

BBZW Sursee



In Kooperation mit dem HERDT-Verlag stellen wir Ihnen eine PDF inkl. Zusatzmedien für Ihre persönliche Weiterbildung zur Verfügung. In Verbindung mit dem Programm HERDT-Campus ALL YOU CAN READ stehen diese PDFs nur Lehrkräften und Schüler\*innen der oben genannten Lehranstalt zur Verfügung. Eine Nutzung oder Weitergabe für andere Zwecke ist ausdrücklich verboten und unterliegt dem Urheberrecht. Jeglicher Verstoß kann zivil- und strafrechtliche Konsequenzen nach sich ziehen.

---

**Windows  
PowerShell 5.1**

---

Andreas Dittfurth

1. Ausgabe, Juni 2021

ISBN 978-3-98569-000-8

Grundlagen und Verwaltung  
des Active Directory

WPOW51



<b>Bevor Sie beginnen ...</b>	<b>4</b>	5.4 Aliase löschen	77
		5.5 Kurz zusammengefasst	78
		5.6 Übung	79
<b>Grundlagen zu PowerShell</b>			
<b>1 Die PowerShell kennenlernen</b>	<b>5</b>	<b>6 Profile</b>	<b>80</b>
1.1 Grundlagen zur PowerShell	5	6.1 Profil als Gedächtnis der PowerShell	80
1.2 PowerShell-Konsole und PowerShell ISE	6	6.2 Anlegen eines Benutzerprofils	82
1.3 Automatische Vervollständigung	8	6.3 Hintergrund: Ausführungsrichtlinie für Skripte	84
1.4 PowerShell starten	9	6.4 Kurz zusammengefasst	89
1.5 Die PowerShell-Konsole individuell konfigurieren	13	6.5 Übung	89
1.6 Bekannte Befehle – leichter Einstieg	17		
1.7 Konsole oder ISE einsetzen?	20		
1.8 Übung	20		
<b>2 PowerShell-Cmdlets</b>	<b>21</b>	<b>7 Programmiergrundlagen</b>	<b>90</b>
2.1 Grundlagen zu PowerShell-Cmdlets	21	7.1 Variablen	90
2.2 Parameter	22	7.2 Arrays – spezielle Variablen mit Wertelisten	96
2.3 Allgemeine Parameter	24	7.3 Konstanten	98
2.4 Get-Cmdlets für den Einstieg	26	7.4 Arithmetische Operatoren	98
2.5 Das Hilfesystem der PowerShell	31	7.5 Vergleichsoperatoren	100
2.6 Kurz zusammengefasst	35	7.6 Kontrollstrukturen in der PowerShell	102
2.7 Übungen	35	7.7 Die einfache <b>If</b> -Anweisung	102
<b>3 Informationen verarbeiten und ausgeben</b>	<b>36</b>	7.8 Die <b>If</b> -Anweisung mit <b>Else</b> -Zweig	103
3.1 Die PowerShell-Pipeline	36	7.9 Erweiterte <b>If</b> -Anweisung mit <b>ElseIf</b>	104
3.2 Verarbeitung vorliegender Daten: Object-Cmdlets	37	7.10 Fallauswahl mit der <b>Switch</b> -Anweisung	105
3.3 Formatierung und Ausgabe	46	7.11 Schleifen	106
3.4 Allgemein: Schrittweise Entwicklung einer Pipeline	54	7.12 Mit der <b>While</b> -Schleife arbeiten	107
3.5 Einsatztipps für die Pipeline	55	7.13 Mit der <b>Do-While</b> -Schleife arbeiten	108
3.6 Kurz zusammengefasst	56	7.14 Mit der <b>For</b> -Schleife arbeiten	109
3.7 Übung	56	7.15 Einsatz der <b>ForEach</b> -Schleife	109
<b>4 Datenspeicher in der PowerShell</b>	<b>57</b>	7.16 Anweisungen zur Ablaufsteuerung: <b>break</b> und <b>continue</b>	111
4.1 PowerShell-Provider	57	7.17 Kurz zusammengefasst	111
4.2 PowerShell-Laufwerke	58	7.18 Übungen	112
4.3 Cmdlets für die Arbeit mit Verzeichnissen	61		
4.4 Cmdlets für die Arbeit mit Elementen und ihren Eigenschaften	65		
4.5 Laufwerke im Dateisystem (Provider FileSystem)	68		
4.6 Registry-Laufwerke (Provider Registry)	71		
4.7 Laufwerk für Umgebungsvariablen (Provider Environment)	72		
4.8 Kurz zusammengefasst	73		
4.9 Übung	73		
<b>5 Aliase – alternative Kurzbefehle</b>	<b>74</b>	<b>8 Skripting, Funktionen, Filter</b>	<b>113</b>
5.1 Alias (Spitzname)	74	8.1 PowerShell ISE verwenden	113
5.2 Vordefinierte und eigene Aliase	74	8.2 Funktionen in der PowerShell	114
5.3 Ex- und Import von Aliasen	76	8.3 Einfache Funktionen erstellen	114
		8.4 Funktionen mit Parametern erstellen	115
		8.5 Standardwert eines Parameters vorgeben	116
		8.6 Funktionen mit Switch-Parametern	117
		8.7 Objekte über die Pipeline an eine Funktion übergeben	118
		8.8 Objekte über die Pipeline an ein Skript übergeben	121
		8.9 Filter	122
		8.10 Das virtuelle Laufwerk <i>Function</i> :	123
		8.11 Eigene Programmierung dokumentieren	123
		8.12 Funktionen und Filter allgemein verfügbar machen	125
		8.13 Kurz zusammengefasst	126
		8.14 Übung	127

<b>Verwaltung von Active Directory-Domänen</b>	
<b>9 PowerShell-Module</b>	<b>128</b>
9.1 Was sind PowerShell-Module?	128
9.2 Mit Modulen arbeiten	128
9.3 Das Modul <i>ServerManager</i>	133
9.4 Das Modul <i>DnsServer</i>	135
9.5 Das Modul <i>ServerCore</i>	136
9.6 Das Modul <i>ActiveDirectory</i>	137
9.7 Das virtuelle Laufwerk AD:	138
9.8 Lokale Benutzer und Gruppen verwalten	140
9.9 Kurz zusammengefasst	142
9.10 Übung	142
<b>10 Active Directory verwalten</b>	<b>143</b>
10.1 Cmdlets für die Verwaltung von Active Directory-Basisobjekten	143
10.2 Informationen auslesen: Benutzer, Gruppen, OUs und Computer	143
10.3 Benutzer, Gruppen, OUs oder Computer erstellen	145
10.4 Eigenschaften von Benutzern, Gruppen, OUs oder Computern ändern	148
10.5 Benutzer, Gruppen, OUs oder Computer löschen	151
10.6 Allgemeine Objektverwaltung	151
10.7 Schutz vor versehentlichem Löschen	153
10.8 Das Active Directory-Verwaltungscenter	156
10.9 Kurz zusammengefasst	158
10.10 Übung	158
<b>11 Weitere Angaben im Active Directory</b>	<b>159</b>
11.1 Arbeit mit fein abgestimmten Kennwortrichtlinien	159
11.2 Verwaltung von Active Directory-Konten	165
11.3 Betriebsmasterrollen mit der PowerShell verwalten	170
11.4 Kurz zusammengefasst	173
11.5 Übung	173
<b>12 Active Directory-Papierkorb</b>	<b>175</b>
12.1 Das Prinzip des Active Directory-Papierkorbs	175
12.2 Gesamtstrukturfunktionsebene ermitteln und ggf. ändern	175
12.3 Aktivieren des Active Directory-Papierkorbs	177
12.4 Der Active Directory-Papierkorb im Active Directory-Verwaltungscenter	178
12.5 Gelöschte Objekte mit der PowerShell finden	179
12.6 Gelöschte Objekte mit der PowerShell wiederherstellen	180
12.7 Was tun, wenn auch das übergeordnete Objekt gelöscht wurde?	181
12.8 Kurz zusammengefasst	182
12.9 Übung	183
<b>13 Sonderaufgaben für die PowerShell</b>	<b>184</b>
13.1 Geplante Aufgaben mit der PowerShell verwalten	184
13.2 IP-Konfiguration: PowerShell anstelle von <i>netsh.exe</i> verwenden	185
13.3 Server Core: PowerShell als Standard-Shell einrichten	188
13.4 Windows Explorer aus der PowerShell heraus starten	190
13.5 Startzeitpunkt und Laufzeit eines Servers bestimmen	191
13.6 Kurz zusammengefasst	194
13.7 Ausblick: PowerShell – next level	194
13.8 Übung	195
<b>14 Praxiskapitel: Automatisierung in der Domäne</b>	<b>196</b>
14.1 Installation einer neuen Gesamtstruktur	196
14.2 Domänencontroller einer Domäne hinzufügen	199
14.3 Aufbau einer Organisationseinheiten-Struktur	203
14.4 Automatisierter Import neuer Benutzer	206
<b>A Anhang: Testumgebung einrichten</b>	<b>211</b>
A.1 Download der benötigten Software	211
A.2 Hyper-V unter Windows 10 (Enterprise) aktivieren	211
A.3 Hyper-V einrichten	213
A.4 Virtuellen Computer erstellen und betreiben	216
A.5 Testanordnung für die Übungen	218
A.6 Mehrere virtuelle Computer in einem gemeinsamen virtuellen Netzwerk betreiben	219
A.7 Windows Server 2016 als Domänencontroller einrichten	222
A.8 Andere virtuelle Maschinen der Domäne hinzufügen	224
<b>Stichwortverzeichnis</b>	<b>228</b>

# Bevor Sie beginnen ...

## **HERDT** BuchPlus – unser Konzept:

Problemlos einsteigen – Effizient lernen – Zielgerichtet nachschlagen

Nutzen Sie dabei unsere maßgeschneiderten, im Internet frei verfügbaren Medien:



Wie Sie schnell auf diese BuchPlus-Medien zugreifen können, erfahren Sie unter [www.herd़t.com/BuchPlus](http://www.herd़t.com/BuchPlus)

### Hinweise zu Soft- und Hardware

Die Windows PowerShell ist fest in die modernen Betriebssysteme Windows 10 und Windows Server 2016/2019 integriert. Ihr Rechner muss in der Lage sein, mit den Betriebssystemen zu arbeiten bzw. diese Betriebssysteme im Rahmen einer Virtualisierungssoftware zu betreiben.

Alle im Buch vorgestellten Aktionen können Sie ohne weitere Software durchführen. Damit Sie alle vorgestellten Übungen mit der PowerShell direkt an Ihrem Rechner durchführen können, benötigen Sie die folgenden Betriebssysteme:



- ✓ Windows 10 Enterprise (90-Tage-Testversion), Download unter <https://www.microsoft.com/de-de/evalcenter/evaluate-windows-10-enterprise>
- ✓ Windows Server 2019 (180-Tage-Testversion), Download unter <https://www.microsoft.com/de-de/evalcenter/evaluate-windows-server-2019?filetype=ISO>
- ✓ Optional: Oracle VirtualBox (freie Virtualisierungssoftware, falls Sie Hyper-V nicht einsetzen können oder wollen), Download unter <https://virtualbox.org/wiki/Downloads>

Auch wenn Sie bereits die genannten Betriebssysteme einsetzen, ist der Aufbau einer virtuellen Testumgebung empfehlenswert, damit Sie nicht in Ihrer Produktivumgebung testen müssen. Die PowerShell ist sehr mächtig, deshalb ist in der Lernphase eine geschützte Testumgebung vorzuziehen. Installations- und Konfigurationshinweise zur Testumgebung mit Microsoft Hyper-V finden Sie im Anhang.

### Typographische Konventionen

**Kursivschrift:** alle von Programmen vorgegebenen Bezeichnungen (Schaltflächen, Dialogfenster, Symbolleisten, Menüs und Menüpunkte (z. B. *Datei - Schließen*) sowie alle vom Anwender zugewiesene Namen. Schriftart Courier New: Programmtext und -namen, Funktions- und Variablenamen, Datentypen, Operatoren, zitierte Programmausgaben etc. *Courier New*: Kursive Passagen in dieser Schriftart kennzeichnen optionalen Code. Bei Darstellungen der Syntax einer Programmiersprache sowie PowerShell-Sprachelementen kennzeichnen eckige Klammern [ ] optionale Angaben. Alternative Elemente in der Syntax werden durch einen Schrägstrich / voneinander getrennt, spitze Klammern <> kennzeichnen Platzhalter.

### Was bedeuten die Symbole im Randbereich?



Praxistipp



Warnhinweis



Download

# 1 Die PowerShell kennenlernen

## 1.1 Grundlagen zur PowerShell

### Was ist die PowerShell?

Die Windows PowerShell (nachfolgend kurz: PowerShell) ist eine junge Befehlszeilenkonsole von Microsoft und Skriptsprache für die System- und Netzwerkverwaltung und -automatisierung. Sie wurde erstmalig Ende 2006 mit dem Microsoft Exchange Server 2007 ausgeliefert.

Microsoft bezeichnet die PowerShell als Basis des Betriebssystems. Administrative Aufgaben, die Sie in der grafischen Oberfläche durchführen, werden im Hintergrund in PowerShell-Befehle übersetzt. Das Konzept wurde konsequent umgesetzt, so dass mittlerweile (fast) alle administrativen Aufgaben mithilfe der PowerShell durchgeführt werden können. Das ist vor allem dann interessant, wenn es keine grafische Alternative zur Durchführung der Aufgabe gibt oder Sie eine Automatisierung anstreben.

Die PowerShell arbeitet objektorientiert und basiert auf dem Microsoft .NET-Framework. Es handelt sich trotz ähnlichen Aussehens nicht um eine Weiterentwicklung der Windows-Eingabeaufforderung (*cmd.exe*), sondern basiert auf einer anderen Technik. Die PowerShell nimmt Konzepte von Unix-Shells auf, arbeitet mit Befehlsverbindungen und Filtern, bietet aber auch die Möglichkeit, eigene Skripte zu schreiben und damit Abläufe zu automatisieren. Letztlich folgt die PowerShell mit ihrer *PowerShell Scripting Language* in erster Linie dem Ansatz der objektorientierten Programmierung.

### PowerShell-Versionen

Windows Server 2008 war das erste Betriebssystem, das mit der PowerShell Version 1 ausgeliefert wurde. Nachträglich wurden Downloads für Windows Vista, Windows XP und Windows Server 2003 angeboten. Seit dieser Zeit ist die PowerShell in jedem veröffentlichten Betriebssystem von Microsoft enthalten.

PowerShell Version 2 wurde gegenüber der Version 1 funktional stark erweitert und ist vorinstalliert in den Betriebssystemen Windows 7 und Windows Server 2008 R2.

PowerShell Version 3 ist automatisch in den Editionen der Betriebssysteme Windows 8 und Windows Server 2012 enthalten und damit bei beiden Betriebssystemen identisch. Die PowerShell Version 3 steht als Teil des Windows Management Frameworks 3.0 für ältere Betriebssysteme als Download zur Verfügung.

PowerShell Version 4 wurde im Oktober 2013 als finale Version veröffentlicht. Analog zu Version 3 ist die PowerShell Version 4 bereits in allen Editionen der Betriebssysteme Windows 8.1 und Windows 2012 R2 enthalten (und Bestandteil des Windows Management Frameworks 4.0).

Die Version 5.0 der PowerShell wurde ab April 2014 in einer Vorabversion des Management Frameworks 5.0 veröffentlicht. Wie bei den Vorgängerversionen ist PowerShell 5 in allen Editionen aktueller Windows-Betriebssysteme enthalten – hier Windows 10 und Windows Server 2016.

Das im Juli 2015 erschienene Betriebssystem Windows 10 enthält die Version 5.0 der PowerShell. Nach Installation des **Windows 10 Anniversary Updates** wird die PowerShell auf Version 5.1 aktualisiert. Mit Windows Server 2016 (verfügbar seit Oktober 2016) wird die PowerShell-Version 5.1 ausgeliefert.

Die Windows PowerShell ist also seit Oktober 2016 in allen aktuellen Windows-Betriebssystemen unter der Bezeichnung Windows PowerShell in der Version 5.1 vorinstalliert.

Zwischenzeitlich sorgten Meldungen für Verunsicherung, dass die Windows PowerShell nicht weiterentwickelt werde. Nutzer, denen die PowerShell mittlerweile ans Herz gewachsen war, befürchteten, dass dies eine Abkehr von der PowerShell bedeuten würde. Dies ist keineswegs der Fall. Wer sich die Äußerungen von Microsoft genauer anschaut, bemerkt schnell, dass Microsoft etwas anderes im Sinn hat: die Entwicklung der PowerShell als neue Sprache wird als abgeschlossen betrachtet. Es wird als vorinstallierte Komponente in Windows-Betriebssystemen keine andere Version mehr geben als die Version 5.1, die uns mittlerweile seit 2016 begleitet. Der Funktionsumfang der PowerShell orientiert sich ohnehin primär an den installierten Komponenten Ihres Rechners und zusätzlich installierten Modulen aus dem Internet.

Nach Abschluss der Arbeiten für die so gut in Windows angebundene und auf .NET Framework basierende Windows PowerShell überraschte Microsoft mit einer PowerShell-Variante, die unter der Bezeichnung PowerShell Core 6 veröffentlicht wurde. Hier handelt es sich um keine Weiterentwicklung der Windows PowerShell, sondern um eine plattformübergreifende Open Source-Variante für Windows, macOS und Linux, die auf .NET Core basiert.

Im Jahr 2020 erschien PowerShell 7. Das Suffix Core wurde aus dem Namen gestrichen. Die Kompatibilität zu Modulen aus in Windows vorhandenen PowerShell-Modulen wurde verbessert.

Auch wenn die Windows PowerShell (bis Version 5.1) und PowerShell (Core; Versionen 6 und 7) zur selben Familie gehören, sind ihre Einsatzzwecke unterschiedlich. Die Windows PowerShell glänzt durch ihre perfekte Einbindung in Windows, PowerShell durch allgemeinere Themen und ihre Einsatzmöglichkeiten auf verschiedenen Betriebssystemen.

Wer ausschließlich in Windows-Netzwerken mit Windows-Rechnern unterwegs ist, verwendet nach wie vor primär die Windows PowerShell. PowerShell 7 ist eine gute Wahl, wenn eine heterogene Umgebung oder eine entsprechende Cloud-Umgebung zu verwalten und automatisieren ist.

Im vorliegenden Buch wird daher ausschließlich die Windows PowerShell verwendet.

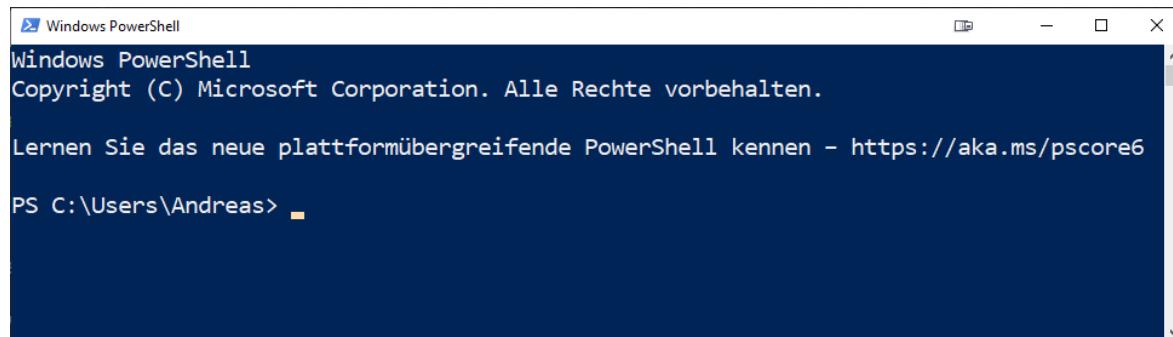
## 1.2 PowerShell-Konsole und PowerShell ISE

Die PowerShell steht als interaktive Eingabeaufforderung (Konsole; `powershell.exe`) und als Version mit integrierter Skriptumgebung (PowerShell Integrated Scripting Environment; `powershell_ise.exe`) zur Verfügung.

### Die PowerShell-Konsole

Die Konsole ähnelt stark den bekannten DOS-Fenstern. Bis auf den weiter unten ausführlich erläuterten QuickEdit-Modus, der das Kopieren und Einfügen von Textbausteinen erleichtert, begegnen Ihnen Aussehen und Handhabung der Windows-Eingabeaufforderung.

