

## 4 Die Windows PowerShell

Die Windows PowerShell versucht, das "Beste aus allen verfügbaren Shells" aus verschiedenen Betriebssystemwelten herauszuholen.

Windows PowerShell ist in jeder Windows-Version ab Windows Vista/Windows Server 2008 verfügbar. Die Verwaltung jeder Microsoft-Anwendung (vor allem Server-Applikationen), die noch längerfristig weiterentwickelt wird, soll mit Hilfe von PowerShell erfolgen.

Eine zentrale Rolle spielt PowerShell bei der Verwaltung von Exchange Server 2007 und 2010; seit SQL Server 2008 R2 gibt es die Möglichkeit, die wichtigsten Tasks nicht nur mit TSQL, sondern auch mit PowerShell auszuführen.

Im Gegensatz zu anderen Shells arbeitet die Windows PowerShell mit **Objekten**. Folgende Objekttypen werden unterstützt:

- .NET-Objekte
- COM-Objekte
- WMI-Objekte

Die PowerShell-Kommandos werden als **Cmdlets** bezeichnet.

- Scripting-Dateien: \*.ps1

Ähnlich wie die MMC kennt die PowerShell das Snap-In-Konzept. So wird die PowerShell etwa für die Verwaltung von Exchange Server 2007 durch ein Snap-In erweitert.

Starten von Windows PowerShell:

Windows PowerShell wird mit zwei Oberflächen ausgeliefert:

Die **PowerShell-Konsole** (PowerShell.exe) kann über das Startmenü oder (mit Parametern) über das Menü Ausführen gestartet werden.



```

Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. Alle Rechte vorbehalten.

PS Z:\> get-help
THEMA
    Get-Help

KURZBESCHREIBUNG
    Zeigt Hilfe zu Windows PowerShell-Cmdlets und -Konzepten an.

DETAILBESCHREIBUNG

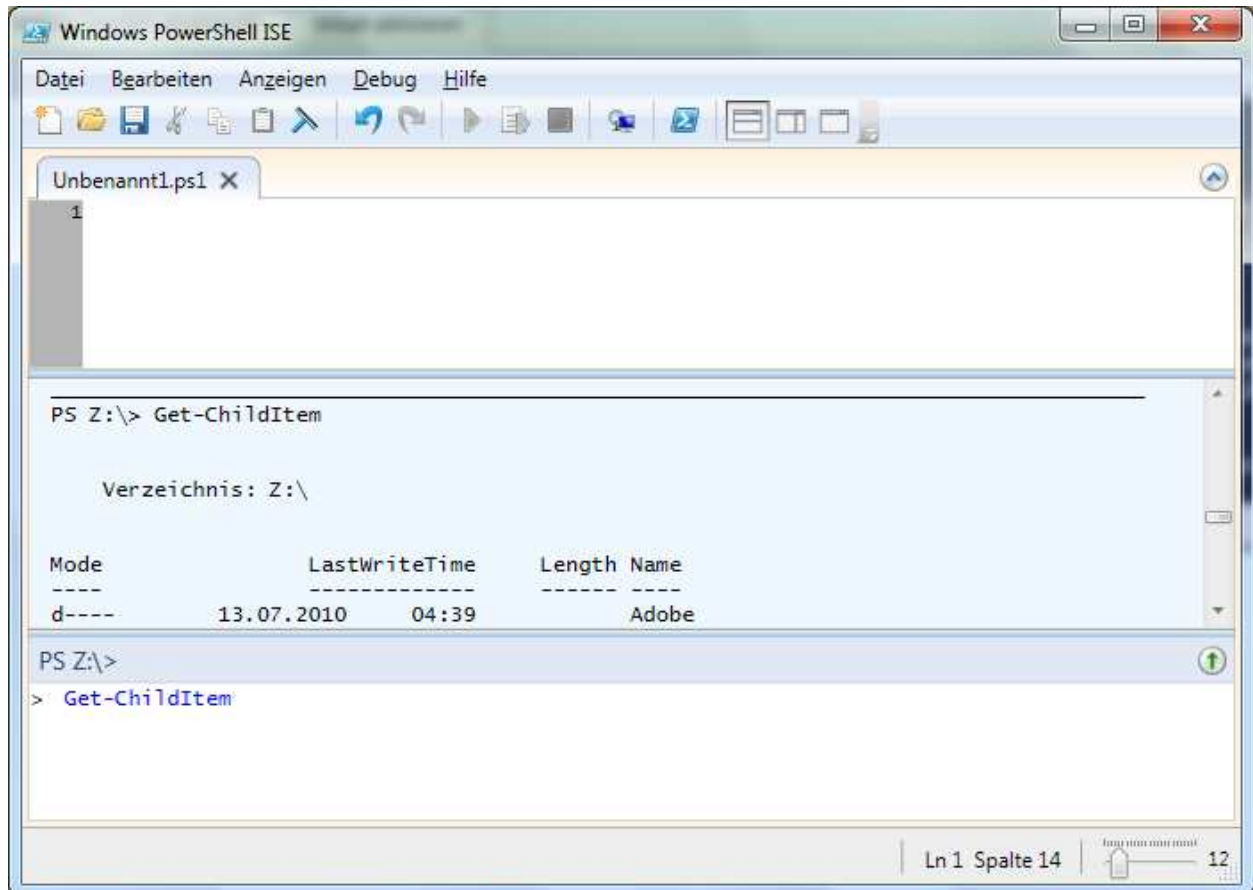
SYNTAX
    get-help <<Cmdlet-Name> ! <Thema>>
    help <<Cmdlet-Name> ! <Thema>>
    <Cmdlet-Name> -?

Mit "Get-help" und "-?" wird die Hilfe auf einer Seite angezeigt.
Mit "Help" wird die Hilfe auf mehreren Seiten angezeigt.

Beispiele:
    get-help get-process      : Zeigt Hilfe zum Cmdlet "Get-Process" an.
    get-help about_signing    : Zeigt Hilfe zum Signieren von Skripts an.
    help where-object         : Zeigt Hilfe zum Cmdlet "Where-Object" an.
    help about_foreach        : Zeigt die Hilfe zu foreach-Schleifen in
                                PowerShell an.
    set-service -?           : Zeigt Hilfe zum Cmdlet "Set-Service" an.

In den help-Befehlen können Sie Platzhalterzeichen verwenden
  
