## Inhalt



# Vorwort

Dieses Dokument soll als Einführung zum Thema Windows Power Shell verstanden werden. Ich gehe hierbei auf die Benutzung gängiger Cmdlets ein und verdeutliche dies mit einigen Beispielen. Da es sich nur um eine Einführung handelt, gehe ich hierbei kaum auf das Scripting bei der PowerShell ein.

# Einführung

Die Power Shell basiert auf dem .NET-Framework 2.0, arbeitet dadurch komplett objektorientiert und verwendet, wie aus UNIX-Shells bekannt, Pipelines und Filter.

Die so genannten Cmdlets bilden den Kern der Windows Power Shell und folgen der Benennungskonvention „Verb-Substantiv“.

Beispiele:
```
Get-Process
Get-Service
Set-Service
```

Dadurch kann sich der Administrator die Kommandos besser merken; wer oft mit der Power Shell zu tun hat, kann die vordefinierten Aliase verwenden oder eigene Aliase definieren, was die Benutzung der Shell durch die schnellere Eingabe beschleunigt.

Beispiel: `Get-ChildItem`
Alias: `gci`

Das Kommando „cd“ ist in der Power Shell ein Alias für das Cmdlet „Set-Location“. Dieses Kommando ist mächtiger als der cd-Befehl der cmd.exe.

Die Windows Power Shell stellt ein Werkzeug dar, mit dem nicht nur der Exchange 2007 Server, sondern auch alle zukünftigen Windows Server-Produkte komplett ohne grafische Oberfläche administriert werden können. Zukünftige MMC-SnapIns werden auf der Power Shell aufbauen, bzw. deren Funktionalitäten nutzen.

Vor der Installation der Windows Power Shell muss zunächst das .NET-Framework 2.0 installiert werden.

# Erste Schritte in Windows Power Shell

## *Get-Command und Get-Member*

Das Cmdlet „Get-Command“ (die Power Shell beachtet bei den Cmdlets, Funktions- oder Skriptnamen keine Groß- und Kleinschreibung) listet alle verfügbaren Cmdlets auf.

Wie schon erwähnt, arbeitet die Power Shell objektorientiert. Auch das Cmdlet „Get-Command“ gibt demnach Objekte zurück und keinen Text, wie es bei herkömmlichen Shells üblich ist. Für eine genauere Analyse wäre es interessant, sich die Klassenstruktur anschauen zu können. Dies erledigt das Cmdlet „Get-Member“. Zum besseren Verständnis wird zunächst ein String angelegt (Klasse System.String), um die Klassenstruktur zu analysieren.

```
[System.String]$a = “Hallo Welt“
```

Der String wird jetzt über eine Pipeline an das Cmdlet “Get-Member” übergeben.

```
$a | Get-Member
```

Die Ausgabe sollte unter anderem die Methoden und Eigenschaften des Objektes $a anzeigen.

Soll die Länge des Strings ermittelt werden, kann die Eigenschaft „Length“ verwendet werden.

```
$a.Length
```

Der Parameter „MemberType“ ermöglicht eine Einschränkung der Ausgabe.

```
$a | Get-Member -MemberType Property
```

Dieses Kommando zeigt nur die Eigenschaften an; der Wert „Method“ für den Parameter „MemberType“ würde stattdessen die Methoden anzeigen.

Weitere Erläuterungen zu Stringfunktionen würden den Rahmen einer Einführung sprengen; primär geht es in diesem Dokument darum, Power Shell Cmdlets zu verwenden und nicht um das Scripting in der Power Shell.

Eine Analyse der Klassenstruktur für „Get-Command“-Objekte (Get-Command | Get-Member) zeigt unter anderem zwei interessante Eigenschaften: Noun und Verb. Folgendes Kommando zeigt beispielsweise alle Cmdlets, die das Substantiv „Process“ beinhalten:

```
Get-Command -Noun Process
```

Über die Eigenschaften „Noun“ und „Verb“ lassen sich relativ schnell gesuchte Kommandos finden.

## *Hilfe und Syntax von Cmdlets*

Mit dem Cmdlet „Get-Help“ kann die Hilfe zu einem Power Shell Cmdlet angezeigt werden:

```
Get-Help Get-Process
```

Folgender Befehl zeigt die Syntax eines Cmdlets an:

```
Get-Command Get-Process -syntax
```

## *Arbeiten mit Dateien und Verzeichnissen*

Das Cmdlet „Get-Location“ ist vergleichbar mit dem UNIX-Kommando „pwd“ und zeigt den aktuellen Pfad an. Dieser kann über „Set-Location“ verändert werden; ein Alias, der dafür in der Standardeinstellung existiert, ist „cd“.

