Windows PowerShell | Cheat Sheet

Basiswissen für Administratoren und Systemmanager

Das Poster zum Download!

Autorenporträt

Markus Widl ist Autor mehrerer Fachbücher zu Microsoft 365, Copilot und Teams. In seinen Büchern, Vorträgen, Workshops und Briefings übersetzt er komplexe Technologien in praxisnahe Lösungen.

Sie erreichen Markus Widl unter:

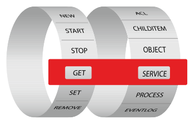
markus@widl.de

Cmdlets

Die wesentlichen Befehle der PowerShell werden Cmdlets (sprich »Commandlets«) genannt.
Welche Cmdlets verfügbar sind, liefert der Befehl
Get-Command -CommandType Cmdlet
Zu jedem Befehl können Sie einen Hilfetext mit dem Cmdlet Get-Help abrufen:

Get-Help CMDLET	(Befehlsübersicht)
Get-Help CMDLET -Examples	(Einsatzbeispiele)
Get-Help CMDLET -Full	(Ausführliche Hilfe)

Aufbau
Alle Cmdlets folgen denselben Namenskonventionen. Jeder Bezeichner beginnt mit einem Verb, dann folgt ein Minuszeichen und zuletzt ein Substantiv, mit dem die Art der zu verarbeitenden Daten bestimmt wird.
Die Groß-/Kleinschreibung ist dabei grundsätzlich egal.



Was tun? Womit?

Parameter & Argumente
Auch die Übergabe von Parametern und Attributen folgt bei jedem Cmdlet denselben Regeln.

Beispiel:

Get-EventLog -LogName System -EntryType Error, Warning

Parameter werden immer mit einem Minuszeichen eingeleitet. Argumente bestimmen den Wert des jeweiligen Parameters. Soll ein Parameter mit mehr als einem Argument belegt werden (im Beispiel -EntryType), werden diese mit einem Komma getrennt.

Aufrufvarianten
Der Cmdletaufruf ist sehr flexibel, was die Schreibweise der Parameter betrifft. Hier einige Varianten:

Get-Service -Name wuauserv -ComputerName London
Ausführliche Variante

Get-Service -N wuauserv -C London
Hier wurden die Parameternamen verkürzt geschrieben. Dies ist möglich, solange es keinen weiteren (optionalen) Parameter gibt, der mit N bzw. C beginnt.

Get-Service -C London -N wuauserv
Vertauschte Reihenfolge

Get-Service wuauserv -C London
Der Parameter -Name wurde weggelassen. Dies ist möglich, sofern die Reihenfolge, die in der Hilfe angegeben ist, eingehalten wird.

Die erste Variante eignet sich besonders in Scripten zur besseren Verständlichkeit des Befehls. Denken Sie an Ihre Kollegen, die Ihre Scripte vielleicht verstehen müssen.

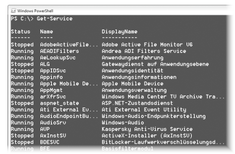
Aliasse
Aliasse gibt es als Cmdlet-Kurzform (z.B. copy für Copy-Item) und um den Umstieg aus anderen Shells zu erleichtern (z.B. dir für Get-ChildItem). Welche Aliasse es gibt, ermitteln Sie mit dem Cmdlet Get-Alias.
Eigene Aliasse legen Sie mit Set-Alias an.

Beispiel:
Set-Alias -Name gs -Value Get-Service

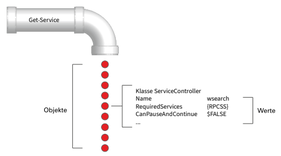
Klassen und Objekte

Ein ganz wesentliches Prinzip der PowerShell ist, dass die Cmdlets grundsätzlich keinen Text liefern, wie etwa bei Kommandozeilentools, die in der alten Eingabeaufforderung ausgeführt werden. Cmdlets liefern grundsätzlich Objekte. Diese Objekte haben einen ganz bestimmten Aufbau, der in der zugehörigen Klasse definiert ist. Die Klasse enthält keine Nutzdaten, etwa welche Benutzerkonten existieren, sondern nur den Aufbau von Benutzerkonten.

Geben Sie etwa den Befehl Get-Service, erhalten Sie folgende Ausgabe:



Get-Service liefert nicht den Text, sondern pro Dienst ein Objekt. Über die Objekte kann auf Eigenschaften (Properties), Methoden (Methods) und Ereignisse (Events) zugegriffen werden. Die ausgegebene Tabelle enthält drei Spalten, die für drei Eigenschaften stehen: Status, Name und Displayname. Im Regelfall enthalten die Objekte viele weitere Eigenschaften als nur die standardmäßig angezeigten. Damit stellen sich zwei Fragen: Wie sind die Objekte aufgebaut (d.h. wie sieht die Klasse aus)? Wie arbeite ich mit den nicht aufgeführten Bestandteilen?