```

Das **PowerShell ISE (Integrated Scripting Environment)** bietet eine farblich gekennzeichnete Syntaxunterstützung und die Möglichkeit, PowerShell Scripts zu erstellen und zu testen.



## 4.1 CmdLets

PowerShell-Anweisungen werden als **Cmdlets** bezeichnet.

Jedes cmdlet hat ein Standardformat: **verb-noun -param**

Dabei gibt **verb** (deutsch: Zeitwort) gibt dabei eine Aktion an, die durchgeführt werden soll (get, set, list, new, start, stop etc.), **noun** (deutsch: Hauptwort) an, worauf sich die Aktion bezieht. Einzelne Parameter werden mit einem Bindestrich (Minus-Zeichen) eingeleitet.

Beispiel: Der „normalen“ Shell-Anweisung `dir` entspricht das Cmdlet

```
get-childitem
```

Um alle verfügbaren Cmdlets aufzulisten, steht das Cmdlet

```
get-command
```

zur Verfügung.

`get-help` (substituiert mit Seitenumbruch durch `help` oder `man`)

Für Kommando und Parameter: Tabulator-Completion

Viele Kommandos können mit Alias-Namen versehen werden; so gibt es für **get-childitem** bereits vordefiniert die Alias-Namen **ls** und **dir**.

Achtung auf Syntax: das Leerzeichen von den Parametern darf nicht weggelassen werden, also

FALSCH: `cd \`

RICHTIG: `cd \`

```
gps = get process
```

Die letzten (standardmäßig 64) verwendeten Eingabezeilen werden gespeichert („History“) und können mit dem Cmdlet **Get-History** oder der Funktionstaste **F7** abgerufen werden.

**Tipp:** Wenn Sie alle eingegebenen Kommandos einer PowerShell-Sitzung in einer Datei speichern möchten, so beginnen Sie diese Sitzung mit der Anweisung

```
Start-Transcript Demo.txt
```

## 4.2 PowerShell-Provider

Die PowerShell verwendet mehrere Provider. In jedem Provider können dieselben cmdlets verwendet werden:

```
[PS] C:\>Get-PSProvider
Name                               Capabilities                               Drives
----                               -
WSMan                               Credentials                               {WSMan}
Alias                               ShouldProcess                             {Alias}
Environment                         ShouldProcess                             {Env}
FileSystem                           Filter, ShouldProcess                     {C, L, D, E...}
Function                             ShouldProcess                             {Function}
Registry                             ShouldProcess, Transactions               {HKLM, HKCU}
Variable                             ShouldProcess                             {Variable}
Certificate                          ShouldProcess                             {cert}
```