Bei einer Anwendung, die Texteingaben von Ihnen erwartet, ist das nicht weiter verwunderlich. Im Laufe der Kapitel werden Sie erkennen, dass ein erster Eindruck täuschen kann und in der PowerShell weit mehr steckt.



PowerShell-Konsolenfenster (Windows 10)

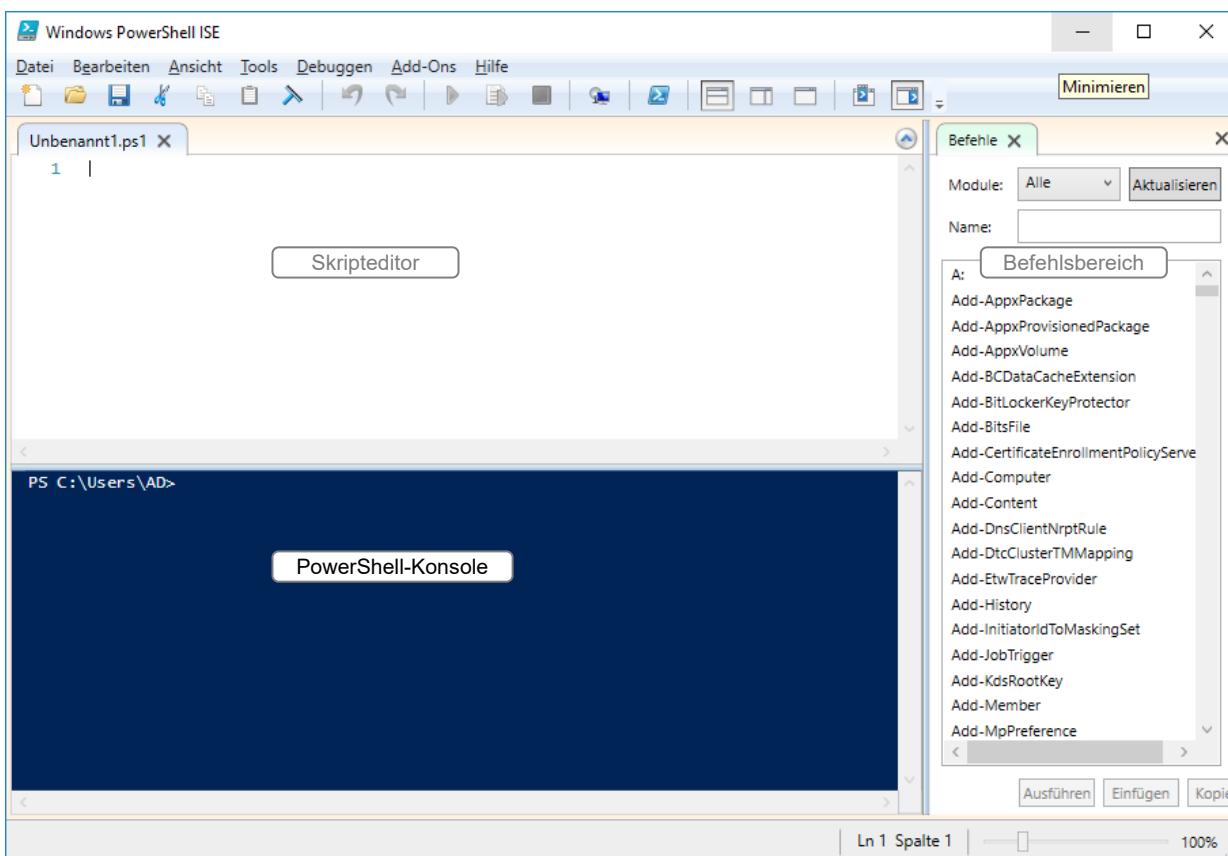
## Wofür die PowerShell-Konsole eingesetzt wird

Die PowerShell-Konsole wird häufig für das schnelle Erledigen einer Aufgabe verwendet. Im Vordergrund steht der interaktive Einsatz: Sie geben einen Befehl ein, der sofort ausgeführt wird und – sofern vorhanden – umgehend die entsprechenden Informationen zurückgibt. In der Konsole lassen sich auch Skripte ausführen.

Im Alltag ist sie das primäre Werkzeug als „echte“ Konsole, die wenig Speicherplatz benötigt und ohne grafische Zusätze auskommt.

## PowerShell Integrated Scripting Environment (ISE)

PowerShell ISE ist ein Skripteditor, den Microsoft seit PowerShell 2.0 mitliefert. In der PowerShell-Version 3.0 wurde PowerShell ISE stark überarbeitet und mit nützlichen Vereinfachungen für den Anwender ausgestattet. Die PowerShell ISE 5 weist im Vergleich zu ihren Vorgängern ab Version 3 nur marginale Änderungen auf. Eine interaktive Arbeit mit PowerShell ISE ist durch die integrierte Konsole ebenfalls möglich.



PowerShell ISE-Anwendungsfenster

Nach dem Start sehen Sie im kompletten linken Bereich des Fensters die integrierte PowerShell-Konsole sowie den Skripteditor. Im obigen Bild wurde der standardmäßig ausgeblendete Skripteditor bereits ausgeklappt. Im rechten Fensterbereich sehen Sie den Befehlsbereich, der Sie grafisch bei der Auswahl von Befehlen (und dazu gehörigen Parametern) unterstützt.

## Wofür PowerShell ISE eingesetzt wird

PowerShell ISE wird häufig für das Schreiben und Ausführen eigener Skripte verwendet. Zusätzlich bietet die PowerShell ISE eine ausfeilte, grafisch orientierte Hilfe, die z. B. von Lernenden gern verwendet wird. Damit bedient PowerShell ISE besonders die Enden des Spektrums: durch Benutzerfreundlichkeit einer Windows-Anwendung PowerShell-Lernende und erfahrene Anwender, die anspruchsvolleren Code entwickeln.

## 64-bit-Version und 32-bit-Version

In 64-bit-Versionen von Windows gibt es zusätzlich zur 64-bit-Version der PowerShell eine 32-bit-Version, die Sie im Verzeichnis `%SystemRoot%\SysWOW64\WindowsPowerShell\v1.0` finden. Das betrifft beide Anwendungen – PowerShell-Konsole und PowerShell ISE.

Die 32-bit-Version benötigen Sie nur dann, wenn die PowerShell auf Bibliotheken zugreift, für die es ausschließlich eine 32-bit-Version gibt. Dies ist z. B. beim Zusammenspiel mit Microsoft Access der Fall, begegnet Ihnen im Normalfall aber nicht. Ansonsten verhalten sich die PowerShell-Versionen identisch.



Sie haben richtig gelesen: Die 32-bit-Version der PowerShell befindet sich unterhalb des Verzeichnisses `SysWOW64`, während die 64-bit-Version in der Struktur unterhalb des Verzeichnisses `System32` zu finden ist.

## 1.3 Automatische Vervollständigung

Bei einer textbasierten Umgebung kann jeder Tippfehler verhindern, dass eingegebene Befehle so funktionieren, wie Sie erwarten. Die PowerShell unterstützt Sie bei der Eingabe und hilft, Eingabefehler zu vermeiden.

Sowohl die PowerShell-Konsole als auch die PowerShell ISE haben Mechanismen zur Erleichterung Ihrer Eingaben:

- ✓ Die PowerShell-Konsole hilft Ihnen mit der sogenannten Tabulator-Vervollständigung. Wenn Sie einen Teil eines Befehls oder eines Pfads eingeben, können Sie mit der Tabulatortaste eine Vervollständigung des Befehls oder Pfads auf Basis der bisher erfolgten Eingabe erreichen.
- ✓ Die PowerShell ISE stellt eine weitergehende Hilfe namens IntelliSense zur Verfügung. Grafisch orientierte Menüs helfen Ihnen automatisch bei der Vervollständigung einer Eingabe. Wird ein solches Hilfsmenü gerade nicht angezeigt, können Sie es durch die Tastenkombination einblenden.

### Tabulator-Vervollständigung in der PowerShell-Konsole

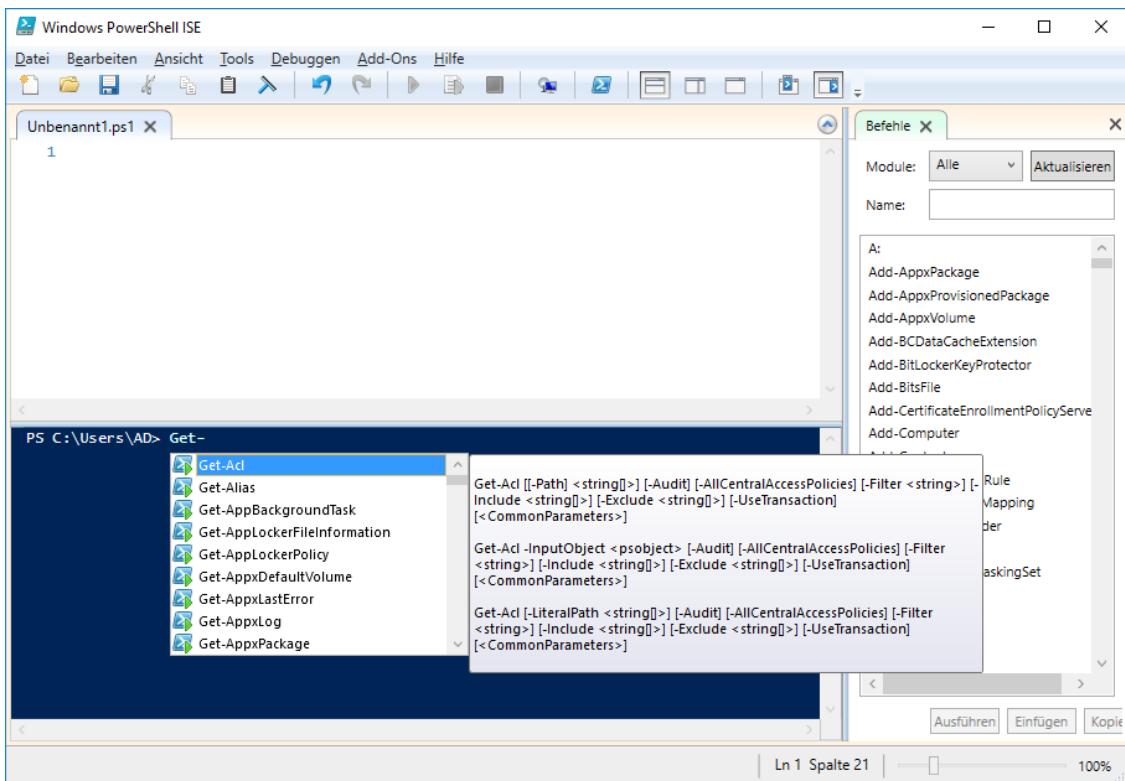
Wenn Sie die automatische Vervollständigung in der PowerShell-Konsole nutzen wollen, betätigen Sie mitten in einer unvollständigen Befehls- oder Pfadeingabe die Tabulatortaste .

- ✓ In einer Pfadangabe tippen Sie nur einen Teilpfad, z. B. `C:\`, und betätigen die Tabulatortaste . Mit jeder Betätigung von schlägt die PowerShell ein anderes Verzeichnis oder eine andere Datei im Stammverzeichnis des Laufwerks C: vor. Die Anzeige erfolgt alphabetisch aufsteigend. Je mehr Buchstaben Sie eintippen, desto genauer wird die Auswahl der PowerShell, da Ihre Eingabe als Vorgabe verwendet wird.
- ✓ Zur Vervollständigung von Befehlen tippen Sie einen Teil des Befehls und drücken die -Taste. Die ersten Befehle lernen Sie bereits in diesem Kapitel kennen. Probieren Sie diese Funktion aus und sparen sich Tipparbeit.

### IntelliSense in der PowerShell ISE

IntelliSense steht in der PowerShell ISE zur Verfügung, und zwar sowohl im dortigen Skripteditor als auch im Konsolenzonenbereich.

Hier erscheint automatisch bei einer Teileingabe von Befehlen und Pfaden ein Hilfsmenü, wie Sie in der folgenden Abbildung sehen können. Die dort präsentierten Angaben können viel Arbeit ersparen, da die Hilfen weitgehend sind: Pfade, Befehle, Parameter und Werte werden passend zur vorgenommenen Eingabe vorgeschlagen.



IntelliSense: grafisch orientierte automatische Vervollständigung in der PowerShell ISE

## 1.4 PowerShell starten

### Verwendete Betriebssysteme

Im ersten Teil des Buches arbeiten Sie mit Windows 10 (oder alternativ mit Windows Server 2016 bzw. 2019). In den Kapiteln des zweiten Teils verwenden Sie Windows Server 2016 (oder 2019) für Arbeiten mit Active Directory sowie der Netzwerkverwaltung.

Falls Sie die genannten Betriebssysteme nicht zur Verfügung haben, keine Administratorrechte besitzen oder eine Testumgebung anstelle der Produktivumgebung verwenden möchten, richten Sie sich eine Testumgebung mit Windows 10 und/oder Windows Server 2016 (oder 2019) ein. Eine Anleitung dazu finden Sie im Anhang.

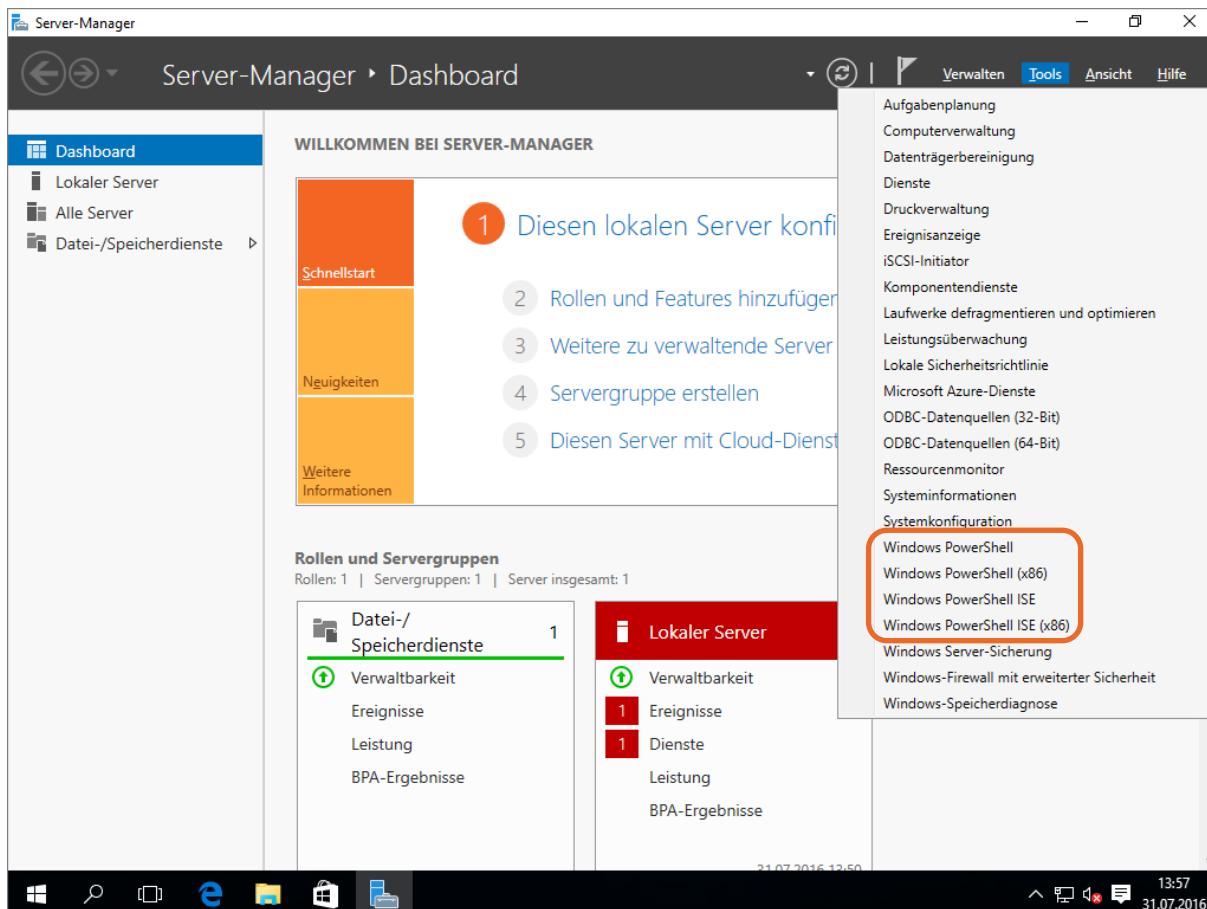
### PowerShell unter Windows Server 2016 / 2019

Die PowerShell zählt zu den wichtigsten Werkzeugen für den Administrator. Aus diesem Grund müssen Sie nicht lange nach der PowerShell suchen. PowerShell-Kacheln für Konsole und ISE sind bereits im Startmenü des Servers vorhanden.

Darüber hinaus ist die PowerShell in das zentrale Administrationsinstrument, den Server Manager, unter dem Menüpunkt *Tools* integriert.



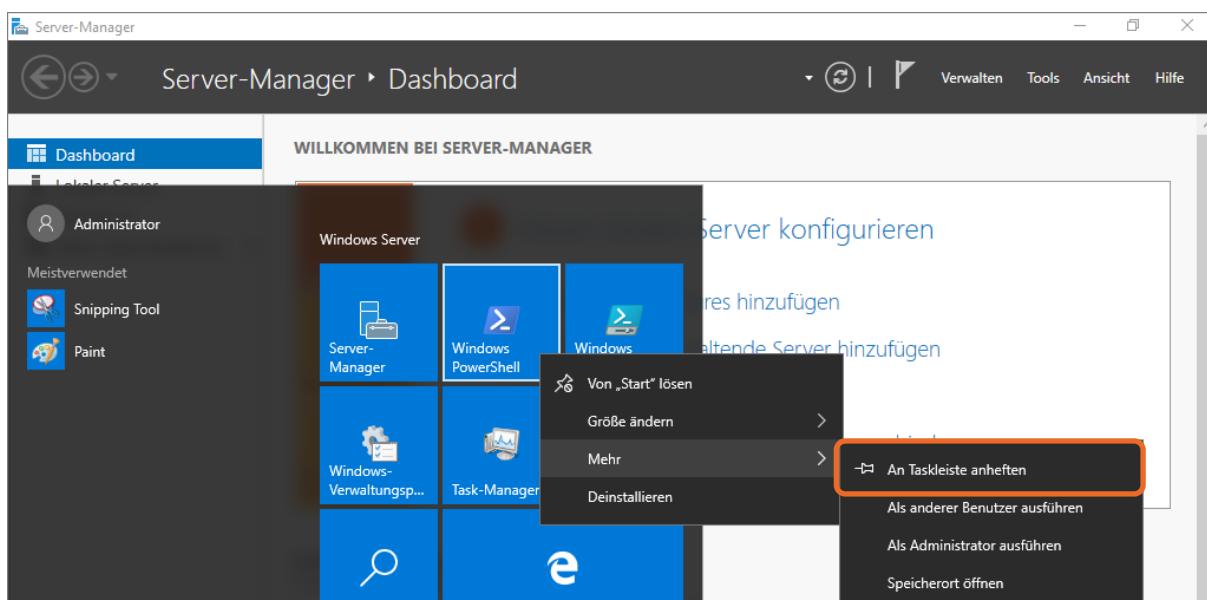
Windows Server 2016: PowerShell-Kacheln im Startmenü



Server Manager in Windows Server 2016: PowerShell-Einträge im Menü „Tools“

In Windows Server 2016 (oder auch Windows Server 2019) ist die PowerShell standardmäßig nicht mehr in der Taskleiste zu finden. Wenn Sie dies ändern möchten, führen Sie die folgenden Schritte aus:

- Öffnen Sie das Startmenü (über  in der Taskleiste).
- Klicken Sie mit der rechten Maustaste auf die Kachel *Windows PowerShell*, um das Kontextmenü zu öffnen.
- Rufen Sie den Eintrag *Mehr* auf und wählen Sie *An Taskleiste anheften*.