Klassendefinition ermitteln
Den Objektaufbau ermitteln Sie über Get-Member.

Beispiel:
Get-Service | Get-Member
In der Ausgabe finden Sie den TypeName (Klassenname) und, je nach Klasse, die Eigenschaften (Properties, AliasProperties, ScriptProperties), Methoden und Ereignisse.

Objektbestandteile auswählen
Sind Sie an der Ausgabe anderer Eigenschaften als im Standardfall interessiert, hilft Select-Object (Alias select).
Durch Komma getrennt geben Sie die Bezeichner der gewünschten Eigenschaften an.
Geben Sie statt bestimmter Eigenschaften nur das Sternchen an, werden alle Eigenschaften samt ihrem Wert ausgegeben.

Get-Service | Get-Member

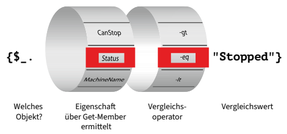
Get-Service | Select-Object Name, RequiredServices

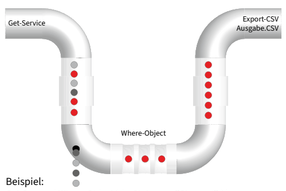
Pipeline

Mit der Pipeline verketten Sie Cmdlets untereinander. Die Ausgabe (Objekte) des einen Cmdlets wird an ein anderes Cmdlet weitergeleitet, das seine Funktion auf diese Objekte anwendet.

Get-Service liefert pro Dienst ein Objekt. Export-CSV verarbeitet ein Objekt nach dem anderen: Die Dienstinformationen werden nacheinander in der CSV-Datei abgelegt.

Objekte filtern
Oft werden Sie auf der Suche nach bestimmten Objekten sein, etwa beendete Dienste, Benutzerkonten mit abgelaufenem Kennwort, volle Mailboxen etc. Mit Where-Object (Aliasse ?, where) filtern Sie Objekte in der Pipeline.





Beispiel:
Get-Service | Where-Object { $_.Status -eq "Stopped" } | Export-CSV Dienste.csv

Seit PowerShell 3 können Sie auch verkürzt schreiben:
Get-Service | Where-Object Status -eq "Stopped" | Export-CSV Dienste.csv

Objekte verarbeiten
Wollen Sie ermittelte Objekte mit mehreren Befehlen verarbeiten oder gibt es kein Cmdlet für die gewünschte Aufgabe und Sie müssen auf eine Objektmethode zurückgreifen, hilft eine Schleife mit dem Cmdlet ForEach-Object (Aliasse %, foreach).

Beispiel:
Get-ChildItem | ForEach-Object { $_.Length / 1KB }

Die Kommandos in den geschweiften Klammern werden für alle Objekte von Get-ChildItem ausgeführt. Das Trennzeichen zwischen Befehlen ist das Semikolon (;) oder ein Zeilenwechsel. Wie bei Where-Object steht die Variable $_ für die jeweiligen Objekte aus der Pipeline.

Scripte, Funktionen und Filter

PowerShell-Scripte sind reine Textdateien mit der Endung .ps1 (auch bei PowerShell 2 und 3). Kommentare in Scripten werden mit # eingeleitet (denken Sie auch hier an Ihre Kollegen).

Ausführungsrichtlinie
Standardmäßig ist die Scriptausführung in der PowerShell deaktiviert (Sicherheit für Anwender). Dieses Verhalten regelt die Ausführungsrichtlinie ("Execution Policy"). Die aktuelle Einstellung wird mit Get-ExecutionPolicy abgefragt und mit Set-ExecutionPolicy gesetzt. Dabei sind u.a. folgende Einstellungen möglich:

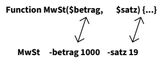
Restricted	(Scriptausführung deaktiviert)
Unrestricted	(Scriptausführung unbeschränkt möglich)
AllSigned	(Scripte müssen eine gültige Signatur tragen)
RemoteSigned	(Scripte aus nicht vertrauenswürdigen Quellen müssen eine gültige Signatur tragen)

Scripte ausführen
Standardmäßig wird eine Scriptdatei bei einem Doppelklick nicht ausgeführt (wie bei Batch- und VBS-Dateien), sondern sie wird in Notepad geöffnet (Sicherheit für Anwender). In der Konsole starten Sie ein Script über die Eingabe des Dateinamens und (ganz wichtig) des vorangestellten Pfads. Ob das .ps1 mit angegeben wird, ist Geschmackssache. Geben Sie also für die Scriptdatei **MeinScript.ps1** nicht **MeinScript** ein, sondern etwa
C:\MeinScript oder **.\MeinScript**, wenn die Datei im aktuellen Ordner liegt.