Anwendung:

```
[PS] C:\>cd hklm:\software
[PS] HKLM:\software>dir
    Hive: Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\software
SKC  VC Name                               Property
---  --  ---
   4   0 ATI Technologies                     {}
  176  0 Classes                               {}
    6   0 Clients                             {}
    1   0 Gemplus                               {}
    2   0 Intel                                 {}
   93  0 Microsoft                             {}
    2   0 ODBC                                 {}
    1   0 Policies                               {}
    0   1 Program Groups                       {ConvertedToLinks}
    1   0 Realtek                               {}
    1   0 Schlumberger                           {}
    0   0 Secure                                 {}
   17  0 Wow6432Node                             {}
[PS] C:\>cd cert:\CurrentUser\CA
[PS] cert:\CurrentUser\CA>dir
    Directory: Microsoft.PowerShell.Security\Certificate::CurrentUser\CA
Thumbprint                               Subject
-----
FEE449EE0E3965A5246F000E87FDE2A065FD89D4 CN=Root Agency
8B24CD8D8B58C6DA72ACE097C7B1E3CEA4DC3DC6 OU=www.verisign.com/CPS
Incorp.by ...
7B02312BACC59EC388FEAE12FD277F6A9FB4FAC1 CN=VeriSign Class 2 CA -
Individua...
12519AE9CD777A560184F1FBD54215222E95E71F CN=VeriSign Class 1 CA
Individual ...
109F1CAED645BB78B3EA2B94C0697C740733031C CN=Microsoft Windows Hardware
Comp...
```

## 4.3 Pipelines

Unter dem Begriff Pipelining versteht man die Möglichkeit, in einer Befehlszeile mehrere PowerShell-Cmdlets verkettet auszuführen.

Dabei wird die Ausgabe des ersten Cmdlets als Eingabe für das zweite Cmdlet verwendet.

PowerShell ist dabei – anders als die normale CommandShell – in der Lage, sowohl reinen Text als auch Objekte zu übergeben.

## 4.4 Filter

Filter werden durch Pipelining einer Ausgabe mit dem CmdLet where programmiert. Dabei ist auch die Verwendung von "Regular Expressions" möglich.

```
gps | where {$_.handles -gt "100"}
gps | where {$_.handles -gt "100"} | sort
stop-process -N *s* -WhatIf
stop-process -N *s* -confirm
```

WhatIf führt das Kommando nicht durch, sondern zeigt an, was passieren würde.

Confirm verlangt führt mehrere Prozess-Schritte Einzelbestätigungen.

## 4.5 Variablen

PowerShell-Variablen sind in der Lage, **Objektreferenzen** oder **Werte** zu speichern.

Variablennamen müssen mit dem \$-Zeichen beginnen.

Variablen können auch das Ergebnis enthalten, das die Ausführung eines CmdLets gebracht hat.

Variablen können auch als Eingabeparameter für CmdLets verwendet werden.

### Variablentypen:

- **Automatische Variablen:** Diese Variablen werden vom System verwaltet, ihr Wert kann nicht geändert werden.

Beispiele:

```
$PsHome      Verzeichnis, in welchem PowerShell installiert ist
$PID         Process ID (Prozesskennung)
$Home
```

- **Einstellungsvariablen:** Diese Variablen werden verwendet, um PowerShell-Voreinstellungen zu setzen.

Beispiele:

```
$MaximumHistoryCount      Wie viele PowerShell-Zeilen sollen in der
Befehlshistory gehalten   werden (Standardwert = 64)?
$errorActionPreference     Wie soll PowerShell im Fehlerfall reagieren?
```

- **Umgebungsvariablen:** Diese Variablen sind identisch mit den Variablen, die in der klassischen CommandShell über die Anweisung **set** angezeigt werden können.