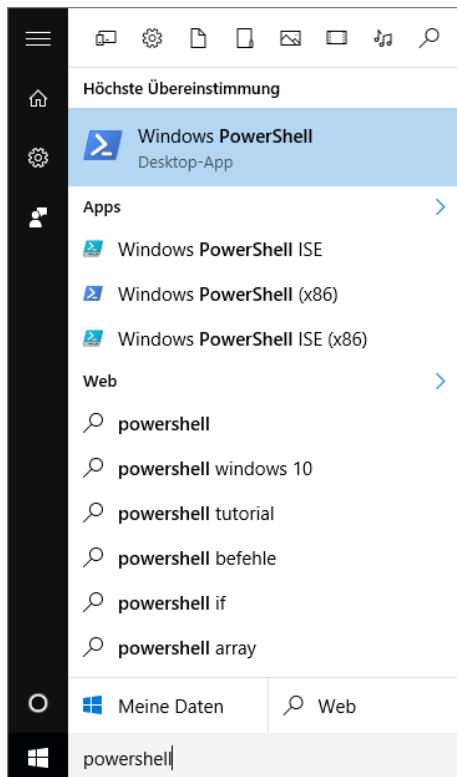


Windows Server 2016 (oder 2019): Die PowerShell an die Taskleiste anheften

## PowerShell unter Windows 10

In Windows 10 ist die PowerShell standardmäßig weder im Startmenü noch auf dem Desktop zu finden. Sie können die PowerShell jedoch auf verschiedene Wege aufrufen und sich die Umgebung so einrichten, dass keine weitere Suche nötig ist:

- ▶ Drücken Sie , um das Startmenü zu öffnen.
- ▶ Geben Sie die Zeichenfolge `powershell` in das Suchfeld ein, um im System nach Anwendungen mit dieser Zeichenfolge suchen zu lassen.  
Es reicht bereits aus, einen Teil der Zeichenkette einzugeben.  
Die Position des Mauszeigers spielt beim Start der Eingabe keine Rolle. Groß- und Kleinschreibung werden nicht unterschieden.  
Sie müssen auch kein Suchfenster öffnen, sondern können einfach lostippen.
- ▶ Klicken Sie mit der linken Maustaste auf das Suchergebnis *Windows PowerShell Desktop-App*.  
Die PowerShell wird in einem Konsolenfenster auf dem Desktop geöffnet.



Wenn Sie mit der rechten Maustaste auf den Eintrag im oben gezeigten Suchergebnis klicken, startet die Anwendung nicht, sondern es wird ein Kontextmenü geöffnet.



Sie können z. B. folgende Aktionen ausführen:

Aktion	Erläuterung	
An „Start“ anheften	Die Anwendung wird als Kachel im Startmenü angezeigt. Somit können Sie PowerShell leicht finden und durch einen einfachen Linksklick auf die Kachel öffnen.  Die Position der Kachel können Sie per Drag & Drop ändern. Durch einen Klick auf die Kachel mit der rechten Maustaste wird ein Kontextmenü angezeigt, das z. B. das Löschen der Kachel ermöglicht (Eintrag Von „Start“ lösen).	 <i>PowerShell-Kachel im Startmenü</i>
Als Administrator ausführen	Nach einer Abfrage der Benutzerkontensteuerung, ob Sie die Aktion zulassen wollen, öffnet Windows ein neues PowerShell-Konsolenfenster mit erhöhten Rechten im Startverzeichnis <code>C:\Windows\System32</code> . Die Titelleiste der Anwendung weist Sie darauf hin, dass Sie die Anwendung als Administrator geöffnet haben. Der Titel des Fensters lautet <i>Administrator: Windows PowerShell</i> – und nicht wie üblich <i>Windows PowerShell</i> .	

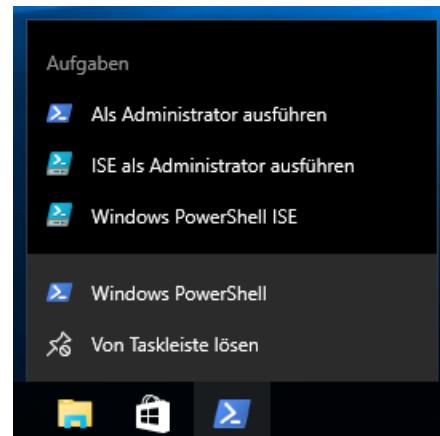
Aktion	Erläuterung
Speicherort öffnen	Durch die Wahl dieser Option öffnet Windows ein Explorerfenster in einem Unterordner des Startmenüordners des angemeldeten Benutzers ( <code>C:\Users\&lt;NAME&gt;\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Windows PowerShell</code> ; wobei <code>&lt;NAME&gt;</code> als Platzhalter für den angemeldeten Benutzer steht). Dort finden Sie die Verknüpfungen von PowerShell und PowerShell ISE (jeweils 32bit und 64bit). Die Verknüpfungen können Sie bei Bedarf über den Kontextmenüpunkt <i>Eigenschaften</i> anpassen.
An Taskleiste anheften	Durch diese Aktion wird ein PowerShell-Icon an die Taskleiste des Windows-Desktops angeheftet. Die Anwendung können Sie mit einem einfachen Linksklick auf das Icon öffnen.



Icon in der Taskleiste

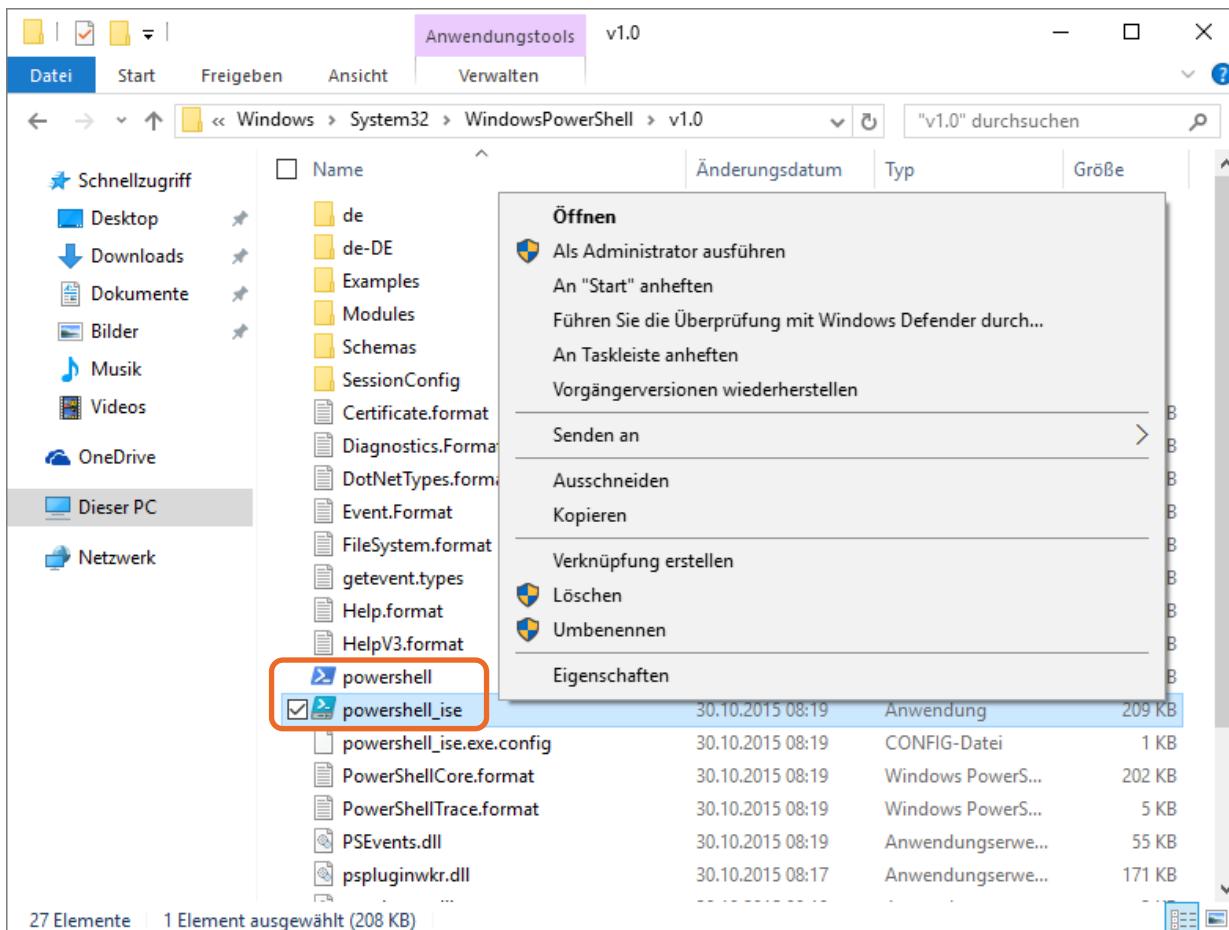
Ist die PowerShell als Icon in der Taskleiste verfügbar, können Sie mit der rechten Maustaste die benötigte Aufgabe festlegen. Sie können die PowerShell oder die PowerShell ISE standardmäßig oder mit erhöhten Rechten starten oder PowerShell wieder aus der Taskleiste entfernen.

- Alternativ öffnen Sie den Windows Explorer und navigieren Sie zum Installationsverzeichnis von PowerShell.  
Das Verzeichnis heißt `%WINDIR%\System32\WindowsPowerShell\v1.0`, wobei die Systemvariable `%WINDIR%` das Windows-Installationsverzeichnis bezeichnet. Standardmäßig ist dies das Verzeichnis `C:\Windows`.

PowerShell in der Taskleiste:  
rechte Maustaste

Lassen Sie sich von der Versionsnummer im Pfadnamen nicht verwirren. Jede PowerShell-Version ist im Verzeichnis `v1.0` zu finden. Der Pfad konnte aus Gründen der Abwärtskompatibilität ab PowerShell-Version 2.0 nicht mehr verändert werden.

- Im Installationsverzeichnis von PowerShell öffnen Sie die gewünschte PowerShell durch einen Doppelklick. Im Verzeichnis stehen Ihnen zwei Anwendungen zur Verfügung: `powershell.exe` für die PowerShell-Konsole und `powershell_ise.exe` für PowerShell ISE. Durch einen Klick mit der rechten Maustaste können Sie weitere Aktionen ausführen, z. B. die Anwendung als Kachel im Startmenü anzeigen, an die Taskleiste im Desktop anheften und das Ausführen der Anwendung mit erhöhten Rechten.



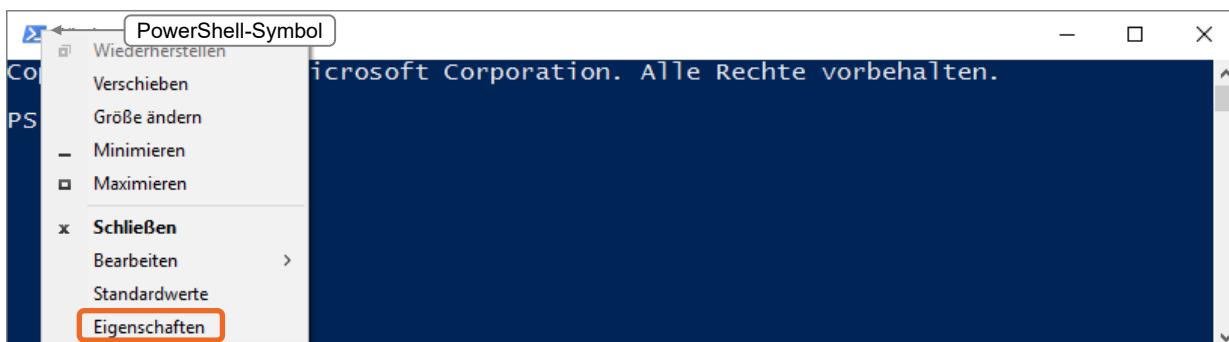
Installationsverzeichnis von PowerShell: „powershell.exe“ und „powershell\_ise.exe“ (rechter Mausklick)

## 1.5 Die PowerShell-Konsole individuell konfigurieren

### Eigenschaften öffnen

Die grundlegenden Einstellungen nehmen Sie wie folgt vor:

- Klicken Sie auf das kleine PowerShell-Symbol links oben in der Titelleiste des Konsolenfensters.
- Wählen Sie im Kontextmenü den Eintrag **Eigenschaften**, um zu den Einstellungsmöglichkeiten zu gelangen.



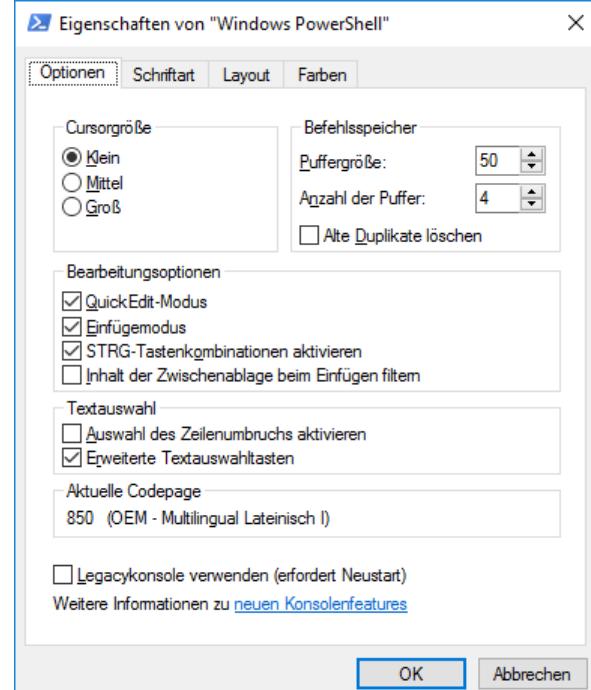
PowerShell-Konsole individuell einrichten: Eigenschaften aufrufen

## Anpassungen vornehmen

In den neuen Betriebssystemen Windows 10 und Windows Server 2016 wurde das Konsolenfenstersystem der PowerShell-Konsole modernisiert. Die Einstellmöglichkeiten verteilen sich über vier Registerkarten:

### Registerkarte Optionen

- ✓ **Cursorgröße:** Hier können Sie bestimmen, ob der blinkende Cursor ein Strich oder ein Block über die halbe bzw. ganze Zeichengröße sein soll.
- ✓ **Befehlsspeicher:** Sie legen über die Puffergröße fest, wie viele Befehle sich die Konsole merken soll. Mit den seit der DOS-Konsole unveränderten Tasten **↑** und **F7** können Sie zuletzt eingegebene Befehle aufrufen bzw. eine Liste der letzten Befehle anzeigen. Bei Aktivierung der Option *Alte Duplikate löschen* führt die Befehlsliste keine doppelten Einträge, auch wenn ein Befehl mehrmals eingegeben wurde. Die Anzahl der Puffer steuert, ob innerhalb der PowerShell geöffnete Programme auch einen Befehlsspeicher führen können. (Öffnen Sie innerhalb einer PowerShell-Sitzung eine Eingabeaufforderung (cmd.exe), so kann diese Ihre Befehle nur dann speichern, wenn die Anzahl der Puffer mindestens 2 beträgt.)
- ✓ **Bearbeitungsoptionen:** Wählen Sie den *QuickEdit-Modus*, werden einfache Textbearbeitungsoperationen deutlich einfacher, als Sie es von einem Konsolenfenster gewohnt sind. (Weiter unten im Kapitel finden Sie einen eigenen Abschnitt zum QuickEdit-Modus.)



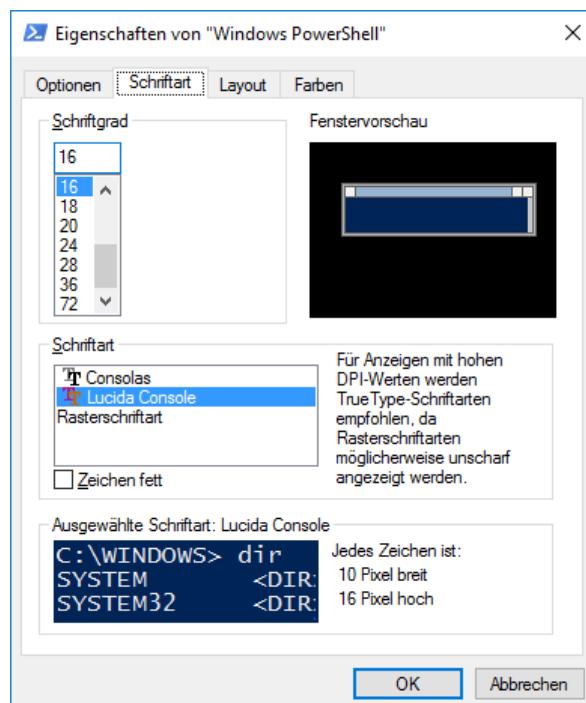
Der *Einfügemodus* sorgt für ein Einfügen von Zeichen in einen vorhandenen Text oder Befehl. Ist dieser Modus nicht aktiviert, überschreiben Eingaben bereits vorhandenen Text. Mit *STRG-Tastenkombinationen aktivieren* können Sie z. B. bekannte Tastenkombinationen zum Kopieren und Einfügen (**Strg C** und **Strg V**) in der Konsole verwenden. *Inhalt der Zwischenablage beim Einfügen filtern* sorgt dafür, dass bei Einfügen des Inhalts der Zwischenablage Tabulatoren entfernt und typografische Anführungszeichen in gerade umgewandelt werden.

- ✓ Hinzugekommen sind ebenfalls Einstellungen zur Textauswahl in der Konsole, eine Anzeige der aktuellen Zeichencodierung sowie die Möglichkeit, durch Auswahl der Option *Legacykonsole verwenden* das „alte“ Konsolenverhalten wiederherzustellen und die neuen Funktionen zu deaktivieren.

### Registerkarte Schriftart

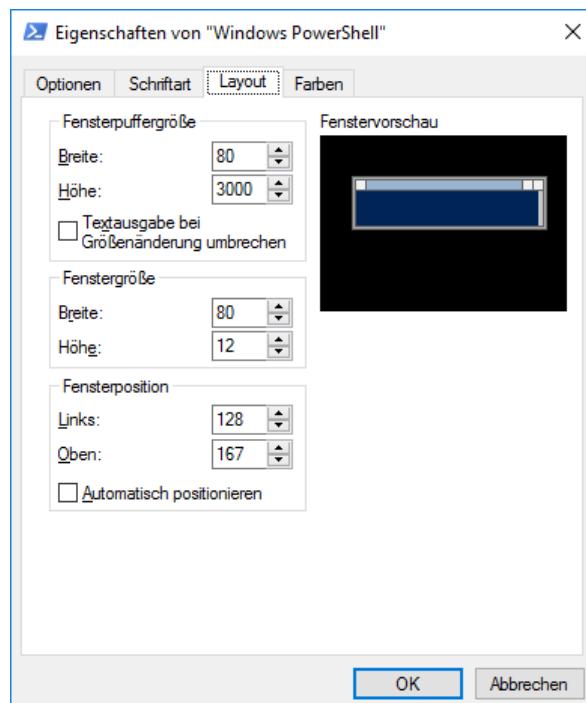
Hier wählen Sie die Schriftart und -größe für die Konsole aus. Standardmäßig ist die TrueType-Schriftart *Lucida Console* ausgewählt. Es steht mit der Schriftart *Consolas* eine weitere TrueType-Schriftart zur Verfügung. Da sich TrueType-Schriftarten stufenlos vergrößern bzw. verkleinern lassen, können Sie eine beliebige Schriftgröße festlegen. Ist der gewünschte Wert nicht in der Liste zu finden, können Sie ihn direkt im Feld *Größe* eintragen. Wenn Sie der Meinung sind, dass die Schrift fettgedruckt besser lesbar ist, aktivieren Sie die Option *Zeichen fett*.

Darüber hinaus können Sie auch die Rasterschriftart mit ihrer Auswahl fester Größen auswählen.



### Registerkarte Layout

Auf dieser Registerkarte legen Sie Fenstergröße und -position fest (Einstellungen *Fenstergröße* und *Fensterposition*). Zusätzlich bestimmen Sie über die Einstellung *Fensterpuffergröße*, wie viele Informationen die PowerShell im Speicher behält. Bei einer Breite von 120 Zeichen und einer Höhe von 3000 Zeichen erfolgt die Ausgabe in einer Breite von 120 Zeichen. Sie können die letzten 3000 Zeilen nach oben scrollen, was Ihnen einen Blick auf ältere Ausgaben ermöglicht.

