementów z obu grup jednocześnie spowoduje błąd składni. Część parametrów jest opcjonalna, co wskazują nawiasy kwadratowe []. Oznaczają one elementy, które można pominąć bez wpływu na poprawność wywołania. Nawiasy klamrowe {} określają z kolei zestaw wartości wzajemnie się wykluczających (należy wybrać dokładnie jedną z nich). Nawiasy trójkątne <> definiują typ oczekiwanej wartości, np. ścieżkę, liczbę czy obiekt. Warto również pamiętać, że nie wszystkie parametry wymagają podania wartości, wiele z nich to przełączniki, które uaktywniają konkretną funkcję jedynie poprzez samo wystąpienie w poleceniu, bez konieczności przypisywania im argumentów.

PARAMETRY WSPÓLNE

Parametry wspólne to specjalny zestaw parametrów, którego możesz używać dla dowolnego cmdletu. Są implementowane przez PowerShella i automatycznie dostępne dla każdego polecenia, choć istnieją pewne wyjątki. Przykładowo, jeżeli cmdlet nie generuje żadnych szczegółowych informacji, użycie wspólnego parametru **-Verbose** nie będzie miało efektu.

Debug (db)	informacje debugowania podczas wykonywania polecenia
ErrorAction (ea)	określa, jak PS reaguje na błędy niekrytyczne
ErrorVariable (ev)	zapisuje komunikaty o błędach do określonej zmiennej
InformationAction (infa)	określa, jak PS reaguje na strumień informacyjny z polecenia
InformationVariable (iv)	zapisuje komunikaty informacyjne do określonej zmiennej
OutVariable (ov)	zapisuje obiekt wyjściowy do określonej zmiennej
OutBuffer (ob)	liczba obiektów do buforowania przed wysłaniem do potoku
PipelineVariable (pv)	zapisuje wartość wyjściową polecenia do określonej zmiennej
Verbose (vb)	szczegółowe informacje o wykonywanych operacjach
WarningAction (wa)	określa, jak PS reaguje na ostrzeżenia
WarningVariable (wv)	zapisuje komunikaty ostrzeżeń do określonej zmiennej

WhatIf (wi)	przewiduje co się stanie po wykonaniu akcji
Confirm (cf)	wymaga dodatkowego potwierdzenia

WYSZUKIWANIE POLECEŃ

PowerShell umożliwia łatwe **wyszukiwanie poleceń**, co jest kluczowe dla efektywnej pracy. Podstawowym poleceniem do tego celu jest **Get-Command**, które pozwala przeszukiwać dostępne cmdlety, funkcje i aliasy. Wyniki można filtrować według różnych kryteriów, np. czasownika (verb) lub rzeczownika (noun) w nazwie polecenia, co ułatwia szybkie znalezienie odpowiednich komend. Parametry takie jak **-Name** (wyszukiwanie po nazwie) czy **-Module** (filtrowanie według modułu) zwiększają precyzję wyników. Aby znaleźć polecenia spoza lokalnego repozytorium, można użyć cmdletu **Find-Command**, który umożliwia przeszukiwanie zewnętrznych źródeł, takich jak przykładowo PowerShell Gallery.

Get-Command -CommandType	wyświetla wszystkie polecenia danego typu
Get-Command -Noun	wyszukuje polecenia zawierające dany rzeczownik
Get-Command -Verb	wyszukuje polecenia zawierające dany czasownik
Get-Command -Name	wyszukuje polecenia zawierające konkretną nazwę
Get-Command -Module	wyszukuje polecenia w danym module
Get-Command -ParameterName	wyszukuje polecenia z danym parametrem
Find-Command -Repository	wyszukuje polecenia w danym repozytorium

KORZYSTANIE Z POMOCY

PowerShell oferuje **wbudowany system pomocy**, który jest niezwykle przydatny dla użytkowników na każdym poziomie zaawansowania. Szczegółowe opisy poleceń i ich parametrów ułatwiają zrozumienie ich działania, a gotowe przykłady pomagają zastosować je w praktyce. Pomoc dostępna jest bezpośrednio w konsoli, co znacznie zwiększa komfort pracy. Podstawowym cmdletem używanym w tym celu jest **Get-Help**. Zastosowanie parametrów pozwala na wyświetlenie różnych poziomów szczegółowości pliku pomocy, a także na dostęp online. Możliwość aktualizacji umożliwia dostęp do najnowszych informacji.