Wichtig:
Das Kommando „cd“ hat nichts mit dem cd-Befehl der „cmd.exe“ zu tun. Es ist, wie bereits erwähnt, ein Alias für das Power Shell Cmdlet „Set-Location“.

Mit dem cd-Befehl der „cmd.exe“ Befehlszeile war es nicht möglich, durch einen Schritt in ein Verzeichnis einer anderen Partition zu wechseln (z.B. vom Benutzerverzeichnis unter C: in D:\test). Das Cmdlet „Set-Location“ hebt diese Einschränkung auf:

```
Set-Location -Path D:\test
```

Der Path-Parameter wird vom Cmdlet „Set-Location“ standardmäßig als erster erwartet, sofern nur ein Parameter angegeben wird. Zudem ist es möglich den Alias „cd“ oder „sl“ zu verwenden, um Schreibarbeit zu sparen:

```
cd D:\test
```

Um sich alle Elemente in einem Verzeichnis anzeigen zu lassen, kann das Cmdlet „Get-ChildItem“ verwendet werden. „gci“ oder „dir“ sind Aliase für dieses Kommando.

# Anpassen, formatieren, filtern und sortieren der Ausgabe

## *Ausgabe anpassen (Select-Object)*

Die Ausgabe kann mit verschiedenen Kommandos angepasst werden. Zunächst soll das Cmdlet „Select-Object“ vorgestellt werden, mit dem, wie der Name schon vermuten lässt, bestimmte Eigenschaften eines Objektes selektiert werden können.

Da sich kein Administrator alle Eigenschaften eines Objektes merken kann, wird zunächst die Klassenstruktur mit „Get-Member“ analysiert:

```
Get-ChildItem | Get-Member
```

Das folgende Beispiel zeigt neben dem Dateinamen die Dateigröße sowie drei interessante Zeitstempel an.

```
Get-ChildItem | Select-Object
Name,Length,CreationTime,LastAccessTime,LastWriteTime
```

Auch diese Zeile lässt sich durch Aliase wesentlich kürzer gestalten:

```
gci | Select
Name,Length,CreationTime,LastAccessTime,LastWriteTime
```

## *Ausgabe formatieren (Format-List, Format-Table)*

Das Cmdlet „Format-List“ reserviert für jede Objekteigenschaft eine separate Zeile; der letzte Befehl mit dem Select-Object-Cmdlet hat auch eine solche Ausgabe gebracht, da dieses Cmdlet automatisch dafür sorgt, wenn die Zeilenbreite nicht ausreicht.

Die Ausgabe von „Get-Command“ zeigt allerdings, dass Text, der über das Zeilenende hinaus ragt, abgeschnitten wird.
Durch das Pipen von „Get-Command“ an das Cmdlet „Format-List“ wird jede Eigenschaft in einer Zeile angezeigt:

```
Get-Command | Format-List
```

Für „Format-List“ kann der Alias „fl“ verwendet werden. Die Ausgabe mit „Format-List“ zeigt häufig weitere Eigenschaften an, die vorher aufgrund der Zeilenbreite nicht sichtbar waren. Dies sind jedoch nicht alle Eigenschaften. Um sich alle Eigenschaften eines Objektes anzeigen zu lassen, kann das * als Platzhalter für das Select-Object verwendet werden:

```
Get-Command | Select-Object *
```

Das Cmdlet „Format-Table“ ermöglicht eine tabellarische Anordnung der Eigenschaften; mit dem Parameter „AutoSize“ wird die Ausgabe automatisch platzsparend angepasst, wodurch es in der Regel möglich sein sollte, mehrere Eigenschaften in einer Zeile unterzubringen:

```
gci | Select Name,LastAccessTime,LastWriteTime,CreationTime | ft
–AutoSize
```

“ft” ist ein Alias für das Cmdlet “Format-Table”.

Sofern eine Eigenschaft nicht komplett angezeigt werden kann (z.B. bei Get-Command), schafft der Parameter „Wrap“ vom Cmdlet „Format-Table“ Abhilfe.

```
Get-Command | Format-Table –AutoSize –Wrap
```

Dieser Parameter sorgt für einen Zeilenumbruch.

## *Ausgabe filtern (Where-Object)*

Durch die Objektorientierung lassen sich Filterungen in der Ausgabe mit der Power Shell eleganter lösen, als Administratoren dies beispielsweise von UNIX-Shells mit dem Kommando „grep“ gewohnt sind.