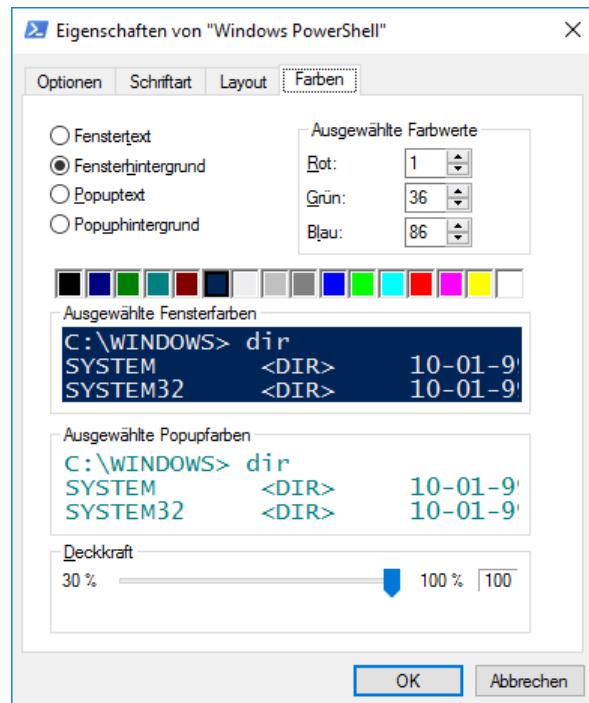


Stellen Sie die Fenstergröße auf mindestens die gleiche Breite wie die Fensterpuffergröße, damit Sie horizontal nicht scrollen müssen. Alternativ können Sie *Textausgabe bei Größenänderung umbrechen* aktivieren, so dass zu breite Textzeilen automatisch umgebrochen werden.



### Registerkarte Farben

Ihre Einstellungen bestimmen das Erscheinungsbild der Konsole. Sie können die Farbe des Fenster- und Popup-Texts sowie die dazugehörigen Hintergrundfarben festlegen. Sie wählen dabei entweder eine der sechzehn angebotenen Farben oder mischen sich selbst einen Farbton aus den Grundfarben rot, grün und blau. Neu sind Einstellungsmöglichkeiten der *Deckkraft* des Konsolenfensters (von 30–100%). Damit können Sie steuern, ob das Konsolenfenster zu einem gewissen Grad transparent erscheint und darunterliegende Fenster bzw. der Desktop im Hintergrund „durchscheint“. Allerdings geht die Einstellung auf Kosten der Lesbarkeit der Schrift im Konsolenfenster.



### QuickEdit-Modus in der Konsole

Mit dem QuickEdit-Modus können Sie in Konsolenfenstern Text in die Zwischenablage kopieren und am Prompt einfügen:

- ▶ Bewegen Sie den Mauszeiger an den Anfang des Textes, den Sie markieren wollen.
- ▶ Ziehen Sie die Maus bei gedrückter linker Maustaste bis an das Ende des zu markierenden Textes. Damit erzeugen Sie eine Markierung, die farblich invers dargestellt wird.
- ▶ Drücken Sie **←**, um den Textbereich in die Zwischenablage zu kopieren.  
Alternative: Klick mit der rechten Maustaste
- ▶ Möchten Sie stattdessen den Vorgang abbrechen, drücken Sie **Esc**.

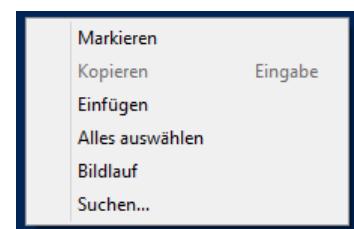
Der Text steht in der Zwischenablage auch in anderen Programmen zur Verfügung und kann dort eingefügt werden (z. B. mithilfe von **Strg** **V**). Das Einfügen des Textes in der PowerShell geht einfacher:

- ▶ Klicken Sie zum Einfügen des Textes aus der Zwischenablage mit der rechten Maustaste. Die Position der Maus spielt dabei keine Rolle, solange sie sich über dem Inhaltsbereichs des Konsolenfensters befindet. Der Text wird automatisch an der Cursorposition eingefügt.

Der QuickEdit-Modus ist standardmäßig in der PowerShell aktiviert. Sollte dies nicht der Fall sein, merken Sie es durch das Erscheinen eines Kontextmenüs bei rechtem Mausklick. Wenn Sie manuell den QuickEdit-Modus ein- oder ausschalten wollen, gehen Sie vor, wie es weiter oben unter den Anpassungen auf der Registerkarte beschrieben wurde.

```
PS C:\Users\Andreas> Get-Date
Sonntag, 23. Mai 2021 23:21:09
```

Markierung im QuickEdit-Modus ...



Kontextmenü im Standardmodus

## 1.6 Bekannte Befehle – leichter Einstieg

Microsoft hat die PowerShell so konzeptioniert, dass Ihnen die ersten Schritte in der PowerShell leichtfallen. In vielen Fällen sind bekannte Befehle aus DOS, Windows oder Linux als alternative Befehlsnamen vordefiniert, um Ihnen die ersten Schritte in der PowerShell zu vereinfachen. Die Arbeit mit der PowerShell ist ein mehrstufiges Modell: Für den Anfang lernen Sie in diesem Kapitel den Umgang mit Befehlen, ohne einen genuinen PowerShell-Befehl kennenzulernen. Das folgt dann in den nächsten Kapiteln als nächste Stufe des Modells.

Sofern nicht anders verzeichnet, arbeiten Sie in den nächsten Kapiteln mit der **PowerShell-Konsole**. Testen Sie, ob Sie die PowerShell-Konsole oder den Konsolenbereich in der PowerShell ISE verwenden möchten. Inhaltlich besteht kein Unterschied. Bearbeiten Sie zur Entscheidungsfindung die Beispiele des Kapitels in beiden Anwendungen.

### Bekannte Befehle aus der Windows-Eingabeaufforderung verwenden

Nachdem Sie Ihre Arbeitsumgebung gemäß Ihren Vorstellungen angepasst haben, kann die inhaltliche Arbeit im Konsolenfenster der PowerShell beginnen. In der PowerShell funktionieren etliche bekannte Befehle aus der Windows-Eingabeaufforderung oder der Unix-Welt wie gewohnt.

Bekannte Befehle, die auch in der PowerShell funktionieren, sind u. a. folgende:

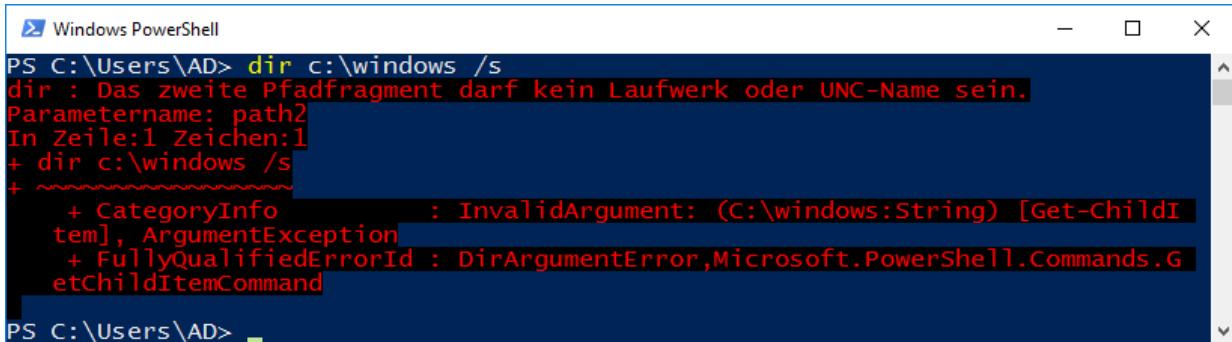
<code>cd</code> oder <code>chdir</code>	Verzeichnis wechseln	<code>kill</code>	Prozess beenden (Unix)
<code>cls</code>	Bildschirminhalt löschen	<code>md</code>	Verzeichnis erstellen
<code>copy</code>	Kopieren z. B. einer Datei	<code>move</code>	Verschieben z. B. einer Datei
<code>date</code>	Datum anzeigen	<code>rd</code>	Verzeichnis löschen
<code>del</code> oder <code>erase</code>	Löschen z. B. einer Datei	<code>sleep</code>	Angegebene Sekundenzahl warten (Unix)
<code>dir</code> oder <code>ls</code>	Verzeichnisinhalt auflisten	<code>type</code> oder <code>cat</code>	Dateiinhalt anzeigen
<code>echo</code>	Meldung anzeigen		

Allerdings werden Sie feststellen, dass Parameter, die Sie vielleicht noch von diesen alten Befehlen kennen, zu einer Fehlermeldung führen.

- ▶ Geben Sie in der PowerShell-Konsole und in der Windows-Eingabeaufforderung jeweils den folgenden Befehl ein: `dir c:\windows ↵`.  
Sollte Ihr Betriebssystem in einem anderen Pfad zu finden sein, müssen Sie den Pfad entsprechend anpassen.  
Mit dem Befehl wird in beiden Anwendungen der Inhalt des Verzeichnisses `C:\Windows` aufgelistet. Die Formatierung der Ausgabe in der PowerShell unterscheidet sich etwas von der Ausgabe der Windows-Eingabeaufforderung.
- ▶ Geben Sie anschließend in der PowerShell-Konsole und in einer Windows-Eingabeaufforderung jeweils den folgenden Befehl ein: `dir c:\windows /s ↵`.  
Die Windows-Eingabeaufforderung zeigt das erwartete Verhalten und listet den Inhalt des Verzeichnisses inklusive aller Unterverzeichnisse auf. Die PowerShell jedoch reagiert auf diesen Befehl mit einer Fehlermeldung.

Wenn Sie die Ausführung eines Befehls abbrechen wollen, drücken Sie `Strg C`. Dies funktioniert sowohl in der PowerShell als auch in der Windows-Eingabeaufforderung.





The screenshot shows a Windows PowerShell window with the following command and its error output:

```
PS C:\Users\AD> dir c:\windows /s
dir : Das zweite Pfadfragment darf kein Laufwerk oder UNC-Name sein.
Parametername: path2
In Zeile:1 Zeichen:1
+ dir c:\windows /s
+ ~~~~~~
+ CategoryInfo          : InvalidArgument: (C:\windows:String) [Get-ChildItem], ArgumentException
+ FullyQualifiedErrorId : DirArgumentError,Microsoft.PowerShell.Commands.GetChildItemCommand
```

Fehlermeldung: Der vom Befehl `dir` bekannte Parameter `/s` ist nicht bekannt.

Die Fehlermeldung in der PowerShell weist darauf hin, dass der eingegebene Parameter unbekannt ist. Der Grund dafür ist, dass hier nicht der Befehl `dir` ausgeführt wird. Vielmehr sind die Befehle aus der obigen Tabelle alternative Kurznamen, die PowerShell-Befehle ausführen.



Bekannte Befehle anderer Umgebungen wie etliche genuine DOS-Befehle funktionieren auch in der PowerShell. Gehen Sie allerdings davon aus, dass Sie die Parameter der Befehle nicht einsetzen können. Die eigenen PowerShell-Befehle, die im Hintergrund ausgeführt werden, haben andere Parameter und kennen die Parameter der „alten“ Befehle nicht. (Ausführliche Informationen zu den alternativen Befehlen, den sogenannten Aliasen, erhalten Sie in Kapitel 5.)

### PowerShell-Fehlermeldungen

PowerShell-Fehlermeldungen werden in einer anderen Schrift- und Hintergrundfarbe angezeigt, in den Standardeinstellungen in roter Schrift auf schwarzem Grund. Lesen Sie die Fehlermeldungen immer aufmerksam. Die Meldungen sind in der Regel brauchbar, möglicherweise können Sie mit ihrer Hilfe den Fehler bereits beheben.

### In der PowerShell mit der Windows-Eingabeaufforderung arbeiten

Wenn Sie mit der PowerShell arbeiten, ist eine Verwendung der klassischen Konsole nicht mehr nötig, da die vergleichsweise wenigen Funktionen der Konsole (fast) komplett in der PowerShell zur Verfügung stehen. Benötigen Sie dennoch die klassische Konsole, während Sie mit der PowerShell arbeiten, müssen Sie keine neue Anwendung starten. Es stehen mehrere Möglichkeiten zur Verfügung, die klassische Konsole innerhalb einer PowerShell-Sitzung zur Verfügung zu stellen.

Wollen Sie mehrere Befehle in der Windows-Eingabeaufforderung eingeben, gehen Sie wie folgt vor:

- Geben Sie in der PowerShell-Konsole den folgenden Befehl ein: `cmd ↵`.  
Der Prompt ändert sich, er verliert das führende PS als Kennzeichnung, dass Sie in der PowerShell arbeiten. Nun können Sie – wie gewohnt – die klassische Konsole, die sog. „Eingabeaufforderung“, verwenden.
- Zur Rückkehr in die PowerShell geben Sie folgenden Befehl ein: `exit ↵`.

Möchten Sie nur einen einzelnen Befehl in der klassischen Konsole ausführen und nach Ausführung des Befehls wieder zur PowerShell zurückkehren, verwenden Sie `cmd` zusammen mit dem Parameter `/c`. Im folgenden Beispiel wird der in der PowerShell unbekannte Befehl `ver` zur Anzeige der Windows-Version verwendet:

```
cmd /c ver
```



Geben Sie in der PowerShell den Befehl `cmd /?` ein, erhalten Sie einen Hilfetext, der Ihnen alle möglichen Parameter für die Verwendung mit dem Befehl `cmd` anzeigt.

## Externe Befehle in der PowerShell

Aus der PowerShell heraus können Sie externe Kommandos aufrufen. Externe Kommandos sind Befehle, die nichts mit der PowerShell zu tun haben, sondern z. B. vom Betriebssystem zur Verfügung gestellt werden. Beim Aufruf externer Kommandos aus der PowerShell gibt es keine Unterschiede zur Windows-Eingabeaufforderung. Alle gewohnten Parameter können verwendet werden. Hier einige Beispiele:

Externer Befehl	Erläuterung
ipconfig /all	Zeigt detaillierte IP-Konfigurationsinformationen an
netsh	Wechselt zum netsh-Prompt der Netshell zum Konfigurieren von Netzwerk-einstellungen
notepad test.txt	Öffnet die Datei <i>test.txt</i> im Windows-Editor, bzw. der Editor fragt, ob diese Datei erstellt werden soll
nslookup /?	Zeigt einen kurzen Hilfetext zur Verwendung des Befehls nslookup
ping www.heise.de	Pingt die Internetadresse <i>www.heise.de</i> an
regedit	Öffnet den Registrierungs-Editor
tracert www.herd़t.com	Verfolgt die Route zwischen eigenem Host und dem Zielhost <i>www.herd़t.com</i>

Beim Aufruf von Windows-Anwendungen wie *notepad* oder *regedit* ist die Konsole gleich wieder einsatzbereit, da sich die aufgerufene Anwendung in einem eigenen Fenster öffnet. Bei textbasierten Konsolenbefehlen ist die Konsole blockiert, bis der Befehl abgearbeitet ist.

## Einfaches Rechnen in der PowerShell

Die PowerShell unterstützt die mathematischen Grundrechenarten, Hexadezimalwerte und in der IT gebräuchliche Maßeinheiten wie kB (Kilobyte) und MB (Megabyte).

Hexadezimalwerte werden mit führendem *0x* notiert, zulässige Maßeinheiten sind Kilo-, Mega-, Giga-, Tera- und Petabyte, zu schreiben als *kb*, *mb*, *gb*, *tb* und *pb*. Die Maßeinheiten müssen direkt – und ohne Leerzeichen – an den Zahlwert angehängt werden. Groß- und Kleinschreibung werden nicht unterschieden.

In der folgenden Tabelle sehen Sie eine Übersicht über die Operatoren anhand einiger Beispiele. Ob Sie vor und/oder nach dem Operator ein Leerzeichen  tippen, bleibt Ihnen überlassen.

Operator	Aktion	Beispieleingabe	Ergebnis	Erläuterung
+	Addition	70 + 35.74 "Herd़t" + " Verlag"	105.74 Herd़t Verlag	Dezimaltrennzeichen ist das Zeichen <input type="text"/> . Texte werden verknüpft.
-	Subtraktion	0x5F – 23 60gb – 45gb	72 16106127360	Ergebnis ist immer ein Dezimalwert. Ergebnis wird als Zahlwert ohne Angabe der Einheit angezeigt.
*	Produkt	20 * 3.554 "aBc" * 3	71.08 aBcaBcaBc	- Text wird dreimal wiederholt.
/	Division	1TB / 4mb 0xFF / 0xa	262144 25.5	Groß- und Kleinschreibung sind irrelevant. Ergebnis ist Dezimalwert.
%	Modulo	7 % 6.8 7 % 2	0.2 1	Rest der Ganzzahldivision 7/2 = 3, Rest 1
..	Bereich	1..6 5..3	1 2 3 4 5 6 5 4 3	Ausgabe erfolgt in einzelnen Zeilen. Nur Zahlen sind zugelassen.



Die PowerShell gibt bei der Eingabe von "aBc" \* 3 als Ergebnis *aBcaBcaBc* zurück. PowerShell erkennt im ersten Operanden eine Zeichenkette und reiht sie mehrfach aneinander. Wie häufig dies geschieht, legt der zweite Operand fest. Geben Sie den Befehl aber in anderer Reihenfolge ein – 3 \* "aBc", gibt die PowerShell die folgende Fehlermeldung aus: *Der Wert "aBc" kann nicht in den Typ "System.Int32" konvertiert werden. Fehler: "Die Eingabezeichenfolge hat das falsche Format."* PowerShell versucht, den zweiten Operanden in das Datenformat des ersten zu konvertieren. Dies funktioniert bei Hexadezimal- und Dezimalwerten, aber nicht beim Versuch, eine Zeichenkette in einen Zahlenwert umzuwandeln.

## 1.7 Konsole oder ISE einsetzen?

Neben den erwähnten Einsatzgebieten für die PowerShell-Konsole und PowerShell ISE spielt Ihre eigene Arbeitsweise eine Rolle, sich für eine der beiden Anwendungen zu entscheiden.

Setzen Sie die PowerShell-Konsole ein, wenn Sie ...

- ✓ eine schnell startende Konsole für interaktive Eingaben benötigen
- ✓ den Komfort einer Windows-Anwendung nicht benötigen
- ✓ Befehle verwenden, die eine interaktive Eingabe des Benutzers erwarten (Diese Art von Befehlen funktioniert in der PowerShell ISE nicht. Die betreffenden Kommandos können Sie anzeigen, wenn Sie in der PowerShell ISE den Befehl \$psUnsupportedConsoleApplications eingeben.)

Setzen Sie die PowerShell ISE ein, wenn Sie ...

- ✓ automatische Programmhilfe bei der Eingabe (IntelliSense) verwenden wollen.
- ✓ sich automatisch Syntaxfehler anzeigen lassen wollen.
- ✓ den Komfort einer Windows-Anwendung einem Konsolenfenster vorziehen.
- ✓ in erster Linie mit der PowerShell programmieren und längeren, fortgeschrittenen Code produzieren.
- ✓ bemerken, dass die Eingabe sehr langer Zeilen im Konsolenfenster zu unübersichtlich wird.
- ✓ die Unterstützung von Unicode-Zeichensätzen (z. B. für Sprachen mit anderen Schriftzeichen) benötigen, was von der Konsole nicht unterstützt wird.

Wenn Sie sich nicht (sofort) entscheiden mögen, üben Sie einfach parallel in beiden Anwendungen.

## 1.8 Übung

### Umgebung einrichten, erste Orientierung in der PowerShell

Übungsdatei: –

Ergebnisdatei: –

1. Binden Sie für einen schnellen Start der PowerShell-Konsole eine PowerShell-Kachel in das Startmenü von Windows 10 ein. (Sollten Sie ausschließlich mit Windows Server 2016 oder 2019 arbeiten, entfernen Sie erst die standardmäßig bereits vorhandene Kachel, um sie dann erneut einzubinden.)
2. Binden Sie die PowerShell in die Taskleiste von Windows 10 ein und richten das Konsolenfenster gemäß Ihren individuellen Bedürfnissen ein.
3. Experimentieren Sie mit bereits bekannten Befehlen sowie externen Befehlen aus der Windows-Umgebung. Schreiben Sie einen der Befehle absichtlich falsch, um eine PowerShell-Fehlermeldung zu erhalten. Machen Sie sich mit der PowerShell vertraut.
4. Testen Sie sowohl die PowerShell-Konsole als auch PowerShell ISE und entscheiden Sie, welche Anwendung Ihrem Arbeitsstil besser entspricht. Nutzen Sie diese Anwendung im weiteren Verlauf des Buches als Hauptanwendung.

## 2 PowerShell-Cmdlets



**Beispieldatei:** 2\_kompletter code.txt (Ordner Kapitel 02)

### 2.1 Grundlagen zu PowerShell-Cmdlets

#### Cmdlets

Im Zentrum der PowerShell stehen kleine Funktionseinheiten, die **Cmdlet** genannt (und „commandlet“ gesprochen) werden.

Jedes Cmdlet ist als integrierte Komplettlösung für eine spezifische Aufgabe zu verstehen. Sie benötigen eine Liste der laufenden Prozesse? Dafür gibt es ein Cmdlet. Die Ergebnisse sollen gefiltert und sortiert werden? Auch für diese Aufgaben stehen entsprechende Cmdlets zur Verfügung.

Cmdlets sind grundsätzlich in Modulen und SnapIns organisiert, die beim Start der PowerShell geladen werden. Ein Modul bzw. ein SnapIn bietet PowerShell-Befehlserweiterungen, meist zu einem speziellen Thema. Neben den Standardmodulen mit den von Beginn an verfügbaren Befehlen können Module und SnapIns automatisch oder manuell nachgeladen werden, z. B. für einen Exchange Server oder Active Directory auf einem Domänencontroller.

Module stellen die modernere Umsetzung dar und haben SnapIns mittlerweile größtenteils abgelöst.

#### Syntax der Cmdlets

Verb-Substantiv [-Parameter [Wert]]

- ✓ Alle Cmdlets sind nach dem Muster Verb-Substantiv aufgebaut. Der erste Namensteil besteht aus einem Verb, das die Tätigkeit des Cmdlets verrät, wie z. B. Get, Set oder Rename. Im zweiten Namensteil steht ein Substantiv, das über das Tätigkeitsfeld des Cmdlets Auskunft gibt (z.B.: Service, Object oder EventLog). Daraus ergeben sich Cmdlets wie Get-Command zur Ausgabe von Befehlslisten.
- ✓ Cmdlets werden in englischer Sprache notiert und die jeweiligen Substantive **immer im Singular** verwendet. Zum Auslesen laufender Dienste verwenden Sie z. B. Get-Service (**nicht** Get-Services) oder für eine Liste laufender Prozesse Get-Process (**nicht** Get-Processe).
- ✓ Groß- und Kleinschreibung wird nicht unterschieden. Zur besseren Lesbarkeit werden häufig Verb und Substantiv, Parameter und Werte – wie im vorliegenden Buch – mit einem Großbuchstaben begonnen. Darüber hinaus beginnen alle einzelnen Wörter in zusammengesetzten Begriffen mit einem Großbuchstaben wie z. B. EventLog, ControlPanelItem oder ADForestMode.
- ✓ Cmdlets besitzen Parameter (vgl. Abschnitt 2.2), die die Funktion des Cmdlets genauer steuern. Häufig ist die Verwendung von Parametern optional.
- ✓ PowerShell hilft Ihnen beim Tippen der Cmdlets. In der Konsole können Sie begonnene Befehle und Parameter mit der Taste automatisch vervollständigen. In PowerShell ISE steht Ihnen das sogenannte IntelliSense zur Verfügung, das Ihnen während des Tippens mit Menüs zur Vervollständigung zur Seite steht.
- ✓ Mit der Konsole der PowerShell 5 weist die aktuelle Befehlszeile Farbcodierungen auf. Beispielsweise werden Cmdlets gelb dargestellt, Parameter grau und Werte weiß. Die verwendeten Werte können Sie sich mit dem Cmdlet Get-PSReadlineOption anzeigen lassen. Auch die PowerShell ISE kennt entsprechende Farbcodierungen.

**Get-Help -Name Get-Command**

Farbcodierung in der PowerShell-Konsole

Grundlegende Englischkenntnisse erleichtern die Arbeit mit der PowerShell. Wenn Sie die häufigsten Begriffe aus Ihrer deutschen Windows-Welt auch auf Englisch kennen, finden Sie schneller passende Cmdlets oder erkennen, welche Aufgabe vorliegende Cmdlets erfüllen.



Bei den folgenden Definitionen finden Sie praxisnahe Anwendungsbeispiele. Die in den Beispielen verwendeten Cmdlets werden im weiteren Verlauf des Kapitels erklärt.

## 2.2 Parameter

Für jede Aufgabe existiert das passende Cmdlet, das die Hauptrichtung bestimmt. Über Parameter ermöglichen Sie darüber hinaus eine fein abgestimmte Steuerung des Cmdlets. Ein Parameter ist an dem führenden Zeichen `-` zu erkennen. Vor diesem führenden Zeichen ist zusätzlich immer ein Leerzeichen  zu verwenden.

Es gibt drei Typen von Parametern, die für die Cmdlets definiert werden können:

- ✓ benannte Parameter,
- ✓ Switch-Parameter und
- ✓ Positionsparameter.

### Benannte Parameter

Benannte Parameter sind Parameter, die Sie mit ihrem Namen angeben müssen. Nach einem Leerzeichen folgt dann zwingend der Wert für den Parameter. Die allgemeine Syntax sieht wie folgt aus:

```
<Cmdlet> -Parameter Wert
```

#### Beispiel

```
Get-Command -Noun service ↵
```

Es werden alle Befehle ausgegeben, die das Substantiv `service` verwenden, also alle Befehle rund um *Dienste*.

### Switch-Parameter

Ein Switch-Parameter ist ein Schalter, der ohne Wert auskommt. Wird der Switch-Parameter angegeben, ist er aktiv. Wird er nicht angegeben, ist er inaktiv. Die allgemeine Syntax ist einfacher als bei benannten Parametern:

```
<Cmdlet> -Parameter
```

#### Beispiel

```
Get-ChildItem -Recurse ↵
```

Mithilfe des Cmdlets `Get-ChildItem` wird der Inhalt des aktuellen Verzeichnisses angezeigt. Der Switch-Parameter `-Recurse` sorgt dafür, dass ebenfalls der Inhalt aller Unterverzeichnisse angezeigt wird.



Besitzen Sie nicht die Berechtigung zum Zugriff auf ein Unterverzeichnis, wird eine entsprechende Fehlermeldung angezeigt.

### Positionsparameter

Ein Positionsparameter ist ein Parameter, dessen Name angegeben werden kann, aber nicht muss. Er stellt somit eine Sonderform dar, die es Ihnen ermöglicht, nur den Wert des Parameters ohne dessen Namen angeben zu können. Manche Cmdlets erlauben mehr als einen Positionsparameter. Es können auch mehrere Werte ohne Angabe des Parameters nacheinander eingegeben werden. Die Zuordnung erfolgt automatisch über die Position des Wertes.

Wenn Sie die Zuordnung aufgrund der Position nicht beachten, kommt es leicht zu Fehlern. Die eingegebenen Cmdlets sind zudem ohne Angabe des Parameters schwerer zu lesen, da nicht auf den ersten Blick klar wird, welcher Wert zu welchem Parameter gehört.



Positionsparameter sind als schnelle Lösung für eine direkte Eingabe erfahrener Anwender gedacht. Lernen Sie den Umgang mit der PowerShell, sollten Sie die Befehle immer komplett eingeben. In allen anderen Fällen, z. B. in Skripten oder bei Arbeiten im Team, empfiehlt sich die komplette Schreibweise.

Positionsparameter sind wie folgt zu notieren:

```
<Cmdlet> [-Parameter] Wert
```

### Beispiele

```
Get-Process [-Name] powershell ↵
```

Das Cmdlet Get-Process ruft Informationen zu laufenden Prozessen ab, hier zum Prozess *powershell*, der auf Ihrem Computer ausgeführt wird. Der Positionsparameter **-Name** ist optional, es kann auch einzig der Wert *powershell* angegeben werden.

```
Copy-Item [-Path] c:\alt [-Destination] c:\neu ↵
```

Copy-Item kopiert ein Element, hier das Verzeichnis *C:\alt* nach *C:\neu*. Das Cmdlet besitzt zwei Positionsparameter **-Path** und **-Destination** für die Angabe der Quelle und des Ziels. Eine verkürzte Schreibweise **Copy-Item c:\alt c:\neu** ist möglich.

Im Folgenden wird darauf verzichtet, bei jedem Beispiel explizit darauf hinzuweisen, dass die Befehle mit der Taste ↵ abgeschlossen werden. Erst nach Betätigung dieser Taste erfolgt die Verarbeitung durch die PowerShell.

### Darstellung von Parametertypen bei der Vorstellung neuer Parameter

Um Parameter korrekt verwenden zu können, müssen Sie wissen, um welchen Parametertyp es sich handelt. Alle Tabellen im Buch, die eine Übersicht über Cmdlets und ihre Parameter liefern, enthalten die in der folgenden Tabelle beschriebenen Typ-Angaben:

Typ	Beschreibung
N	<b>Benannter Parameter:</b> Name des Parameters <b>muss</b> stets angegeben werden.
P(x)	<b>Positionsparameter:</b> Name des Parameters <b>kann</b> angegeben werden. Wird der Parametername nicht angegeben, bezeichnet die Zahl in Klammern die Position, an der der Wert des Parameters stehen muss.
S	<b>Switch-Parameter:</b> Parameter, der als Ein-/Aus-Schalter fungiert und ohne Wertangabe auskommt

Ist die Angabe eines Parameters zwingend erforderlich, wird der Parametertyp in der jeweiligen Übersicht fett markiert.

## 2.3 Allgemeine Parameter

### Was ein allgemeiner Parameter ist

Zusätzlich zu den drei vorgestellten Parametertypen gibt es einen zusätzlichen Typ, der allen Cmdlets zur Verfügung steht: allgemeine Parameter (**common parameters**). Allgemeine Parameter sind bereits durch die PowerShell implementiert und damit unabhängig von einzelnen Cmdlets.

Allgemeine Parameter sind für grundlegende Aufgaben vorgesehen. Die am häufigsten verwendeten allgemeinen Parameter sind folgende:

Allgemeine Parameter		
Parameter	Typ	Beschreibung
-Verbose	S	Zeigt sehr ausführliche Informationen zu dem Vorgang an, der mit dem Befehl ausgeführt wird
-Debug	S	Zeigt zusätzlich Informationen für Programmierer, wenn Warnungen und Fehler bei dem ausgeführten Vorgang auftreten
-WarningAction	N	Bestimmt das Verhalten des Cmdlets, falls eine Warnung auftritt. Gültige Werte sind: ✓ <i>SilentlyContinue</i> : Warnung unterdrücken, Ausführung fortsetzen ✓ <i>Continue</i> : Warnung anzeigen, Ausführung fortsetzen ✓ <i>Inquire</i> : Warnung anzeigen, weitere Ausführung nachfragen ✓ <i>Stop</i> : Warnung anzeigen, Ausführung beenden
-WarningVariable	N	Speichert auftretende Warnmeldungen in der angegebenen Variablen
-ErrorAction	N	Bestimmt das Verhalten des Cmdlets, falls ein Fehler auftritt. Gültige Werte sind: ✓ <i>SilentlyContinue</i> : Warnung unterdrücken, Ausführung fortsetzen ✓ <i>Continue</i> : Warnung anzeigen, Ausführung fortsetzen ✓ <i>Inquire</i> : Warnung anzeigen, weitere Ausführung nachfragen ✓ <i>Stop</i> : Warnung anzeigen, Ausführung beenden
-ErrorVariable	N	Speichert auftretende Fehlermeldungen in der angegebenen Variablen
-OutVariable	N	Zeigt die Ausgabeobjekte des Befehls an und speichert sie in der angegebenen Variablen

Ein häufig verwendetes Parameter ist `-ErrorAction`, der das Verhalten bei einem auftretenden Fehler bei der Ausführung des betreffenden Cmdlets steuert.

- ▶ Um den Inhalt des Verzeichnisses `C:\Users\Public\Documents` inklusive aller Unterverzeichnisse anzuzeigen, geben Sie folgenden Befehl ein:

```
Get-ChildItem -Path C:\Users\Public\Documents -Force -Recurse
```

- ✓ Get-ChildItem ist ein Cmdlet, das den bekannten Befehlen `dir` bzw. `ls` entspricht. Zum Auflisten von Unterverzeichnissen wird der Switch-Parameter `-Recurse` angegeben. Der Switch-Parameter `-Force` erzwingt die Anzeige von versteckten oder Systemdateien.
- ✓ Die oben genannten Unterverzeichnisse sind versteckte Systemobjekte, auf die Sie wahrscheinlich keinen Zugriff haben. Sie erhalten Fehlermeldungen wie diese:

```
Get-ChildItem : Der Zugriff auf den Pfad "C:\Users\Public\Documents\Eigene Bilder" wurde verweigert.
In Zeile:1 Zeichen:1
+ Get-ChildItem -Path C:\Users\Public\Documents -Force -Recurse
+ ~~~~~
  + CategoryInfo          : PermissionDenied: (C:\Users\Public\Documents\Eigene Bilder:String) [Get-ChildItem], UnauthorizedAccessException
  + FullyQualifiedErrorId : DirUnauthorizedAccessError,Microsoft.PowerShell.Commands.GetChildItemCommand
```

Fehlermeldung bei Ausführung eines Cmdlets

- ▶ Erweitern Sie nun die eingegebene Zeile um den allgemeinen Parameter `-ErrorAction` mit dem Wert `SilentlyContinue`:

```
Get-ChildItem -Path C:\Users\Public\Documents -Force -Recurse
              -ErrorAction SilentlyContinue
```

Die Angabe des allgemeinen Parameters `-ErrorAction` bewirkt, dass die Ausgabe aller Fehlermeldungen unterdrückt wird. Die Ausgabe wird lesbarer, eine mögliche Verwirrung wegen auftretender Fehler entsteht nicht.

Eine verkürzte Schreibweise für `-ErrorAction SilentlyContinue`, die Sie z. B. häufiger in Beispielen im Internet finden, lautet `-ea 0`. Dies erspart zwar Tipparbeit, ist allerdings kaum nachvollziehbar.

## Risikominderungsparameter

Zu den allgemeinen Parametern gehören auch die beiden sogenannten Risikominderungsparameter `-WhatIf` und `-Confirm`. Beide Parameter können nur bei Cmdlets eingesetzt werden, die Änderungen am System durchführen.

Alle Get-Cmdlets (siehe folgender Abschnitt) lesen nur Informationen aus und nehmen keine Veränderungen am System vor.

Risikominderungsparameter		
Parameter	Typ	Beschreibung
<code>-WhatIf</code>	S	<p>Die Anweisung wird nicht ausgeführt. Eine Meldung beschreibt die Auswirkungen, wenn das Cmdlet wirklich ausgeführt wird.</p> <p>Der Parameter wird eingesetzt, wenn Sie nicht sicher sind, welche Auswirkungen die Ausführung eines Cmdlets hat, das Änderungen am System vornimmt.</p> <p>Beispiel:</p> <pre>Stop-Process -Name powershell -WhatIf</pre> <p>Das Cmdlet beendet ohne den Parameter <code>-WhatIf</code> alle laufenden PowerShell-Prozesse, schließt also alle offenen PowerShell-Sitzungen.</p> <p>Mit dem Parameter <code>-WhatIf</code> wird das Cmdlet nicht ausgeführt, sondern folgende Meldung angezeigt:</p> <pre>WhatIf: Ausführen des Vorgangs "Stop-Process" für das Ziel "powershell (&lt;Prozess-ID&gt;)".</pre>

Risikominderungsparameter		
Parameter	Typ	Beschreibung
-Confirm	S	<p>Fordert Sie vor der Ausführung jeder Aktion zur Bestätigung auf</p> <p>Beispiel:</p> <pre>Stop-Process -Name powershell -Confirm</pre> <p>Sie erhalten folgende Ausgabe:</p> <pre>Bestätigung Möchten Sie diese Aktion wirklich ausführen? Ausführen des Vorgangs "Stop-Process" für das Ziel "powershell (&lt;Prozess-ID&gt;)".</pre> <p>[J] Ja [A] Ja, alle [N] Nein [K] Nein, keine  [H] Anhalten [?] Hilfe (Standard ist "J"):</p> <p>Laufen im Beispiel mehrere Instanzen der PowerShell, erhalten Sie für jeden einzelnen Prozess eine Nachfrage.</p>

## 2.4 Get-Cmdlets für den Einstieg

### Get-Command: Befehle finden

Mit dem Cmdlet Get-Command können Sie zum einen nach dem geeigneten Befehl für eine bestimmte Aufgabe suchen, zum anderen auch wertvolle Information zur Verwendung und Struktur von Befehlen erhalten. Standardmäßig werden Cmdlets, Funktionen, Workflows und Aliases angezeigt.

- ▶ Geben Sie Get-Command ein und schließen Sie die Eingabe mit der -Taste ab.  
Das Cmdlet Get-Command liefert eine sehr lange Liste mit allen vordefinierten Cmdlets und Funktionen. Funktionen sind unter einem eigenen Namen abgespeicherte Befehlsfolgen, die Sie wie einfache Befehle verwenden können. Funktionen werden in einem späteren Kapitel ausführlich besprochen.  
Die Liste ist nach Kategorien und innerhalb der Kategorien nach Namen sortiert. Die Ausgabe ist automatisch als Tabelle formatiert, die die Spalten  *CommandType, Name und ModuleName* beinhaltet.

Als Ergebnis erhalten Sie sehr viele Informationen. Um die Anzahl der Informationen zu verringern, können Sie zusätzliche Parameter eingeben, die steuern, wie gefiltert wird:

Get-Command		
Parameter	Typ	Beschreibung
-Name	P (1)	Filtermöglichkeit für angegebene Namen bzw. Teile des gesuchten Namens
-CommandType	N	Begrenzt die Suche auf angegebene Befehlstypen. Mögliche Werte sind: <i>Alias, Function, Filter, Cmdlet, ExternalScript, Application, Script, Workflow, All</i> .
-Module	N	Schränkt die Ausgabe auf angegebene Modulnamen ein
-Noun	N	Sucht nur Befehle (Standard: Cmdlets, Funktionen, Workflows und Aliases), die im Substantiv des Befehls der eingegebenen Zeichenfolge entsprechen
-ParameterName	N	Sucht nach Befehlen, die den eingegebenen Parameter unterstützen
-Verb	N	Sucht nur Befehle, die im Verb des Befehls der eingegebenen Zeichenfolge entsprechen

Get-Command		
Parameter	Typ	Beschreibung
-Syntax	S	Zeigt die Syntax des eingegebenen Befehls inklusive aller möglichen Parameter an
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help -Name Get-Command -Full

In der nachfolgenden Tabelle finden Sie einige Beispiele, die Ihnen die Verwendung des Cmdlets Get-Command veranschaulichen soll:

Sie möchten ...	
alle PowerShell-Befehle auflisten	Get-Command
nur Cmdlets auflisten	Get-Command - CommandType cmdlet
alle Befehle finden, die mit dem Buchstaben g beginnen	Get-Command -Name g*
alle Befehle, die das Verb <i>remove</i> verwenden	Get-Command -Verb remove
alle Befehle, mit der Zeichenfolge <i>event</i> an beliebiger Stelle im Substantiv	Get-Command -Noun *event*
alle Befehle, die das Verb <i>get</i> verwenden und deren Substantiv mit p beginnt	Get-Command -Verb get -Noun p*
die Syntax des Cmdlets Get-Service anzeigen	Get-Command Get-Service -Syntax
wissen, welche Befehle den Parameter <i>List</i> verwenden	Get-Command -Name * -ParameterName List

Wenn Sie den oben angezeigten Befehl Get-Command - CommandType cmdlet verwenden, wissen Sie vielleicht nicht genau, welche gültigen Werte der Parameter - CommandType hat. Dies können Sie über die PowerShell-Hilfe erfahren, wie Sie im nächsten Abschnitt lesen werden. Oder aber Sie geben einen hier unpassenden Wert wie *Herdt* oder *Bodenheim* an. Die daraufhin erscheinende Fehlermeldung ist aussagekräftig und verrät Ihnen alle gültigen Werte des Parameters.



Diese Technik ist sehr praktisch: Geben Sie z. B. Get-Command - CommandType Herdt ein, erhalten Sie die folgende Fehlermeldung (Ausschnitt):

Fehler: "Der Bezeichner "Herdt" kann keinem gültigen Enumeratornamen zugeordnet werden. Geben Sie einen der folgenden Enumeratornamen an, und wiederholen Sie den Vorgang: **Alias, Function, Filter, Cmdlet, ExternalScript, Application, Script, Workflow, Configuration, All.**"

Der Wert der Eigenschaft - CommandType wird dabei als Enumerator bezeichnet. Die gültigen Werte sind fest definiert und fett markiert in der obigen Fehlermeldung.

### Get-Befehle: Auswahl zur Informationsbeschaffung



Für den Einstieg in die Arbeit mit der PowerShell können Sie mit Befehlen experimentieren, die als Verb *Get* verwenden. Bei diesen Befehlen können Sie sicher sein, dass Sie keine – vielleicht ungewollten – Änderungen am System durchführen.