Get-Help <polecenie>	wyświetla podstawowe informacje o poleceniu
<polecenie> -?	skrót do podstawowej pomocy dla polecenia
Get-Help <polecenie> -Full	pokazuje pełną dokumentację dla polecenia
Get-Help <polecenie> -Examples	wyświetla tylko przykłady użycia dla polecenia
Get-Help <polecenie> -Online	otwiera stronę Microsoft Docs z dokumentacją
Get-Help <polecenie> -ShowWindow	otwiera pomoc w osobnym oknie
Get-Help -Category HelpFile	wyświetla dostępne pliki pomocy dla PowerShella
Get-Help about_<temat>	pokazuje dokumentację dotyczącą danego tematu
Update-Help	aktualizuje pliki pomocy PowerShella
Save-Help	pobiera pliki pomocy i zapisuje je lokalnie

ALIASY

Aliasy to skrócone nazwy poleceń lub parametrów, które ułatwiają i przyspieszają pracę. Są one szczególnie przydatne podczas interaktywnej pracy w konsoli, gdzie szybkość wpisywania poleceń ma duże znaczenie. PowerShell oferuje wiele wbudowanych aliasów dla często używanych poleceń. Istnieje również możliwość tworzenia własnych. Trzeba jednak pamiętać, że bez dodania ich do profilu będą istniały **tylko w bieżącej sesji**. Warto mieć na uwadze, że choć skróty są wygodne w użyciu, to w skryptach zaleca się używanie pełnych nazw dla zwiększenia czytelności i niezawodności kodu.

Get-Alias	wyświetla wszystkie aliasy dostępne w bieżącej sesji
Get-Alias -Definition	wyświetla alias dla konkretnego cmdletu
Set-Alias	tworzy nowy alias lub zmienia istniejący
New-Alias	tworzy nowy alias, ale nie modyfikuje istniejącego aliasu
Remove-Alias	usuwa aliasy z bieżącej sesji
Export-Alias	eksportuje aliasy do pliku CSV lub pliku tekstowego
Import-Alias	importuje aliasy z pliku utworzonego przez Export-Alias

POPULARNE ALIASY

gcm	Get-Command	mi / move / mv	Move-Item
dir / gci / ls	Get-ChildItem	del / erase / rm	Remove-Item
cd / chdir / sl	Set-Location	ren / rni	Rename-Item
gl / pwd	Get-Location	% / foreach	ForEach-Object
gm	Get-Member	select	Select-Object
cat / gc / type	Get-Content	sort	Sort-Object
ghy / h / history	Get-History	group	Group-Object
copy / cp / cpi	Copy-Item	? / where	Where-Object

MODUŁY

Moduł to pakiet zawierający różne komponenty PowerShella, takie jak cmdlety, aliasy, funkcje, zmienne, pliki pomocy oraz skrypty, które stosowane są do rozszerzenia możliwości. Moduły są podzielone według tematyki lub zastosowania, co umożliwia łatwe zarządzanie. Koncepcja ta pojawiła się w PowerShellu w wersji 2.0 (wcześniej realizowane to było przez tzw. PSSnapins i wymagało niemałej wiedzy programistycznej). Obecnie moduły łaadowane są automatycznie przy wykonywaniu polecenia z danego modułu. W zależności od lokalizacji wyróżniamy trzy zakresy dostępności: **CurrentUser**, **AllUsers**, **System**.