Übergabeparameter
Möchten Sie in Ihren Scripten Parameter entgegennehmen, die beim Aufruf mit angegeben werden, schreiben Sie als erste Codezeile im Script einen Param-Block mit Variablen.
Beispiel:
Param($file, $ou)
(Variablen werden in der PowerShell mit dem Dollar-zeichen kenntlich gemacht.)
Der Aufruf des Scripts erfolgt dann wie folgt:
C:\MeinScript -file test.txt -ou HR

Funktionen und Filter
Mit Funktionen und Filtern definieren Sie eigene Befehle. Der grundlegende Aufbau sieht so aus:
Function NAME(ÜBERGABEPARAMETER) { }
Filter NAME(ÜBERGABEPARAMETER) { }
Beispiel:
Function MwSt($betrag, $satz) { $betrag / 100 * $satz }
Rufen Sie die Funktion wie bei den Cmdlets auf:

MwSt -betrag 1000 -satz 19	oder
MwSt -b 1000 -s 19	oder
MwSt -s 19 -b 1000	oder
MwSt 1000 19	



Für Filter gilt dasselbe. Funktionen und Filter unterscheiden sich, wenn sie innerhalb der Pipeline eingesetzt werden.

Beispiel:
CMDLET | FUNKTION oder **CMDLET | FILTER**

Die Tabelle listet die wesentlichen Unterschiede auf:

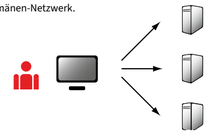
	Funktion	Filter
Erstellung mit	Function	Filter
Ausführen in der Pipeline	1x für komplet-ten Pipeline-Inhalt	1x für jedes Objekt in der Pipeline
Objektzugriff	Pipeline-Inhalt: $Input	Pipeline-Objekt: $_

Alle Objekte, die Sie innerhalb von Funktionen und Filtern ausgeben, werden in der Pipeline weitergeleitet.

Remoting

Mit der Remoting-Funktionalität können Sie auf anderen Computern im Netzwerk eine PowerShell-Sitzung starten und dort Kommandos ausführen. Die Ergebnisse werden dagegen auf der lokalen Maschine ausgegeben. Damit Sie Remoting nutzen können, müssen folgende Voraussetzungen erfüllt sein:

- Auf der Remote-Maschine wurde der WinRM-Dienst gestartet und konfiguriert sowie eine Firewallausnahme eingerichtet (am einfachsten über den einmaligen Aufruf von Enable-PSRemoting).
- Sie verfügen auf der Remote-Maschine über Administratorrechte.
- Die Remote-Maschine befindet sich im selben Domänen-Netzwerk.



Remoting mit PowerShell ISE
Am einfachsten ist die Remoting-Funktionalität über die PowerShell ISE ("Integrated Scripting Environment") zu nutzen. Ggf. müssen Sie diese auf den Server-Betriebssystemen als optionales Feature nachinstallieren. Es handelt sich dabei um eine (sehr) einfache grafische Oberfläche für die PowerShell-Anwendung und Script-Entwicklung.
Für das Remoting geben Sie den Menübefehl "File/New Remote-PowerShell-Tab" oder klicken auf das entsprechende Symbol.

Es erscheint ein Anmeldefenster. Geben Sie dort den Namen des Computers ein, zu dem Sie eine Verbindung aufbauen wollen. Der Benutzername samt Kennwort ist nur nötig, wenn Sie für die Anmeldung auf der Remote-Maschine ein anderes Benutzerkonto verwenden wollen.
Hat der Verbindungsaufbau geklappt, steht nun vor dem PowerShell-Prompt der Remote-Computername.
Alle Kommandos, die Sie nun in der ISE eingeben, werden auf der Remote-Maschine ausgeführt, die Ausgabe erfolgt jedoch lokal.

Remoting in Scripten
PowerShell-Remoting über die ISE ist für den Einsatz in Scripten kaum geeignet. Es gibt aber eine Reihe verschiedener Cmdlets, um das Remoting zu automatisieren. Wichtig ist dabei Invoke-Command. Hier geben Sie einen oder – durch Komma getrennt – mehrere Computernamen an, und einen Scriptblock, der auf der Remote-Maschine ausgeführt werden soll.