Häufig verwendet werden diese Befehle, die Sie in der folgenden Übersicht sehen:

Cmdlet	Kurzbeschreibung
Get-ChildItem	Listet den Verzeichnisinhalt eines angegebenen Verzeichnisses auf
Get-Date	Ruft aktuelle Datums- und Uhrzeitangaben auf

Cmdlet	Kurzbeschreibung
Get-History	Zeigt eine Liste der zuletzt eingegebenen Befehle in der aktuellen PowerShell-Sitzung an
Get-Module	Zeigt die PowerShell-Module an, die aktuell geladen wurden. Geben Sie den Parameter -ListAvailable an, zeigt die PowerShell alle verfügbaren, noch <b>nicht</b> geladenen Module an.
Get-NetIPAddress	Ruft Informationen zur IPv4- und IPv6-Adressierung eines Rechners ab
Get-Process	Zeigt die aufgeführten Prozesse auf einem Rechner an
Get-Service	Ruft die Dienste auf einem Rechner ab
Get-PSDrive	Zeigt die PowerShell-Laufwerke der aktuellen Sitzung an
Get-PSPowerProvider	Fordert Informationen über PowerShell-Provider an

Zum Testen können Sie alle angegebenen Cmdlets ohne weitere Parameter verwenden. Die komplette Funktionalität entfalten Cmdlets durch ihre Parameter, wie in der Übersicht bei dem Cmdlet Get-Module angedeutet. Oft werden Cmdlets – sofern syntaktisch zulässig – für einen ersten Überblick ohne Parameter eingesetzt und anschließend durch Parameter verfeinert.

Am Ende des Kapitels werden Sie in den Übungen mit den Ihnen bereits bekannten Methoden nach Parametern und Einsatzgebieten für diese Cmdlets suchen.

### Get-Member: Weitere Informationen zu Befehlen

Wenn Sie Informationen über einen Get-Befehl erhalten, erhalten Sie von der PowerShell eine Standardausgabe mit einer Vorauswahl an angezeigten Informationen. Sie wissen allerdings nicht, welche Eigenschaften und Methoden das angezeigte Objekt sonst noch bereithält. Es sind in der Regel weit mehr Daten, auf die Sie bei Bedarf zugreifen können.

Methoden sind Aktionen, die für das betreffende Objekt ausgeführt werden. Eigenschaften sind Daten, die mit dem Objekt verknüpft sind.

Bei den Fragen rund um verfügbare Eigenschaften und Methoden hilft das Cmdlet Get-Member. Zuerst nehmen Sie Verbindung zum Objekt auf und reichen es dann an Get-Member weiter. Get-Member listet dann die Eigenschaften und Methoden des übergebenen Objekts auf. Mithilfe der sogenannten Pipeline werden die nötigen Informationen an das zweite Cmdlet übergeben. Ausführliche Informationen zum PowerShell-Konzept der Pipeline lesen Sie im nächsten Kapitel.

Get-Member wird häufig ohne Parameter verwendet. Geläufige Parameter des Cmdlets sehen Sie in der folgenden Übersicht:

Get-Member		
Parameter	Typ	Beschreibung
-Name	P (1)	Filtermöglichkeit für Eigenschaften oder Methoden des Objekts. Nur angegebene Werte oder Teile von Werten werden angezeigt. Wird meist bei der Suche nach einer bestimmten Eigenschaft oder einer bestimmten Methode eingesetzt.
-MemberType	N	Begrenzt die Suche auf angegebene Elementtypen. Standardeinstellung ist All. Die gängigsten Werte sind <i>Method</i> und <i>Property</i> .
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help -Name Get-Member -Full

Anhand eines Beispiels zeigt sich, wie nützlich Get -Member sein kann:

```
Get-Service | Get-Member
```

- ✓ Sie generieren mit dem Cmdlet Get -Service eine Liste aller Dienste Ihres Rechners.
- ✓ Diese Liste wird nicht am Bildschirm angezeigt, sondern über den Pipeline-Operator (senkrechter Strich) an das zweite Cmdlet übergeben.
- ✓ Get-Member zeigt schließlich am Bildschirm an, über welche Eigenschaften die Dienste verfügen (standardmäßig werden nur Dienstname, Anzeigename und Status der Dienste angezeigt) und welche Aktionen auf sie angewendet werden können.
- ✓ Die Ausgabe zeigt, dass Methoden wie u. a. *Start*, *Stop*, *Pause* und *Refresh* sowie Eigenschaften wie beispielsweise *CanShutdown*, *CanStop* und *MachineName* zur Verfügung stehen.

Sie beschaffen sich über das Cmdlet Get -Member Informationen über Objekte und erfahren, welche weiteren Daten zur Verfügung stehen und welche Aktionen Sie anwenden können. Wie Sie die erlangten Informationen zur Weiterverarbeitung einsetzen, erfahren Sie im folgenden Kapitel.

### Show-Command: interaktiver Cmdlet-Generator

Das Cmdlet Show-Command öffnet ein eigenes Fenster und zeigt – für ein von Ihnen spezifiziertes Cmdlet – ein Formular, in das Sie die Werte für Parameter eingeben. Sie setzen also interaktiv ein Cmdlet mit allen gewünschten Parametern zusammen. Schließlich kopieren Sie den kompletten Befehl in die Zwischenablage oder führen ihn direkt aus.

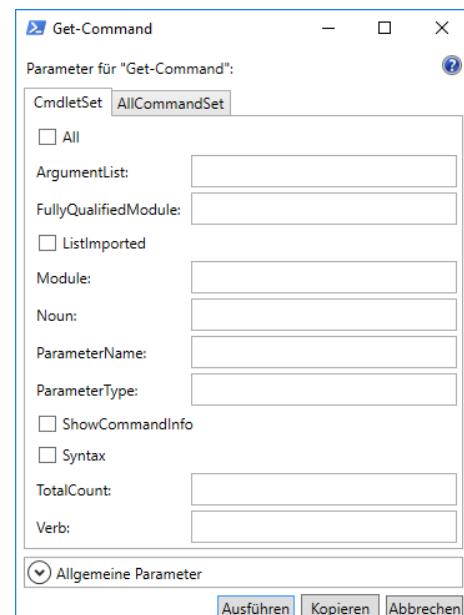
Show-Command ist vor allem dann sehr nützlich, wenn Sie sich syntaktisch noch nicht ganz sicher fühlen oder sich an mögliche Parameter für das gewünschte Cmdlet nicht erinnern.

Rufen Sie das Cmdlet ohne Parameter auf, erhalten Sie in dem neuen Fenster eine Liste aller Cmdlets, aus der Sie auswählen können.

Rufen Sie Show-Command mit dem Parameter-Namen auf und geben als Wert den Namen des gewünschten Cmdlets an, erhalten Sie direkt das Formular wie in der nebenstehenden Abbildung.

Der komplette Befehl im Beispiel lautet:

```
Show-Command -Name Get-Command
```



Eigenes Fenster von Show-Command

Muss ein Parameter zwingend angegeben werden, ist er mit einem Stern markiert.

Klicken Sie auf das Fragezeichensymbol im rechten oberen Fensterbereich, öffnet sich ein weiteres Fenster mit dem kompletten Text der Hilfe zu diesem Cmdlet.

Show-Command kennt u. a. folgende Parameter:

Show-Command		
Parameter	Typ	Beschreibung
-Name	P (1)	Angabe z. B. des gewünschten Cmdlets, mit dem Sie im neuen Fenster arbeiten wollen
-Height	N	Festlegung der Fensterhöhe des neuen Fensters (Wert in Pixel)
-Width	N	Festlegung der Fensterbreite des neuen Fensters (Wert in Pixel)
-NoCommonParameter	S	Die Anzeige allgemeiner Parameter wird unterdrückt, wenn der Parameter angegeben wird.
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help -Name Show-Command -Full

### Get-Help: die integrierte Hilfe

Ein überaus nützliches Cmdlet ist Get-Help, das Ihnen Zugriff auf das PowerShell-Hilfesystem ermöglicht. Da Sie aber bis zur funktionierenden Hilfe einige Schritte absolvieren müssen, finden Sie umfassende Informationen direkt anschließend in einem eigenen Abschnitt.

### Get-History: zuletzt eingegebene PowerShell-Befehle

Die PowerShell protokolliert die während der aktuellen PowerShell-Sitzung eingegebenen Befehle. Sie können sich durch die bisher eingegebenen Befehle mit den Tasten **[↑]** und **[↓]** bewegen. Suchen Sie allerdings einen Befehl, den Sie nicht als letzten oder vorletzten Befehl eingegeben haben, ist dies relativ mühsam und nicht unbedingt zu empfehlen.

Die PowerShell stellt das Cmdlet Get-History zur Verfügung, mit dessen Hilfe die in der aktuellen PowerShell-Sitzung eingegebenen Befehle angezeigt werden:

#### Get-History

Die eingegebenen Befehle werden zusammen mit einer ID-Nummer angezeigt, die Sie verwenden, wenn Sie einen Befehl erneut ausführen möchten. In der PowerShell 4.0 werden standardmäßig die letzten 4096 Befehle protokolliert. Die Festlegung erfolgt über die interne Variable \$MaximumHistoryCount, deren Standardwert 4096 ist. Den aktuellen Wert sehen Sie, wenn Sie in der Konsole nur den Variablennamen eingeben. Sie können der Variablen bei Bedarf einen anderen Wert – zwischen 1 und 32767 – zuweisen:

```
$MaximumHistoryCount = <Wert zwischen 1 und 32767>
```

Zum erneuten Ausführen eines bereits früher in der Sitzung eingegebenen Befehls verwenden Sie das Cmdlet Invoke-History unter Angabe der ID-Nummer des Befehls:

```
Invoke-History -Id <ID>
```

Sollten Sie die bislang eingegebenen Befehle der aktuellen PowerShell-Sitzung aus der Liste entfernen wollen, verwenden Sie das Cmdlet Clear-History ohne weitere Parameter.

Beachten Sie, dass sich die PowerShell keine eingegebenen Befehle über die Grenzen der aktuellen Sitzung hinaus merkt. Beenden Sie die aktuelle PowerShell-Sitzung und öffnen eine neue, ist die Liste zu Beginn leer und wird neu geführt.

## 2.5 Das Hilfesystem der PowerShell

### Zu Beginn Hilfe nur online

Das Cmdlet Get-Help ruft das integrierte, überaus ausführliche Hilfesystem zur PowerShell auf. Es zeigt Informationen zu PowerShell-Befehlen und -Konzepten. Allerdings enthält die PowerShell zu Beginn noch keine lokalen Hilfedateien.

- Geben Sie in der PowerShell-Konsole Folgendes ein:

```
Get-Help
```

Sie erhalten eine Ausgabe, dass online eine Hilfe zur Verfügung steht, die über den Switch-Parameter -Online erreichbar ist.

Wenn Sie die Onlinehilfe für ein beliebiges Cmdlet verwenden wollen, richten Sie sich nach der allgemeinen Syntax:

```
Get-Help -Name <Cmdlet> -Online
```

Die Hilfe wird daraufhin im Standard-Webbrowser angezeigt, sofern eine Internetverbindung besteht.

Geben Sie das Cmdlet ohne Parameter -Online ein, greift PowerShell auf lokale Hilfedateien zurück. Da aber die Hilfedateien erst aus dem Internet geladen werden müssen, erhalten Sie nur einen kurzen Überblick über die Syntax des Cmdlets sowie den Hinweis, dass für dieses Cmdlet keine Hilfedateien gefunden wurden.

```
PS C:\Users\AD> Get-Help -Name Get-Command

NAME
  Get-Command

SYNTAX
  Get-Command [[-ArgumentList] <Object[]>] [-Verb <string[]>] [-Noun
    <string[]>] [-Module <string[]>] [-FullyQualifiedModule
    <ModuleSpecification[]>] [-TotalCount <int>] [-Syntax] [-ShowCommandInfo]
    [-All] [-ListImported] [-ParameterName <string[]>] [-ParameterType
    <PSTypeName[]>] [<CommonParameters>]

  Get-Command [[-Name] <string[]>] [[-ArgumentList] <Object[]>] [-Module
    <string[]>] [-FullyQualifiedModule <ModuleSpecification[]>] [- CommandType
    <CommandTypes> {Alias | Function | Filter | Cmdlet | ExternalScript |
    Application | Script | Workflow | Configuration | All}] [-TotalCount
    <int>] [-Syntax] [-ShowCommandInfo] [-All] [-ListImported] [-ParameterName
    <string[]>] [-ParameterType <PSTypeName[]>] [<CommonParameters>]

ALIASE
  gcm

HINWEISE
  Die Hilfedateien für dieses Cmdlet können von "Get-Help" auf diesem
  Computer nicht gefunden werden. Es wird nur ein Teil der Hilfe angezeigt.
    -- Sie können die Hilfedateien für das Modul, das dieses Cmdlet
    enthält, herunterladen und installieren, indem Sie "Update-Help" verwenden.
    -- Wenn Sie das Hilfethema für dieses Cmdlet online anzeigen möchten,
    geben Sie Folgendes ein: "Get-Help Get-Command -Online", oder
    gehen Sie zu "http://go.microsoft.com/fwlink/?LinkId=113309".
```

Die PowerShell-Hilfe steht lokal (noch) nicht zur Verfügung.

## Hilfdateien aus dem Internet laden: **Update-Help**

Es empfiehlt sich, die aktuellen Hilfdateien auf den eigenen Computer herunterzuladen. Dies ist ein einmaliger Vorgang. Dazu steht Ihnen das Cmdlet **Update-Help** zur Verfügung.

- ▶ Starten Sie die PowerShell mit vollen Administratorrechten, z. B. durch Rechtsklick auf das PowerShell-Symbol in der Taskleiste und Auswahl des Eintrags *Als Administrator ausführen* im Kontextmenü.
- ▶ Geben Sie in der PowerShell-Konsole folgende Zeile ein:

```
Update-Help -UICulture en-US -Force
```

Mit diesem Cmdlet werden die Hilfetexte für die PowerShell aus dem Internet heruntergeladen. Da für die meisten Befehle keine deutsche Hilfe zur Verfügung steht, wird mithilfe des Parameters **-UICulture** explizit die englischsprachige Hilfe (**en-US**) angefragt und auf den lokalen Computer übertragen. Der Parameter **-Force** sorgt dafür, dass ein Download der Hilfdateien sofort stattfindet und nicht nur einmal in 24 Stunden, mit einer Versionsprüfung sowie einer Größenbeschränkung der Hilfdateien.

Da die Hilfdateien in Unterverzeichnissen des Windows-Ordners gespeichert werden, können Sie den Download nur mit vollen administrativen Rechten ausführen.

```
Administrator: Windows PowerShell
PS C:\> Update-Help -UICulture en-us -Force

Hilfe für Modul NetworkSwitchManager wird aktualisiert
Der Hilfeinhalt wird heruntergeladen...
[oooooooooo]
```

*Laden der Hilfdateien via Internet durch **Update-Help***



Für Computer ohne Internetzugang oder bei der Verteilung von Betriebssystemen haben Sie zusätzlich die Möglichkeit, mit Hilfe des Cmdlets **Save-Help** z.B. in eine Netzwerkfreigabe oder einen bereits vorhandenen lokalen Ordner zu speichern und von dort auf die anderen Computer zu verteilen.

```
Save-Help -DestinationPath <Netzwerkfreigabe>
```

Die Hilfdateien stehen seit PowerShell 3.0 nur für wenige Befehle in deutscher Sprache zur Verfügung. Es ist eher unwahrscheinlich, dass die Hilfe zukünftig ins Deutsche übersetzt wird.

Es ist normal, wenn beim Aktualisieren der Hilfe einige wenige Fehlermeldungen für einzelne Module erscheinen. Dies betrifft in der Regel nicht abrufbare oder online nicht verfügbare Hilfeinhalte. Sie können diese Meldungen ignorieren, sofern sie nicht alle verfügbaren Module betreffen. In dem Fall prüfen Sie Ihre Internetverbindung. Möglicherweise haben Sie die PowerShell auch nicht als Administrator gestartet.

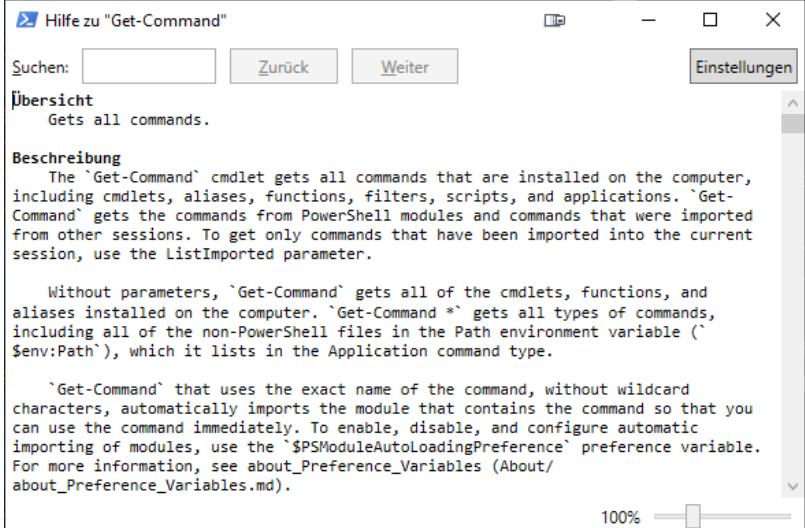
## Hilfe effektiv nutzen: **Get-Help**

Nachdem Sie die Hilfdateien aus dem Internet geladen haben, steht Ihnen lokal an Ihrem Rechner die komplette Hilfe der PowerShell zur Verfügung.

Der Aufruf des Cmdlets folgt der folgenden allgemeinen Syntax:

```
Get-Help -Name <Cmdlet/Konzept> [-Parameter]
```

- ✓ Get-Help zeigt einen einfachen Hilfetext zu einem angegebenen PowerShell-Cmdlet oder -Konzept an.
- ✓ Durch weitere Parameter können Sie die besonderen Stärken der PowerShell-Hilfe nutzen. Die nachfolgende Tabelle zeigt Ihnen wichtige Parameter des Cmdlets:

Get-Help		
Parameter	Typ	Beschreibung
-Name	P (1)	Angabe des Cmdlets oder Konzepts, zu dem Sie Hilfe anfordern
-Detailed	S	Angabe des Cmdlets oder Konzepts, zu dem Sie Hilfe anfordern
-Examples	S	Zeigt für Cmdlets nur Namen, Zusammenfassung und Beispiele an
-Full	S	Zeigt für Cmdlets die vollständige Hilfe an
-Online		Ruft die Onlineversion für ein Hilfethema an
-Parameter	N	Ruft für das gewünschte Cmdlet Informationen zu angegebenen Parametern ab. Bei der Angabe des Werts sind Wildcards erlaubt.
-ShowWindow	S	<p>Zeigt den kompletten Hilfetext in einem eigenen Fenster an, in dem Sie die integrierte Suchfunktion nutzen und Einstellungen zu anzuzeigenden Hilfeabschnitten vornehmen können.</p> <p>Wichtig: Andere Switch-Parameter müssen Sie weglassen, ansonsten kommt es zu einer Fehlermeldung ("Der Parametersatz kann mit den angegebenen benannten Parametern nicht aufgelöst werden.") und der Befehl wird nicht ausgeführt.</p>  <p>Eigenes Hilfafenster (Get-Help -Name Get-Command -ShowWindow)</p>
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help -Name Get-Help -Full

Investieren Sie Zeit in die Beschäftigung mit dem Hilfesystem der PowerShell. Neben Syntaxerläuterungen finden Sie viele Beispiele, die Sie auf neue Ideen bringen.

Bei der Verwendung von Get-Help können Sie auch Suchbegriffe verwenden. Geben Sie z. B. Get-Help -Name process ein, listet die PowerShell alle Cmdlets auf, in denen die Zeichenfolge process vorkommt. Da wichtige Spalten nicht vollständig dargestellt werden, empfiehlt sich eine Anpassung der Eingabe:

```
Get-Help -Name process | Out-GridView
```

- ✓ Der linke Teil der Zeile besteht aus einem kompletten Cmdlet. Dann folgt als Trennzeichen ein senkrechter Strich, die sogenannte Pipeline.
- ✓ Letztlich folgt ein weiteres komplettes Cmdlet, das die Ergebnisse des ersten Cmdlets aufgreift und in einer Tabelle in einem eigenen interaktiven Fenster anzeigt.