Beispiele:

```
$env:WinDir           In welchem Verzeichnis befinden sich die Windows-
Systemdateien?
$env:ComputerName    Wie lautet der Computername des PCs, auf dem PowerShell
ausgeführt wird?
```

### Datentypen von Variablen:

- Einfache Werte:
 

```
$a = 5
```
- Mehrfache Werte:
  - Arrays (Wertbereiche):
 

Beispiel: Die Variable \$dateien enthält eine Liste der Einträge im Stammverzeichnis des Laufwerks c: Die Methode count liefert die Anzahl der gespeicherten Einträge.

```
$dateien = dir c:\
$dateien.count
```
  - Hash-Tabelle
 

Beispiel:

```
$Roles = @{ "DC1"="Server 2008 R2 DC"; "DC2"="Server 2008 R2
RODC" }
$Roles.DC2
```

Datentypen können implizit oder explizit festgelegt werden

- **Implizit:** Es ist keine Angabe nötig, welcher Datentyp verwendet werden soll. Der Datentyp ergibt sich aus der ersten Wertzuweisung. Im folgenden Beispiel wird eine Variable mit dem Wert 10 belegt, ohne dass der Datentyp angegeben wird. Die Methode GetType liefert den korrekten Datentyp **Int32**.

Beispiel:

```
$var = 10
$var.GetType()
```

- **Explizit:** Hier wird angegeben, welchen Datentyp die Variable haben soll.

Beispiel:

```
[DateTime]$today = "08/26/2010"
[int]$num = 123
[string]$str = "PowerShell"
```

## 4.6 Powershell-Scripts

Scripts werden mit der Erweiterung \*.ps1 (kommt von PowerShell 1.0) gespeichert. Scripts werden standardmäßig nicht ausgeführt, da eine Sicherheitseinstellung ausschließlich die Ausführung digital signierter Scripts erlaubt.

Um die Ausführungsrichtlinie zu ändern, verwenden Sie das CmdLet **set-ExecutionPolicy**.

**Set-ExecutionPolicy remotesigned**

**Set-ExecutionPolicy unrestricted**

Scripts, die in der PowerShell-Kommandozeile geschrieben werden, sind nur für die Zeitdauer der aktiven Sitzung verfügbar.

## 4.7 PowerShell-Profile

PowerShell-Profile sind spezielle Scripts, die beim Starten der PowerShell ausgeführt werden.

Für die Erstellung solcher Profile gibt es mehrere Möglichkeiten:

- Für jeden User oder für jedes System
- Für jede PS-Konsole oder für alle Konsolen

PowerShell kann durch Module erweitert werden. Diese Erweiterungen werden nicht automatisch geladen. Eine Liste aller verfügbaren (d.h. installierten) Module kann folgendermaßen aufgerufen werden:

```
PS C:> Get-Module -ListAvailable
ModuleType Name                               ExportedCommands
-----
Manifest   ActiveDirectory                         {}
Manifest   AppLocker                               {}
Manifest   BitsTransfer                             {}
Manifest   FailoverClusters                       {}
Manifest   GroupPolicy                             {}
Manifest   PSDiagnostics                           {}
Manifest   TroubleshootingPack                    {}
```

Die hier angeführten Module könnten zusätzlich geladen werden.

Die Systemvariable \$profile beinhaltet das aktuelle Profilscrip.

Um ein neues Profil zu erzeugen, geben Sie folgendes CmdLet ein:

```
PS C:> New-Item $profile -type file -force
```

Mit notepad.exe erstellen Sie dann den Inhalt Ihres neuen Profils:

```
Import-module ActiveDirectory
```

Das ActiveDirectory-Modul fügt einen neuen PowerShell-Provider hinzu, der unter der Bezeichnung AD: erreichbar ist.

```
PS C:\> cd AD:
PS AD:\> dir
Name                               ObjectClass                       DistinguishedName
----                               -
zahler                             domainDNS                         DC=zahler,DC=at
Configuration                       configuration
CN=Configuration,DC=zahler,DC=at
Schema                               dMD
CN=Schema,CN=Configuration,DC=zahl...
DomainDnsZones                      domainDNS
DC=DomainDnsZones,DC=zahler,DC=at
ForestDnsZones                      domainDNS
DC=ForestDnsZones,DC=zahler,DC=at
```

Wie hier zu sehen ist, werden alle verfügbaren Namenskontexte angezeigt. Es ist nun möglich, durch diese Namenskontexte zu navigieren und die jeweils enthaltenen Active Directory-Objekte anzuzeigen.

## 4.8 Remote-Ausführung von PowerShell-Scripts und -CmdLets

Seit Windows 7/Windows Server 2008 R2 unterstützt PowerShell auch die Ausführung von CmdLets und Scripts auf entfernten Rechnern.

Voraussetzung für eine erfolgreiche Ausführung ist allerdings, dass bestimmte Firewall-Ausnahmeregeln gesetzt werden müssen.