Das Cmdlet „Where-Object“ (Alias: „Where“) schränkt die Ausgabe ein und ist durch Operatoren in der Lage, Objekteigenschaften als Filterkriterien zu verwenden.

Folgende Operatoren sind in diesem Zusammenhang interessant:

| **Operator** | **Bedeutung** |
|---|---|
| -eq | Ist gleich (equal) |
| -ne | Ist ungleich (not equal) |
| -lt | Ist kleiner als (lower than) |
| -gt | Größer als (greater than) |
| -le | Kleiner oder gleich (lower or equal) |
| -ge | Größer oder gleich (greater or equal) |
| -like | Entspricht (Vergleich mit Platzhaltern) |
| -notlike | Entspricht nicht (Vergleich mit Platzhaltern) |

Das folgende Beispiel zeigt alle laufenden Dienste auf dem Computer an und verwendet das Where-Object-Cmdlet, um die Ausgabe einzuschränken:

```
Get-Service | Where { $_.Status –eq „Running“ }
```

$_ bezieht sich immer auf das aktuelle Pipelineobjekt, welches das Where-Object-Cmdlet durchläuft. Der Punkt trennt das Objekt von der Eigenschaft, in diesem Fall „Status“.

### *Ausgabe sortieren (Sort-Object)*

Um bei dem Beispiel der Liste aller Dienste zu bleiben: Interessant wäre eine Ausgabe der Dienste sortiert nach ihrem Status. Das Cmdlet „Sort-Object“ sortiert Objekte und erwartet als Parameter Objekteigenschaften.

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

Durch die Angabe der Eigenschaft “DisplayName” wird unterhalb vom Dienststatus auch nach dem Anzeigenamen des Dienstes (alphabetisch von a bis z) sortiert. Um die Sortierung umzukehren (beispielsweise um die Dienste von z bis a zu sortieren), kann der Parameter „descending“ verwendet werden.

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

Hierbei wird absteigend nach den Attributen Status und DisplayName sortiert.

## Umleiten der Ausgabe

Die Power-Shell kann die Ausgabe, wie andere Shells auch, umleiten. Entsprechende Kommandos beginnen mit „Out“. Demnach kann eine Liste dieser Cmdlets mit folgendem Kommando angezeigt werden:

```
Get-Command -verb out
```

### *Out-File*

Das Cmdlet „Out-File“ leitet die Ausgabe, wie der Name schon vermuten lässt, in eine Datei um. Dies entspricht dem Umleiten von Daten durch „>“ in einer UNIX-Shell.

```
Get-Process | Out-File C:\processlist.txt
```

Die Datei wird überschrieben, falls Sie bereits mit Inhalt gefüllt ist.

Um die Ausgabe an die Datei anzuhängen, was dem „>>“ unter UNIX-Shells entspricht, verwendet man beim Cmdlet „Out-File“ den Parameter „append“.

```
Get-Process | Out-File C:\processlist.txt -append
```

### *Out-Host*

Out-Host sendet die Ausgabe, was auch standardmäßig der Fall ist, an die Konsole. Interessant ist hierbei der Parameter „paging“ der auf einen Tastendruck wartet und die Ausgabe seitenweise darstellt. Dies entspricht dem „more“-Kommando unter UNIX-Shells; auch unter der Power Shell gibt es einen Alias „more“, der das Out-Host-Cmdlet mit dem Parameter „paging“ aufruft.

```
Get-Service | Out-Host -paging
```

oder:

```
Get-Service | more
```

## *Out-Null*

Das Cmdlet “Out-Null” verwirft die Ausgabe; dies entspricht unter UNIX der Weiterleitung einer Ausgabe an das “/dev/null”-Device.

## *Out-Printer*

Wird das Cmdlet Out-Printer ohne einen Parameter verwendet, nutzt Windows den eingerichteten Standarddrucker. Alternativ kann über den Parameter „Name“ der Druckername angegeben werden. Sind beispielsweise die Microsoft Office Document Imaging-Tools installiert, können somit auch Daten direkt in einer Bilddatei gespeichert werden.

## *Konvertieren der Ausgabe*

### ConvertTo-Html

Das Cmdlet ConvertTo-Html konvertiert die Daten in HTML-Code. Durch Stringfunktionen lässt sich dieser HTML-Code dann natürlich auch bearbeiten.

Beispiel:

```
$htmlcode = Get-Service | Sort-Object Status,DisplayName | ConvertTo-Html
```

Eine Analyse des Objekts $htmlcode mit “Get-Member” zeigt, dass es sich um ein Objekt vom Typ String handelt.