Interessant ist auch die Möglichkeit, Cmdlets zu finden, die einen bestimmten Parameter unterstützen. Dies erreichen Sie durch folgende Eingabe:

```
Get-Help -Name * -Parameter Wert
```

- ✓ Sie erhalten eine Liste aller Cmdlets, die den angegebenen Parameter unterstützen.
- ✓ Wenn Sie z. B. im Verlauf des Buchs dem Parameter *ComputerName* begegnen, ist eine solche Liste interessant. Über den Parameter werden Aktionen auf Remoterechnern unterstützt, aber nur wenige Cmdlets unterstützen den Parameter.

## Hilfe für PowerShell-Konzepte

Die PowerShell-Hilfe kann aber noch mehr. Es stehen ausführliche Texte zu „Konzepten“ bzw. grundlegenden Themen bereit. All diese Themen beginnen mit der Zeichenkette *about\_*.

- Um alle grundlegenden Themen der PowerShell-Hilfe anzuzeigen, geben Sie die folgende Zeile ein:

```
Get-Help -Name about_*
```

- ✓ Sie erhalten eine Liste mit mehr als 100 Hilfetexten zu PowerShell-Konzepten, darunter sind z. B. Texte zu den allgemeinen Parametern (*about\_CommonParameters*), den in späteren Kapiteln angesprochenen Aliassen (*about\_Aliases*) oder Profilen (*about\_Profiles*).



Die Switch-Parameter *-Detailed*, *-Examples* und *-Full* haben keine Auswirkungen auf die Anzeige der Hilfetexte zu konzeptionellen Themen. Der Parameter *-ShowWindow* sei Ihnen für eine bessere Lesbarkeit in einem separaten Fenster an dieser Stelle besonders empfohlen.

Unter dem Titel *Konzeptionelle Hilfethemen für Windows PowerShell Core* stehen diese Texte zum angenehmen Lesen ebenfalls online unter der Adresse [https://docs.microsoft.com/de-de/previous-versions/windows/powershell-scripting/hh847856\(v=wps.640\)](https://docs.microsoft.com/de-de/previous-versions/windows/powershell-scripting/hh847856(v=wps.640)) bereit.

## Beispiele für die Verwendung von Get-Help

In der nachfolgenden Tabelle finden Sie einige Beispiele, die Ihnen die Verwendung des Cmdlets *Get-Command* veranschaulichen soll:

Sie möchten ...	
alle PowerShell-Befehle auflisten	Get-Help
Tipps zur Verwendung des Cmdlets <i>Get-Command</i> durch die Anzeige von Beispielen erhalten	Get-Help -Name <i>Get-Command</i> -Examples
eine Liste aller PowerShell-Konzepte sehen, zu denen Hilfetexte zur Verfügung stehen	Get-Help -Name <i>about_*</i>
den Hilfetext zur Beschreibung des Parameters <i> CommandType</i> des Cmdlets <i>Get-Command</i> abrufen	Get-Help -Name <i>Get-Command</i> -Parameter <i> CommandType</i>
in Ihrem Standardbrowser die Onlinehilfe zum Cmdlet <i>Get-Command</i> anzeigen lassen	Get-Help -Name <i>Get-Command</i> -Online
die Hilfe zum PowerShell-Konzept der allgemeinen Parameter in einem eigenen Fenster lesen	Get-Help -Name <i>about_CommonParameters</i> -ShowWindows
alle Parameter des Cmdlets <i>Get-Command</i> inklusive Beschreibung und Definition anzeigen	Get-Help -Name <i>Get-Command</i> -Parameter *

## 2.6 Kurz zusammengefasst

### Durchgängig klare Strukturen

Cmdlets sind die internen Funktionen bzw. Komplettlösungen der PowerShell. Durch ihren klar strukturierten Aufbau *Verb-Substantiv-Parameter* erleichtern Sie Ihnen die Orientierung. Je nach Aufgabenstellung wird schnell klar, welches Verb zu verwenden ist, abhängig vom Themengebiet ergibt sich mindestens ein Teil des Substantivs. Zu Beginn der Beschäftigung mit der PowerShell bieten sich *Get*-Befehle an, die Informationen liefern, aber keine Veränderungen am System vornehmen.

Weitere Unterstützung bei der Verwendung von Cmdlets steckt in der PowerShell selbst. Ausführliche Hilfetexte, Beispiele und Syntaxerläuterungen weisen den Weg.

Mit eigenen, etwas grafischer orientierten Fenstern durch das Cmdlet *Show-Command* und den Hilfeparameter *-ShowWindow* erhalten Anwender Hilfen im vertrauten Windows-Stil.

Nach der Orientierung in der PowerShell und den gezeigten Möglichkeiten zur Selbsthilfe und Informationsbeschaffung lesen Sie im nächsten Kapitel, wie Sie Daten weiter verarbeiten können.

## 2.7 Übungen

### Verwenden der PowerShell-Hilfe

**Übungsdatei:** –

**Ergebnisdatei:** *2\_ergebnisse.txt*

1. Führen Sie Update-Help auf Ihrem System aus. Wie lautet das vollständige Cmdlet, das den sofortigen Download garantiert?
2. Rufen Sie die Hilfe für das Cmdlet *Get-Service* auf. Vergleichen Sie die Ausgabe, wenn Sie keine weiteren Parameter angeben, mit der Ausgabe, die Sie jeweils bei zusätzlicher Eingabe des Parameters *-Detailed*, *-Full* bzw. *-Examples* erhalten.
3. Finden Sie heraus, welche Cmdlets den Parameter *-Verb* verwenden. Lesen Sie in der Hilfe zu einem der angezeigten Cmdlets nach, was der Parameter bedeutet.
4. Lassen Sie sich Beispiele für das Cmdlet *Get-Process* anzeigen.

### Verwenden wichtiger Cmdlets

**Übungsdatei:** –

**Ergebnisdatei:** *2\_ergebnisse.txt*

1. Finden Sie heraus, welche Befehle das Verb *Import* verwendet.
2. Welche Eigenschaften von Diensten auf Ihrem Rechner kann das Cmdlet *Get-Service* anzeigen?
3. Recherchieren Sie mit den im Kapitel vorgestellten Mitteln
  - a) die Syntax des Cmdlets *Get-Module*,
  - b) Beispiele zum Einsatz des Cmdlets *Get-PSDrive*,
  - c) was der Parameter *-Format* des Cmdlets *Get-Date* bedeutet.
4. Lesen Sie die vollständigen Hilfetexte zu den vorgestellten Get-Befehlen *Get-History*, *Get-Process* und *Get-PSProvider*.
5. Finden Sie heraus, welche Eigenschaften Ihnen für das Cmdlet *Get-Module* zur Verfügung stehen.

# 3 Informationen verarbeiten und ausgeben



Beispieldatei: 3\_kompletter code.txt und hosts.txt (Ordner Kapitel 03)

## 3.1 Die PowerShell-Pipeline

### Warum PowerShell-Pipelines benötigt werden

Im letzten Kapitel haben Sie eine Reihe von Cmdlets kennengelernt, die zur Informationsbeschaffung gedacht sind. Sie kennen auch bereits das Cmdlet `Get-Member`, um Eigenschaften und Methoden von Objekten zu finden. Nicht zuletzt wissen Sie die PowerShell-Hilfe zu bedienen, um nützliche Beschreibungen und Beispiele zu erhalten.

Was aber tun Sie, wenn die benötigte spezielle Information nicht angezeigt wird? Wie verarbeiten Sie die vorliegenden Informationen, z. B. um sie zu filtern oder zu exportieren?

Hier reicht ein einzelnes Cmdlet nicht mehr aus. Sie können das Verhalten von Cmdlets nur in engen Grenzen mithilfe von Parametern steuern. Im Resultat bedeutet das, dass eine spezielle Aufgabe nach Ihren Vorstellungen funktioniert oder nicht. Damit müssen Sie sich nicht zufriedengeben.

Die Lösung lautet: Lassen Sie die PowerShell-Cmdlets im Team arbeiten, indem Sie sie kombinieren. Vergleichbar ist das mit der Produktion eines Artikels in der Wirtschaft. Nach einem Bearbeitungsschritt wird ein Objekt, das noch nicht Ihren Erwartungen entspricht, an eine andere Arbeitsstation weitergegeben.

Ähnlich können Sie die PowerShell verwenden, indem Sie die PowerShell-Pipeline verwenden. Sie bedienen sich des Pipelineoperators „|“ zur Verkettung der einzelnen Arbeitsschritte.

### Prinzip der Pipeline

Mithilfe des Pipelineoperators bilden Sie eine Befehlskette. Von einem Befehl ausgegebene Objekte werden zur Weiterverarbeitung an den nächsten Befehl der Kette gesendet. Damit erreichen Sie komplexere Problemlösungen, als es mit einem einzelnen Cmdlet möglich wäre.

Wichtig ist hierbei, dass in der Pipeline vollständige Objekte weitergegeben werden. Es handelt sich nicht um die Weitergabe einer Zeichenkette, wie bei vielen anderen Shells.

Befehlsketten sind in der Regel nach dem folgenden Prinzip aufgebaut:

Bearbeitungsschritt	Erläuterung	Häufige Cmdlets
1 Daten bereitstellen	Im ersten Schritt benötigen Sie Daten, mit denen Sie arbeiten wollen. Dafür setzen Sie in der Regel Cmdlets mit dem Verb <i>Get</i> ein.	Get-...
2 Ergebnis bearbeiten, bis es den Erwartungen entspricht	Hier werden Cmdlets eingesetzt, die mit Daten arbeiten, die sie über die Pipeline erhalten. Häufig finden Sie in diesem zweiten Schritt, der auch aus etlichen Teilschritten bestehen kann, Cmdlets mit dem Substantiv <i>Object</i> .	...-Object
3 Endergebnis ausgeben	Lassen Sie diesen Schritt weg, erfolgt eine Ausgabe im Konsolenfenster. Ansonsten haben Sie die Möglichkeit, <ul style="list-style-type: none"> <li>✓ die Formatierung Ihren Wünschen anzupassen,</li> <li>✓ die Ausgabe umzuleiten, z. B. in eine Datei zu exportieren.</li> </ul>	Export-... Format-... Out-...

Eine Ausgabe erfolgt – sofern vorgesehen – erst im letzten Schritt, also mit dem letzten Cmdlet der Pipeline. Alle vorherigen Cmdlets leiten ihre Objekte direkt an das Cmdlet im nächsten Bearbeitungsschritt weiter. Soll das letzte Cmdlet z. B. das Ergebnis in einer Datei exportieren, erfolgt keine Ausgabe im Konsolenfenster.

Im letzten Kapitel haben Sie den ersten Bearbeitungsschritt kennengelernt und Informationen gesammelt. Die optionalen weiteren Schritte der Ergebnisbearbeitung und -ausgabe lernen Sie in den nachfolgenden Abschnitten kennen. Dabei lernen Sie in Beispielen, wie die PowerShell-Pipeline eingesetzt wird.

## 3.2 Verarbeitung vorliegender Daten: Object-Cmdlets

### Wichtige Cmdlets, die die Pipeline benötigen

Fast alle Object-Cmdlets benötigen Daten aus der Pipeline, mit denen sie arbeiten können. Wenn Sie nach Cmdlets mit dem Substantiv *Object* suchen (Get-Command -Noun object), erhalten Sie die neun Cmdlets als Ergebnis. Während die Cmdlets Compare-Object (Vergleich zweier Sätze von Objekten) und New-Object (Erstellen eines neuen Objekts) an dieser Stelle keine Rolle spielen, werden die übrigen sieben Cmdlets in diesem Zusammenhang häufig verwendet.

Die folgende Übersicht zeigt Ihnen, um welche Cmdlets es sich handelt und welche Aufgabe sie erfüllen (jedes Cmdlet wird weiter unten ausführlich besprochen):

Cmdlet	Kurzbeschreibung
ForEach-Object	Erlaubt die Anwendung eines Skriptblocks auf jedes übergebene Objekt, funktioniert wie eine Schleife, die alle einzelnen Elemente durchläuft <i>Beispiel:</i> 3,4,5,6,7   ForEach-Object -Process {\$_*2} Sie wollen alle vorliegenden Zahlenwerte verdoppeln.
Group-Object	Gruppert übergebene Objekte nach den Werten ihrer Eigenschaften <i>Beispiel:</i> Get-Service   Group-Object -Property Status Die Liste der Dienste auf Ihrem Rechner sollen nach ihrem Status (gestartet oder beendet) gruppiert werden.
Measure-Object	Berechnet numerische Eigenschaften von Objekten wie Anzahl, Mittelwert, Summe etc. <i>Beispiel:</i> Get-Command - CommandType cmdlet   Measure-Object Sie wollen ermitteln, wie viele Cmdlets aktuell in der PowerShell verfügbar sind.
Select-Object	Wählt Eigenschaften eines Objekts gemäß Ihren Wünschen aus <i>Beispiel:</i> Get-Process   Select-Object -Property Id, ProcessName Sie möchten von der Liste der laufenden Prozesse nur die Eigenschaften <i>Id</i> und <i>ProcessName</i> sehen.
Sort-Object	Sortiert Objekte nach den Werten der Eigenschaften und entfernt bei Bedarf Mehrfachwerte <i>Beispiel:</i> Get-ChildItem   Sort-Object -Property length Sie wollen die Dateien bei einer Auflistung des aktuellen Verzeichnisses nach Größe sortieren.
Tee-Object	Die Ausgabe des Befehls wird in einer Datei oder Variablen gespeichert und zusätzlich in der Konsole angezeigt. <i>Beispiel:</i> Get-Service   Tee-Object -FilePath .\dienste.txt Eine Liste der Dienste auf Ihrem Rechner wird in der Konsole angezeigt und gleichzeitig in die Datei <i>dienste.txt</i> im aktuellen Verzeichnis gespeichert.

Cmdlet	Kurzbeschreibung
Where-Object	<p>Filtert Objekte nach beliebigen Kriterien</p> <p><i>Beispiel:</i> Get-ChildItem C:\Windows\System32   Where-Object {\$_ .length -gt 10mb}</p> <p>Sie suchen im Verzeichnis C:\Windows\System32 nach Dateien mit einer Dateigröße von mehr als 10 MB.</p>

Sie sehen bereits an diesen Beispielen, wie effektiv eine solche Befehlskette funktionieren kann. Schauen Sie sich in den nächsten Abschnitten die vorgestellten Cmdlets nach ihrer Verwendungshäufigkeit noch etwas genauer an.

### Select-Object – Filterung ganz nach Wunsch

Das Cmdlet Select-Object ist eines der am häufigsten verwendeten Cmdlets in einer Pipeline. Mit seiner Hilfe wählen Sie die Eigenschaften eines Objekts aus, die Sie sehen wollen. Dabei stehen alle für das Objekt definierten Eigenschaften zur Verfügung. Es spielt keine Rolle, ob die gewünschte Eigenschaft in der Standardausgabe der PowerShell angezeigt wird.

Für das Cmdlet stehen u. a. folgende Parameter zur Verfügung:

Select-Object		
Parameter	Typ	Beschreibung
-Property	P (1)	Gibt die Eigenschaft(en) an, die ausgewählt werden soll(en). Dies kann auch eine Eigenschaft sein, die Ergebnis einer Berechnung ist.
-First	N	Gibt die ersten x Objekte des Ergebnisses an (x ist der angegebene Zahlenwert des Parameters)
-Last	N	Gibt die letzten x Objekte des Ergebnisses an (x ist der angegebene Zahlenwert des Parameters)
-Skip	N	Überspringt x Objekte des Ergebnisses (x ist der angegebene Zahlenwert des Parameters)
-Unique	S	Zeigt alle mehrfach vorkommenden Werte nur einmalig an
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help -Name Select-Object -Full

In den folgenden praktischen Beispielen sehen Sie Anwendungsmöglichkeiten für das Cmdlet Select-Object:

- 1) Gefällt Ihnen die Standardausgabe eines PowerShell-Cmdlets nicht, weil sie zu viele Informationen liefert, filtern Sie die Ausgabe:

```
Get-Service -Name s* | Select-Object -Property Name, Status
```

- ✓ Für alle Prozesse, deren Name mit dem Buchstaben s beginnt, werden nur noch die Eigenschaften *Name* und *Status* angezeigt. Die Namen der Eigenschaften sehen Sie als Spaltenkopf in der Standardausgabe.

- 2) Vergleichen Sie die Ausgabe der PowerShell bei Eingabe der beiden folgenden Zeilen:

```
Get-Process -Name powershell
```

- ✓ Sie erhalten Informationen zum laufenden PowerShell-Prozess, standardmäßig formatiert als Tabelle mit acht vorgegebenen Spalten.

```
Get-Process -Name powershell | Select-Object -Property *
```

- ✓ PowerShell ändert die Formatierung und zeigt nunmehr eine Liste von mehr als 60 Eigenschaften des Prozesses an. Sie können gewünschte Eigenschaften direkt benennen oder Wildcards einsetzen.
- ✓ Über diesen Weg erhalten Sie als Nebenprodukt Kenntnis aller Namen der verfügbaren Eigenschaften. Alternativ finden Sie diese mit der folgenden Zeile heraus:

```
Get-Process -Name powershell | Get-Member
```

- 3) Sie haben eine Liste mit Namen von Siegern eines PowerShell-Wettbewerbs. Sie möchten aber nur herausfinden, welche Personen bereits den Wettbewerb gewonnen haben:

```
"Sam", "Hugo", "Sam", "Sam", "Karla", "Hugo" | Select-Object -Unique
```

- ✓ Als Ergebnis liefert die PowerShell drei Namen zurück, doppelte Werte werden ignoriert.
- ✓ Sie können dieses Beispiel gut mit den Parametern `-Skip`, `-First` oder `-Last` kombinieren. In der Praxis bietet sich an, vor der Auswahl von Eigenschaften die Objekte zu sortieren, so dass Sie eine Anzahl Objekte aus einer Art Rangliste wählen können.

### Sort-Object – Sortieren von Ergebnissen

Das Sortieren von Werten der Objekteigenschaften gehört zu den Standardaufgaben, wenn man eine Anzahl von Objekten weiter verarbeiten möchte. Die PowerShell stellt für diese Aufgabe das Cmdlet `Sort-Object` mit u. a. folgenden Parametern zur Verfügung:

Sort-Object		
Parameter	Typ	Beschreibung
<code>-Property</code>	P (1)	Gibt die Eigenschaft an, nach deren Werten sortiert werden soll. Kann weggelassen werden, woraufhin die Sortierung nach Standardeigenschaften des Objekttyps durchgeführt wird.
<code>-CaseSensitive</code>	S	Groß- und Kleinschreibung soll bei der Sortierung berücksichtigt werden.
<code>-Descending</code>	S	Die Sortierung wird in absteigender Reihenfolge durchgeführt.
<code>-Unique</code>	S	Entfernt Duplikate und zeigt nur eindeutige Werte an
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: <code>Get-Help -Name Sort-Object -Full</code>

Beispielanwendungen für die Sortierung von Objekten mithilfe des Cmdlets `Sort-Object` sind u. a.:

- 1) Sie lesen Einträge aus einem Ereignisprotokoll aus und möchten Sie nach der Eigenschaft `InstanceId` sortieren:

```
Get-EventLog -LogName System -Newest 20 | Sort-Object -Property InstanceID
```

- ✓ Sie erhalten die 20 jüngsten Einträge aus dem System-Ereignisprotokoll und geben Sie an das Cmdlet `Sort-Object` weiter, damit sie nach den Werten der Eigenschaft `InstanceId` aufsteigend sortiert werden.
- ✓ Eine absteigende Sortierung erreichen Sie, indem Sie den Parameter `-Descending` ergänzen.

- 2) Es bietet sich an, eine Dateiliste eines Verzeichnisses nach bestimmten Kriterien zu sortieren:

```
Get-ChildItem C:\Windows -File | Sort-Object -Property Extension, Length
```

- ✓ Das erste Cmdlet Get-ChildItem listet durch den Parameter -File ausschließlich Dateien des Windows-Verzeichnisses auf.
- ✓ Das folgende Cmdlet Sort-Object sortiert die übergebenen Objekte in erster Ebene nach ihrer Dateierweiterung, in zweiter Ebene zusätzlich aufsteigend nach Dateigröße.