Einige CmdLets unterstützen diese Remote-Ausführungsfunktionalität bereits standardmäßig:

- Get-Process
- Get-Service
- Get-WinEvent

Man kann herausfinden, welche CmdLets netzwerkfähig sind, indem man nach CmdLets sucht, die den Parameter ComputerName unterstützen:

```
Get-Help * -parameter ComputerName
```

Beispiel:

```
Get-Service -ComputerName dc01 | where {$_.status -eq „Stopped“}
```

Diese Anweisung liefert alle gestoppten Dienste auf dem Computer dc01.

```
Get-Service -ComputerName dc01 -name vss | fl *
```

Diese Anweisung liefert detaillierte Informationen zum Volume Shadow Copy-Dienst, der auf dc01 läuft.

```
gwmi win32_service -ComputerName dc01 | where {$_.name -like "*vss*"}
```

Diese Anweisung ist eine Alternative zum Beispiel vorher. Hier wird der WMI-Provider abgefragt, um detaillierte Informationen zum Volume Shadow Copy-Dienst zu liefern, der auf dc01 läuft.

Alle „Nicht-Remotefähigen“ CmdLets und Scripts können über eine **WinRM-Sitzung** (Windows Remote Management) ausgeführt werden.

Dafür sind folgende Voraussetzungen nötig:

- .NET-Framework 2.0 oder höher
- PowerShell 2.0
- Windows Remote Management 2.0

Es besteht die Möglichkeit, zwei Typen von WinRM-Sitzungen herzustellen:

- Einmalige Sitzung
- Permanente Sitzung

Schnellkonfiguration:

```
Enable-PSRemoting
```

Einmalige Sitzung:

```
Invoke-Command -ComputerName pc01, dc02 (get-process | sort-object CPU -  
desc | select first 5)
```

Für das CmdLet Invoke-Command gibt es auch den Alias **icm**.

Dauerhafte Sitzung:

```
$pc01 = New-PsSession -ComputerName pc01 -Credential domain\administrator  
Get-PsSession  
Enter-PsSession $pc01  
...  
Exit-PsSession
```

Hinweis: In einer permanenten Sitzung sind keine PowerShell-Module (außer die beiden immer geladenen Basismodule) verfügbar.

## 4.9 Hilfe

Um Hilfe zu erhalten, geben Sie folgendes ein:

```
[PS] C:\>Get-Help
```

Nun wird Ihnen eine komplette Liste angezeigt, wie Sie Hilfe und Erklärungen zu allen Microsoft Shell Befehlen erhalten. Geben Sie zum Beispiel folgendes ein, um alle automatischen Variablen anzuzeigen:

```
[PS] C:\>Get-Help about_Automatic_variables
```

NAVIGATION:

Um eine Liste aller verfügbaren Laufwerke anzuzeigen geben Sie folgendes ein:

```
[PS] C:\>Get-Drive
```

Wenn Sie zu einem anderem Laufwerk wechseln möchten, zB. auf D: müssen Sie anstatt des „cd“-Befehls mit folgendem Befehl arbeiten:

```
[PS] C:\>Set-Location D:\
```

Um alle verfügbaren CMDLETs anzuzeigen, geben Sie folgendes ein:

```
[PS] C:\>Get-Command
```

Um alle verfügbaren „export“- Befehle anzuzeigen, geben Sie folgendes ein:

```
[PS] C:\>Get-Command export-*
```



## 5 Skriptsprachen unter Windows Server 2008 R2

Früher war die einzige systemeigene Skriptsprache, die vom Betriebssystem Windows unterstützt wurde, die **Batchdateisprache**. Die Batchsprache ist zwar schnell und klein, ihre Funktionen sind jedoch eingeschränkt. Die Batchsprache enthält z. B. keine Möglichkeit, den Programmfluss zu steuern.

Dies hat Microsoft bewogen, ab Windows 2000 die neue **Windows Script Host-Architektur** einzuführen.

Es gibt zwei Versionen von Windows Script Host:

- eine Windows-basierte Version (**Wscript.exe**), die ein Eigenschaftenblatt zum Festlegen der Skripteigenschaften bereitstellt, und
- eine Befehlszeilenversion (**Cscript.exe**), die Befehlszeilenoptionen zum Festlegen der Skripteigenschaften bereitstellt.

Sie können die gewünschte Version ausführen, indem Sie an der Eingabeaufforderung entweder Wscript.exe oder Cscript.exe eingeben.

Trotzdem werden Batchdateien weiterhin unterstützt.