Bei einem solchen String kann man jede Zeile durch die Verwendung des Arrays ansprechen, d.h. $htmlcode[0] steht für die erste Zeile. Um Stringfunktionen verwenden zu können, muss der Index des Arrays, d.h. die Zeilennummer, mit angegeben werden. Dadurch könnte in einer Schleife das Arrays (alle Zeilen) durchlaufen und HTML-Code editiert werden.
Dieses Vorhaben fällt jedoch eindeutig unter den Bereich Scripting und wird daher hier nicht weiter erläutert.

### Export-Csv

Das Cmdlet „Export-Csv“ ermöglicht es, CSV-Dateien zu erstellen.

Beispiel:

```
Get-Service | Select DisplayName,Status | Export-Csv C:\services.csv
```

# Dateisystemaufgaben

## *Power Shell Laufwerke*

Die Windows Power Shell verwaltet eigene Laufwerke, die mit dem Cmdlet „Get-PSDrive“ aufgelistet werden können. Unter anderem sind dort auch die Dateisystemlaufwerke (z.B. C) zu finden.

Interessant sind die vordefinierten Laufwerke „HKCU“ (HKEY_CURRENT_USER) und „HKLM“ (HKEY_LOCAL_MACHINE, mit denen direkt auf die Registry zugegriffen werden kann.

```
Set-Location HKLM:
```

Neue Schlüssel könnten beispielsweise mit dem Cmdlet „New-Item“ erstellt werden.

## *Neue Verzeichnisse und Dateien anlegen*

Mit dem Cmdlet „Get-ChildItem“, was anfangs schon kurz erwähnt wurde, können Elemente in Power Shell Laufwerken angezeigt werden. Der Parameter „Force“ zeigt hierbei auch versteckte Dateien an; der Parameter „Recurse“ führt eine rekursive Auflistung von untergeordneten Dateien und Verzeichnissen durch.

Neue Elemente werden mit dem Cmdlet „New-Item“ angelegt. Mit diesem Cmdlet können Registrierungsschlüssel, Dateien oder Verzeichnisse angelegt werden. Der „ItemType“-Parameter bestimmt, um was es sich dabei handelt.

Folgendes Kommando erstellt ein neues Verzeichnis:

```
New-Item C:\test –ItemType Directory
```

Um neue Dateien anzulegen, verwendet der Parameter “ItemType” den Wert “file”.

```
New-Item C:\test.txt –ItemType file
```

## *Kopieren von Elementen (Copy-Item)*

Das Power Shell Cmdlet “Copy-Item” ist in der Lage, Elemente, z.B. Dateien oder Verzeichnisse, zu kopieren.

Mit dem Parameter “Path” wird die Quelle angegeben; das Ziel wird mit dem Parameter “Destination” angegeben. Folgendes Kommando kopiert die Datei “test.txt” auf die Partition D:

```
Copy-Item –Path C:\test.txt –Destination D:\
```

Das Cmdlet “Copy-Item” erwartet standardmäßig als ersten Parameter die Quelle und als zweiten Parameter das Ziel. Dadurch ist es in diesem Fall möglich, die Parameternamen wegzulassen. Zudem existiert für das Cmdlet “Copy-Item” der Alias “cp”, wodurch man das

Cmdlet für diese einfache Aufgabe genauso verwenden kann, wie man es von der cmd.exe gewohnt ist.

```
cp C:\test.txt D:\
```

Sofern die Zieldatei bereits existiert, kann sie mit dem Parameter “Force” überschrieben werden. Der Parameter “Recurse” kopiert Verzeichnisse rekursiv.

### *Verschieben von Elementen (Move-Item)*

Das „Move-Item“ Cmdlet akzeptiert, genauso wie das Cmdlet „Copy-Item“, die Parameter Quelle und Ziel in der genannten Reihenfolge. Möchte man also die Datei C:\test.txt auf die Partition D: verschieben, funktioniert dies analog zum „Copy-Item“ Cmdlet:

```
Move-Item C:\test.txt D:\
```

### *Umbenennen vorhandener Elemente (Rename-Item)*

Das Cmdlet „Rename-Item“, welches Elemente umbenennt, funktioniert beim Umbenennen von Elementen analog zum „Copy-Item“ und „Move-Item“ Cmdlet.

```
Rename-Item D:\test.txt D:\testdatei.txt
```

### *Löschen von Elementen (Remove-Item)*

Eine Datei kann mit dem Cmdlet „Remove-Item“ gelöscht werden.

```
Remove-Item D:\testdatei.txt
```

Ein leeres Verzeichnis kann auf die gleiche Weise gelöscht werden; möchte man auch alle im Verzeichnis enthaltenen Dateien löschen, muss der Parameter „Recurse“ angegeben werden.

```
Remove-Item C:\testordner\ -Recurse
```

## Systemverwaltungsaufgaben

### *WMI*

Über WMI (Windows Management Instrumentation), die Kerntechnologie der Windows-Systemverwaltung, lassen sich Informationen einheitlich abrufen.