Beispiel:
Invoke-Command -Computername London -ScriptBlock { Get-EventLog -LogName System }

Mehrere voneinander unabhängige Befehle trennen Sie im ScriptBlock-Parameter mit einem Semikolon oder einem Zeilenwechsel.
Beim gezeigten Einsatz von Invoke-Command wird jeweils eine neue Remoting-Session aufgebaut und danach geschlossen (temporäre Session), was bei mehreren Invoke-Commands unnötig Zeit kostet. In diesem Fall wäre eine dauerhafte Session (persistente Session) von Vorteil.

Beispiel:
$s = New-PSSession -Computername London
Invoke-Command -Session $s -ScriptBlock { ... }
...
Invoke-Command -Session $s -ScriptBlock { ... }
Remove-PSSession -Session $s

Zu guter Letzt erhalten Sie eine Remoting-Session (interaktive Session) wie bei der ISE über das folgende Kommando:
Enter-PSSession -Computername London

Wichtige Cmdlets

Cmdlet	Funktion
Get-Help	Abfrage eines Hilfetextes für Cmdlets, -Examples für Beispiele, -Full für eine ausführliche Hilfe.
Get-Member	Ermitteln der Klasse(n) von Pipeline-Objekten.
Select-Object	Auswahl von Objektbestandteilen, Eigenschaften über -Property.
Get-WMIObject	Abfrage von WMI-Objekten über die Angabe einer WMI-Klasse unter -Namespace und -Class.
Measure-Object	Einfache Statistikfunktionen für Pipeline-Objekte für die unter -Property angegebenen Eigenschaften mit Anzahl, Summe (-Sum), Durchschnitt (-Average), kleinster Wert (-Minimum), größter Wert (-Maximum).
New-Object	Erzeugen eines .NET Framework-Objekts über die Angabe einer .NET-Klasse unter -TypeName oder eines COM-Objekts unter -COM-Object.
Sort-Object	Sortiert Objekte aus der Pipeline nach der unter -Property angegebenen Eigenschaft aufsteigend, mit -Descending absteigend.
Where-Object	Filtern von Pipeline-Objekten.
Export-CliXML	Exportieren von Pipeline-Objekten in die unter -Path angegebene Datei im XML-Format.
Import-CliXML	Importieren von Daten aus der unter -Path angegebenen XML-Datei.
Export-CSV	Exportieren von Pipeline-Objekten in die unter -Path angegebene Datei im CSV-Format.
Import-CSV	Importieren von Daten aus der unter -Path angegebenen CSV-Datei.
Read-Host	Eingabe einer Zeichenfolge durch den Benutzer in der Kommandozeile.
Out-GridView	Grafische Ausgabe der Pipeline-Objekte in einem Windows-Fenster. Der Anwender kann dort Filtern und Sortieren.
Send-MailMessage	Versand einer E-Mail.
Format-Table -Autosize	Die Ausgabe der Eigenschaften der Pipeline-Objekte erfolgt in einer Tabelle, bei der zwischen den Spalten möglichst wenig Platz gelassen wird.
Show-Command	Grafische Anzeige und Angabe von Parametern und Argumenten.

Wichtige Operatoren

Zuweisungsoperatoren	Funktion	Beispiel
=	Zuweisung	**$s = "London"**
+=	Addition	**$a = 1; $a += 1** liefert **$a = 2**
-=	Differenz	**$a = 2; $a -= 1** liefert **$a = 1**
***=**	Multiplikation	**$a = 3; $a *= 4** liefert **$a = 12**
/=	Division	**$a = 6; $a /= 3** liefert **$a = 2**
++	Addition +1	**$a = 1; $a++** liefert **$a = 2**
--	Differenz -1	**$a = 2; $a--** liefert **$a = 1**

Vergleichsoperatoren

-replace	Suchen/Ersetzen	**"Das ist ein Test" -replace "ein", "kein"** liefert **"Das ist kein Test"**
-like	Vergleich mit Wildcards	**"C:\Windows" -like "*\"** liefert **$false**
-match	Vergleich mit regulären Ausdrücken	**"user@host.com" -match "^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$"** liefert **$true**
-eq	Vergleich	**5 -eq 6** liefert **$false**
-gt	größer als	**5 -gt 6** liefert **$false**
-ge	größer oder gleich	**5 -gt 6** liefert **$false**
-lt	kleiner als	**5 -lt 6** liefert **$true**
-le	kleiner oder gleich	**5 -le 6** liefert **$true**

Formatierungsoperator -f

Operator	Funktion	Beispiel	Ergebnis
{0}	Anzeige eines bestimmten Elements	**"{0} {1}" -f "Hans", "Muster"**	**Hans Muster**
{0:p}	Anzeige einer Prozentzahl	**"{0:p}" -f .456**	**45,60 %**
{0:F2}	Feste Anzahl Nachkommastellen	**"{0:F2}" -f (1000/3)**	**333,33**