Get-Module	wyświetla aktualnie zaimportowane moduły
Get-Module -ListAvailable	wyświetla listę wszystkich dostępnych modułów
Import-Module	importuje moduł do bieżącej sesji
Remove-Module	usuwa moduł z bieżącej sesji
Install-Module	instaluje moduł z zewnętrznego repozytorium
Get-InstalledModule	wyświetla zainstalowane moduły
\$PSModulePath	domyślna ścieżka dla modułów

PRZETWARZANIE POTOKOWE

Przetwarzanie potokowe (pipeline) to mechanizm umożliwiający łączenie wielu poleceń w jeden ciąg oraz przekazywanie danych pomiędzy nimi. Odbywa się to przy użyciu operatora potoku **| (pipeline operator)**, który pozwala przekazywać obiekty zwracane przez jeden cmdlet bezpośrednio do kolejnego. Dzięki temu możliwe jest budowanie bardziej złożonych operacji przy jednoczesnym zachowaniu czytelności i wysokiej wydajności przetwarzania danych. PowerShell automatycznie mapuje obiekty przesyłane potokiem do parametrów następnego polecenia, bazując na typach obiektów oraz nazwach ich właściwości. Przykład:

```
Get-Service | Where-Object {$_.Status -eq "Running"} | Select-Object Name
```

Dzięki przetwarzaniu potokowemu ze wszystkich serwisów wybieramy tylko te, które są uruchomione, dodatkowo ograniczamy się jedynie do wyświetlenia ich nazwy.

LOGIKA OBIEKTOWA

PowerShell opiera się na **logice obiektowej**, co oznacza, że wszystkie dane przetwarzane w tym środowisku mają postać **obiektów .NET**. Każdy taki obiekt posiada **właściwości (properties)**, które przechowują dane, oraz **metody (methods)**, czyli operacje, które można na nim wykonać. W przeciwieństwie do tradycyjnych interpreterów poleceń, które operują głównie na tekście, PowerShell umożliwia bezpośrednią pracę z obiektami. Dzięki temu jest bardziej elastyczny i wydajny w zakresie zarządzania systemem.

Get-Member	wyświetla właściwości i metody obiektów
\$data = Get-Date	tworzenie obiektu typu DateTime
\$data.Year	dostęp do właściwości
\$data.AddDays(1)	dostęp do metod
Measure-Object	oblicza sumę, średnią, min, max dla właściwości liczbowych
New-Object	tworzy nowy obiekt określonego typu

PROVIDERS

W przypadku PowerShella terminem **dostawcy (ang. Providers)** określa się specjalne moduły, które umożliwiają nawigację i zarządzanie różnymi typami danych w sposób podobny do systemu plików. Dzięki nim można używać standardowych poleceń do pracy z rejestrem, certyfikatami, zmiennymi środowiskowymi, aliasami i innymi zasobami systemu. Działają jak wirtualne dyski, pozwalając na przeglądanie i manipulowanie danymi w ich strukturach. Do wylistowania dostępnych dostawców użyj cmdletu: **Get-PSProvider**.

Provider	Opis	Windows PowerShell	PowerShell 7
Alias	aliasy	✓	✓
Environment	zmienne środowiskowe	✓	✓
FileSystem	pliki i katalogi	✓	✓
Function	funkcje zdefiniowane w sesji	✓	✓
Registry	rejestr Windows	✓	✗*
Variable	zmienne PowerShell	✓	✓
Certificate	magazynu certyfikatów	✓	✗**
WSMan	konfiguracja zdalna (WinRM)	✓	✗*

* Wspierane tylko na Windows
** Ograniczone wsparcie dla Linux/macOS, może wymagać dodatkowych modułów

SORTOWANIE

Sortowanie obiektów to proces porządkowania danych na podstawie określonych właściwości, co ułatwia ich analizę i prezentację. W PowerShellu najczęściej wykorzystuje się do tego celu cmdlet **Sort-Object**, który pozwala na sortowanie wyników w kolejności rosnącej (**Ascending**), jak i malejącej (**Descending**). Możliwe jest również sortowanie według wielu właściwości jednocześnie, co bywa przydatne przy pracy ze złożonymi strukturami danych. Dzięki obsłudze wyrażeń oraz bloków skryptowych Sort-Object pozwala na bardziej zaawansowane i elastyczne podejście do sortowania.

Przykład: Sortowanie według jednej wartości, malejąco

```
Get-ChildItem | Sort-Object -Property Length -Descending
```

Przykład: Sortowanie według kilku właściwości

```
Get-Service | Sort-Object -Property Status, DisplayName
```