Alle WMI-Klassen werden mit dem Parameter „List“ vom Cmdlet „Get-WmiObject“ angezeigt:

```
Get-WmiObject -List
```

Das Cmdlet „Get-WmiObject“ akzeptiert auch den Parameter „ComputerName“, durch den ein Remote-Computer angegeben werden kann. Für solche WMI-Abfragen sind Administratorrechte auf dem Remotecomputer notwendig.

## Verwalten lokaler Prozesse

Zum Verwalten lokaler Prozesse stellt die Power Shell zwei Cmdlets zur Verfügung: „Get-Process“ und „Stop-Process“.

Das Cmdlet „Get-Process“ zeigt ohne Parameter alle Prozesse auf dem lokalen Computer an. Die Prozessnamen weisen im Gegensatz zum „tasklist“-Befehl nicht die Endung „.exe“ auf. Dies liegt an der Konvention der System.Diagnostics.Process-Klasse, welche Windows Power Shell Prozesse verwenden.

Der Get-Process Befehl akzeptiert den Parameter „Name“; um beispielsweise alle laufenden Firefox-Prozesse anzuzeigen, hilft folgende Kommandozeile weiter:

```
Get-Process -Name firefox
```

Diese Zeile kann durch eine Pipeline an das Cmdlet „Stop-Process“ übergeben werden, um den Prozess zu beenden.

## Verwalten lokaler Dienste

Microsoft Windows Power Shell stellt zum Verwalten von Diensten verschiedene Cmdlets bereit. Welche das sind, kann durch Get-Command herausgefunden werden:

```
Get-Command -Noun Service
```

Das Cmdlet „Get-Service“ zeigt ohne Parameter eine Liste aller Dienste an. Beim Betrachten der Klassenstruktur fallen einem interessante Eigenschaften auf, z.B. die Eigenschaft „Status“, die mit dem Wert „Running“ einen laufenden Dienst angibt.

Das Cmdlet „Stop-Service“ stoppt, wie der Name schon sagt, einen Dienst. Dieses Cmdlet lässt sich auch über eine Pipeline an ein Get-Service Cmdlet anhängen; dies gilt auch für das „Restart-Service“ Cmdlet zum Neustarten von Diensten.

Mit dem Cmdlet „Set-Service“ kann auch der Starttyp des Dienstes geändert werden. Dazu muss der Parameter „StartupType“ einen der folgenden Werte annehmen:

- Automatic
- Manual
- Disabled

## Verwalten des Active-Directory

Mit dem Snap-In „ActiveRoles“ von Quest, lässt sich das Active Directory komfortabel verwalten und Abläufe können so mit Hilfe von Skripts automatisiert werden.

Quest bietet dieses Snap-In kostenlos zum Download an:

http://www.quest.com/activeroles-server/arms.aspx

Der folgende Abschnitt soll eine kleine Einführung für nützliche Cmdlets dieser Erweiterung geben.

### Get-QADUser

Dieses Cmdlet listet alle Benutzer im AD auf. Durch die Eigenschaft „SamAccountName“ lassen sich folgendermaßen verschiedene Eigenschaften eines Benutzers anzeigen:

```
Get-QADUser -SamAccountName testuser | Select *
```

Folgendes Beispiel listet alle Benutzer mit einem dreistelligen Benutzernamen auf:

```
Get-QADUser | where { $_.SamAccountName.Length -eq 3 } | Select
SamAccountName, DisplayName
```

Durch die Eigenschaft „Company“ können Benutzer einer bestimmten Niederlassung/Firma ausgegeben werden; in diesem Beispiel wird zudem ein dreistelliger Benutzername erwartet:

```
Get-QADUser | where { ($_.Company -eq „testfirma“)  -and
($_.SamAccountName.Length -eq 3) } | Select
SamAccountName,DisplayName
```

### Set-QADUser

Mit diesem Cmdlet können AD-Benutzer verändert werden. Um beispielsweise die Firma für den Benutzer „seb“ zu ändern, ist folgende Kommandozeile zulässig:

```
Get-QADUser -SamAccountName tes | Set-QADUser -Company
"testfirma"
```