### Where-Object – Filtern von Ergebnissen

Wenn Sie über die Pipeline übergebene Objekte inhaltlich filtern wollen, kann das Cmdlet Where-Object nützlich sein. Folgende Parameter stehen Ihnen u. a. zur Verfügung:

Where-Object		
Parameter	Typ	Beschreibung
-FilterScript	P (1)	Als Wert wird ein Skriptblock in geschweiften Klammern erwartet, der auf jedes übergebene Objekt angewendet werden soll. Innerhalb des Skriptblocks wird das aktuelle Objekt durch die Variable \$ <u>_</u> angesprochen.
-Property	P (1)	Alternativ zu -FilterScript: Es wird kein Skriptblock angegeben, sondern direkt eine Objekteigenschaft.
-Value	P(2)	Wird der Parameter -Property verwendet, wird mit dem Parameter -Value der Wert für den angegebenen Parameter angegeben.
Vergleichsoperatoren	N	Filter machen häufig nur Sinn, wenn Sie mit Vergleichsoperatoren (wie <i>größer als</i> , <i>kleiner als</i> , <i>gleich</i> etc.) arbeiten. So können Sie die Werte der Objekteigenschaften mit einer Vorgabe abgleichen und auswählen (oder nicht). In der neuen Version der PowerShell sind viele Vergleichsoperatoren als Parameter integriert (vgl. Abschnitt 7.5). Wichtige Parameter sind -EQ (gleich), -GT (größer als), -LT (kleiner als), die in den folgenden Beispielen verwendet werden. Die komplette Liste der Parameter erhalten Sie durch die folgende Eingabe: Get-Help Where-Object -Parameter *
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help -Name Where-Object -Full

Sie können bei der Verwendung des Cmdlets aus zwei Syntaxtypen wählen: die neue *vereinfachte Syntax* (seit PowerShell 3.0) oder die *klassische Syntax* (Skriptblock, der in geschweiften Klammern anzugeben ist). In den folgenden Beispielen, die die Verwendung von Where-Object verdeutlichen sollen, werden beide Typen parallel verwendet:

- 1) Sie wollen aus der Liste der vorhandenen Dienste nur die momentan ausgeführten Dienste filtern:

Vereinfachte Syntax:	Get-Service   Where-Object Status -EQ Running
Klassische Syntax:	Get-Service   Where-Object { \$_.Status -EQ "Running" }

- ✓ Mit dem angegebenen Befehl wird verglichen, ob der Wert der Eigenschaft *Status* der übergebenen Objekte mit *Running* („aktuell ausgeführt“) übereinstimmt. Falls nicht, wird das aktuelle Objekt verworfen und das nächste betrachtet. Falls ja, wird es in der Ausgabe angezeigt.

2) Sie wollen aus dem Windows-Verzeichnis alle Dateien mit einer Dateigröße kleiner als 100 kB anzeigen:

Vereinfachte Syntax:	<code>Get-ChildItem -Path C:\Windows -File   Where-Object Length -LT 100KB</code>
Klassische Syntax:	<code>Get-ChildItem -Path C:\Windows -File   Where-Object { \$_.Length -LT 100KB }</code>

- ✓ Der Wert der Eigenschaft *Length* wird mit dem angegebenen Wert 100 kB verglichen. Das Objekt wird angezeigt, wenn der Vergleich WAHR ergibt, ansonsten wird das Objekt übersprungen.

Mit beiden Schreibweisen erreichen Sie Ihr Ziel. Die klassische Schreibweise unterscheidet sich von der vereinfachten Syntax wie folgt:

- ✓ Die klassische Syntax arbeitet mit geschweiften Klammern. In diesem Skriptblock wird die Bedingung formuliert.
- ✓ Das aktuell verarbeitete Objekt wird durch die Zeichenfolge `$_` angesprochen.
- ✓ Zeichenketten sind in Anführungszeichen zu stellen.
- ✓ Im Gegensatz zur vereinfachten Syntax ist die klassische Syntax abwärtskompatibel.
- ✓ Im Skriptblock können mehrere Bedingungen miteinander verknüpft werden. Dies geschieht mit logischen Operatoren wie `-AND`, `-OR`, `-NOT` etc. Mit der vereinfachten Syntax geht dies nicht. Sie können nur in der Pipeline mehrere `Where-Object`-Cmdlets verwenden, was im Resultat einer AND-Verknüpfung entspricht.

### Beispiel

AND (beide Bedingungen müssen zutreffen), auch mit vereinfachter Syntax möglich.

```
Get-ChildItem -Path C:\Windows -File | Where-Object Length -LT 100KB |
Where-Object Length -GT 50KB
```

OR (eine der beiden Bedingungen muss zutreffen), nur mit klassischer Syntax möglich.

```
Get-ChildItem -Path C:\Windows -File | Where-Object { $_.Length -LT 100KB
-OR $_.Length -GT 1MB }
```

Sollte ein in der Pipeline weiter vorn stehendes Cmdlet die Möglichkeit zur Filterung bieten, sollten Sie sie nutzen. `Where-Object` ist sicherlich praktisch, aber bedenken Sie, dass erst einmal alle Objekte beschafft und übertragen werden müssen, bevor sie nachträglich gefiltert werden. Beherrscht das Cmdlet, das die Daten beschafft, direkt die gewünschte Filterung, müssen nur die nötigen Daten übertragen werden. Die Filterung findet vor der Übertragung statt, ist damit auch häufig deutlich schneller. Schreiben Sie also z. B.:



```
Get-Process -Name powershell
```

... und nicht:

```
Get-Process | Where-Object Name -EQ powershell
```

### ForEach-Object – dieselbe Aktion für alle Objekte

Das Cmdlet `ForEach-Object` führt dieselbe Aktion für alle Objekte aus, die über die Pipeline an das Cmdlet übergeben werden. Es wird also eine Schleife durchlaufen, die nacheinander die Objekte bearbeitet. Die Aktion, die dabei durchgeführt wird, ist frei programmierbar. Es kann sich durchaus um einen ganzen Skriptblock handeln.

Für das Cmdlet stehen u. a. folgende Parameter zur Verfügung:

ForEach-Object		
Parameter	Typ	Beschreibung
<code>-Process</code>	P (1)	Als Wert wird ein Skriptblock in geschweiften Klammern erwartet, der auf jedes übergebene Objekt angewendet werden soll. Innerhalb des Skriptblocks wird das aktuelle Objekt durch die Variable <code>\$_</code> angesprochen.
<code>-MemberName</code>	P (1)	Alternativ zu <code>-Process</code> : Es wird kein Skriptblock angegeben, sondern direkt eine anzugehende Eigenschaft oder eine auszuführende Methode.
<code>-Begin</code>	N	In geschweiften Klammern wird ein Skriptblock angegeben, der vor Ausführung der Aktion für alle Objekte verarbeitet wird.
<code>-End</code>	N	Angabe eines Skriptblocks in geschweiften Klammern, der nach Beendigung der Aktion für alle Objekte ausgeführt wird
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: <code>Get-Help -Name ForEach-Object -Full</code>

Folgende Beispiele zeigen Ihnen mögliche Einsatzgebiete des Cmdlets `ForEach-Object`:

- 1) Es werden drei Zahlen an das Cmdlet `ForEach-Object` weitergereicht, um mit den unterschiedlichen Werten je ein Echoanforderungspaket („Ping“) an verschiedene IP-Adressen zu senden:

```
1..3 | ForEach-Object -Process { Test-Connection 127.0.0.$_ -Count 1 }
```

- ✓ Da auf der linken Seite der Pipeline drei Zahlenwerte (1, 2, 3) vorhanden sind, wird das Cmdlet `ForEach-Object` drei Mal durchlaufen.
- ✓ Im Skriptblock, der den Wert für den Parameter `-Process` bildet, wird der Befehl mit jeweils nur einem Paket ausgeführt. Als Zieladresse werden die IP-Adressen 127.0.0.1, 127.0.0.2 und 127.0.0.3 verwendet. Im ersten Schleifendurchlauf repräsentiert `$_` das erste übergebene Objekt (also 1), im zweiten Durchlauf 2, im letzten schließlich 3.

- 2) Beispiel 1) wird erweitert und mit Aktionen vor und nach der eigentlichen Ausführung des Cmdlets versehen:

```
1..3 | ForEach-Object -Begin { Clear-Host; Write-Host "Ich beginne zu
                                pingen :" }
          -Process { Test-Connection 127.0.0.$_ -Count 1 }
          -End { Write-Host "Aktion beendet." }
```

- ✓ Hinweis: Die eine lange Eingabezeile wurde aus Gründen der Übersichtlichkeit umformatiert.
- ✓ Es wurden Aktionen zu Beginn und Ende hinzugefügt.
- ✓ Im Parameter `-Begin` sind mehrere Cmdlets zu finden. Als Trennzeichen zwischen den Cmdlets fungiert das Semikolon.

- 3) Mit der PowerShell können Sie auch Ihren Rechner herunterfahren. Das Cmdlet `Stop-Computer` – ohne weitere Parameter verwendet – führt dazu, dass Ihr lokaler Rechner herunterfährt. Aber als Administrator können Sie das Cmdlet auch anders einsetzen:

```
"Client1", "Server2", "Server3", "Client7" |
    ForEach-Object -Process { Stop-Computer -ComputerName $_ -WhatIf }
```

- ✓ Es liegt eine Anzahl von Computernamen aus dem lokalen Netzwerk vor. Die Namen können ebenso zeilenweise in einer Textdatei stehen. In dem Fall lesen Sie die Textdatei (im aktuellen Verzeichnis) mit dem Befehl `Get-Content .\dateiname.txt` ein.
- ✓ Auf jedes übergebene Objekt, also auf jeden Computernamen, wird ein Skriptblock angewendet. Dies ist das Cmdlet `Stop-Computer` mit dem Parameter `-ComputerName`. Folglich fährt jeder Rechner herunter, dessen Name als Wert (`$_`) an den genannten Parameter übergeben wird.
- ✓ Um das Beispiel zu entschärfen, wird zusätzlich der allgemeine Parameter `-WhatIf` verwendet, durch den eine Ausführung des Cmdlets nur simuliert wird.

### Group-Object – Eigenschaften mit gleichen Werten gruppieren

Das Cmdlet `Group-Object` dient zur Analyse von Häufigkeiten. Objekte werden auf Basis des Werts einer angegebenen Eigenschaft gruppiert. Standardmäßig erhalten Sie als Rückgabe eine Tabelle mit Angaben zur Häufigkeit, dem gruppierten Wert und den Mitgliedern der Gruppierung (Einzelwerten).

Geben Sie mehrere Eigenschaften an, wird zuerst nach der ersten Eigenschaft gruppiert und dann weiter verschachtelt, also Untergruppen für die weiteren Eigenschaften innerhalb der Hauptgruppierung gebildet.

Für das Cmdlet stehen u. a. folgende Parameter zur Verfügung:

Group-Object		
Parameter	Typ	Beschreibung
<code>-Property</code>	P (1)	Gibt die Eigenschaft für die Gruppierung an. Die Objekte werden auf Basis der Werte der Eigenschaft gruppiert.
<code>-NoElement</code>	S	Gibt die Einzelwerte nicht mit aus, sondern nur Anzahl und Wert, nach dem gruppiert wird
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: <code>Get-Help -Name Group-Object -Full</code>

In den folgenden praktischen Beispielen sehen Sie Anwendungsmöglichkeiten für das Cmdlet `Group-Object`:

- 1) Sie wollen sich informieren, welche Dateiendungen wie häufig im Windows-Verzeichnis zu finden sind:

```
Get-ChildItem C:\Windows -File | Group-Object -Property Extension
```

- ✓ `Get-ChildItem` mit dem Parameter `-File` liest alle Dateien im angegebenen Verzeichnis aus.
- ✓ Im zweiten Schritt werden die Dateien nach ihrer Endung gruppiert. Sie sehen in der Ausgabe alle vorkommenden Dateiendungen, die Häufigkeit ihres Vorkommens und die ersten vier Elemente der Gruppierungen (soweit es die Fensterbreite zulässt).

Sollten Sie nicht wissen, nach welchen Eigenschaften Sie hier im Beispiel gruppieren können, helfen Ihnen zum einen die Autovervollständigung in der PowerShell-Konsole sowie das IntelliSense in PowerShell ISE. Zum anderen erinnern Sie sich an das Cmdlet `Get-Member`, das die zur Verfügung stehenden Eigenschaften und Methoden anzeigt (z. B.: `Get-ChildItem C:\Windows -File | Get-Member`).



- 2) Sie wollen herausfinden, welche Verben in welcher Häufigkeit in allen Arten von PowerShell-Befehlen vorkommen:

```
Get-Command | Group-Object -Property Verb -NoElement | Sort-Object -Property Count
```

- ✓ Get-Command liefert die eine Liste der PowerShell-Befehle.
- ✓ Mit Group-Object werden sie nach ihrem Verb gruppiert. Auf die Ausgabe der Einzelemente wird verzichtet.
- ✓ Letztlich sorgt Sort-Object für eine Sortierung nach der Häufigkeit der vorkommenden Verben. Oder genauer: Group-Object zeigt eine Tabelle mit den Eigenschaften *Count* (Häufigkeit des Vorkommens) und *Name* (Wert der Eigenschaft *Verb*) an. Es erfolgt eine Sortierung aufsteigend nach dem Zahlwert der Eigenschaft *Count*.

### Measure-Object – Statistiken für die Objekte

Das Cmdlet Measure-Object berechnet numerische Eigenschaften bestimmter Objekttypen. Das Cmdlet kann Objekte zählen sowie Minimum, Maximum, Mittelwert und Summe numerischer Werte berechnen. Bei Textobjekten werden die Anzahl von Textzeilen, Wörtern und Zeichen gezählt.

Für das Cmdlet stehen u. a. folgende Parameter zur Verfügung:

Measure-Object		
Parameter	Typ	Beschreibung
-Property	P (1)	Gibt die Eigenschaft(en) für die Messung an. Ohne weitere Parameter wird die <i>Count</i> -Eigenschaft des Objekts angezeigt.
-Average	S	Für numerische Objekte: zeigt den Mittelwert der angegebenen Eigenschaft(en) an
-Character	S	Für Textobjekte: zählt die Anzahl der Zeichen
-IgnoreWhiteSpace	S	Für Textobjekte: Leerzeichen werden bei der Zählung von Wörtern und Zeichen ignoriert.
-Line	S	Für Textobjekte: zählt die Anzahl der Zeilen
-Maximum	S	Für numerische Objekte: zeigt den größten Wert der angegebenen Eigenschaft(en) an
-Minimum	S	Für numerische Objekte: zeigt den kleinsten Wert der angegebenen Eigenschaft(en) an
-Sum	S	Für numerische Objekte: zeigt die Summe der Werte der angegebenen Eigenschaft(en) an
-Word	S	Für Textobjekte: zählt die Anzahl der Wörter
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help -Name Measure-Object -Full

Die folgenden Beispiele zeigen das Cmdlet im Praxiseinsatz:

- 1) Sie möchten Statistikwerte für eine Zahlenreihe ermitteln:

```
7, 3, 5, 22, -5, 4, 11, 17, 2 | Measure-Object -Average -Sum -Maximum -Minimum
```

- ✓ Numerische Werte erhalten Sie im Beispiel aus einer Eingabe. Mithilfe von Measure-Object und den entsprechenden Switch-Parametern werden nicht nur die Anzahl der numerischen Werte, sondern auch Mittelwert, Summe, Maximal- und Minimalwert ausgegeben.

2) Sie möchten die Statistiken einer Textdatei auslesen:

```
Get-Content -Path .\beispiel.txt | Measure-Object -Line -Word -Character
```

- ✓ Über die Pipeline wird der Inhalt einer Textdatei übergeben und statistisch ausgewertet. In diesem Fall werden Zeilen, Wörter und Zeichen gezählt.
- ✓ Sollten Sie in diesem Beispiel einen Switch-Parameter für numerische Objekte einsetzen, weisen Fehlermeldungen auf den unpassenden Objekttyp hin.

3) Sie möchten erfahren, wie groß die Dateien im Windows-Verzeichnis sind:

```
Get-ChildItem -Path C:\Windows | Measure-Object -Property Length -Sum
```

- ✓ Im ersten Cmdlet erfolgt der Zugriff auf die Dateien im Windows-Verzeichnis. Measure-Object zeigt für die übergebenen Objekte die Summe der Eigenschaft *Length* (Dateigröße) an.

### Tee-Object – Variable und Ausgabe

Das Cmdlet Tee-Object ist ein Sonderfall. Während ein Objekt normalerweise entweder angezeigt oder verarbeitet wird, sendet Tee-Object die Ausgabe eines Befehls in beide Richtungen. Zum einen erfolgt die Speicherung in einer Variablen oder einer Datei, zum anderen erfolgt die Ausgabe in der Konsole bzw. Weitergabe über die Pipeline.

Für dieses Cmdlet stehen u. a. folgende Parameter zur Verfügung:

Tee-Object		
Parameter	Typ	Beschreibung
-FilePath	P (1)	Speichert das Objekt in der angegebenen Datei
-Variable	N	Das Objekt wird in der angegebenen Variablen gespeichert. Als Wert ist hier nur der Name erwartet, nicht das übliche führende \$, das Variablen in vielen Programmiersprachen kennzeichnet.
-Append	S	Wird der Parameter angegeben, werden Daten an eine Datei angehängt. Ohne diesen Parameter wird eine bestehende Datei ersetzt.
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help -Name Tee-Object -Full

Zwei Beispiele zeigen das Cmdlet im Einsatz:

1) Alle Prozesse auf Ihrem Rechner sollen ausgegeben und zusätzlich in einer Datei gespeichert werden:

```
Get-Process | Tee-Object -FilePath .\beispiel.txt
```

- ✓ Die Liste der Prozesse wird in der Datei *beispiel.txt* im aktuellen Verzeichnis gespeichert. Tee-Object ist das letzte Cmdlet in der Pipeline, damit erfolgt eine Ausgabe und keine Weiterleitung an ein folgendes Cmdlet.

- 2) Sie wollen das aktuelle Datum und die aktuelle Uhrzeit in einer Variablen speichern und zusätzlich die Werte einiger Eigenschaften in der Konsole anzeigen.

```
Get-Date | Tee-Object -Variable Beispiel | Select-Object -Property Month, DayOfWeek
```

- ✓ Tee-Object sorgt dafür, dass das übergebene Objekt in der Variablen \$Beispiel gespeichert wird. Da das Cmdlet in diesem Beispiel nicht am Ende der Pipeline steht, wird das verarbeitete Objekt an das folgende Cmdlet weitergegeben.
- ✓ Achten Sie darauf, dass der Variablenname als Wert der Eigenschaft -Variable ohne das führende \$ einzugeben ist.
- ✓ Durch Eingabe von \$Beispiel in der Konsole (nach Ausführung der obigen Beispielzeile) können Sie sich den Inhalt der Variablen anzeigen lassen.
- ✓ Select-Object zeigt die Werte der beiden ausgewählten Eigenschaften in der Konsole an.



### 3.3 Formatierung und Ausgabe

Wenn die gewünschten Daten die Pipeline durchlaufen haben, bleibt als optionaler letzter Arbeitsschritt die Aufarbeitung der Daten. Sie können die Daten nach Ihren Wünschen für die Ausgabe formatieren oder die Ausgabe in eine Datei umleiten. Zudem stehen Ihnen Werkzeuge zur Verfügung, die Daten in verschiedene Formate zu konvertieren.

Für diese Aufgaben stehen Ihnen Cmdlet-Familien zur Verfügung, die an ihrem Verb erkennbar sind: *Out* (für Ausgabe), *Export* (für den Export) und *ConvertTo* (für die Konvertierung in ein anderes Format).

#### Out-Cmdlets – Ausgabe organisieren

Die PowerShell stellt einige Out-Cmdlets zur Verfügung, damit Sie ein Ausgabeziel gemäß Ihren Wünschen definieren können:

Cmdlet	Kurzbeschreibung
Out-Default	Ein PowerShell-internes Standardformatierungsprogramm (ETS; extended type system) sorgt für eine automatische, typabhängige Formatierung vorliegender Daten. In den meisten Fällen erfolgt dies als Tabelle mit automatisch formatierten Spalten. Danach wird die Ausgabe an das Standard-Ausgabe-Cmdlet gesendet.
Out-File	Die Ausgabe wird an eine Datei gesendet.
Out-GridView	Die Ausgabe erfolgt in einer interaktiven Tabelle in einem eigenen Fenster.
Out-Host	Standard-Ausgabe-Cmdlet, das die Ausgabe zur Anzeige an die Befehlszeile des PowerShell-Hosts sendet
Out-Null	Die Ausgabe wird gelöscht. Es wird nichts angezeigt.
Out-Printer	Sendet die Ausgabe an einen Drucker
Out-String	Wandelt die Objekte einer Ausgabe in Text um

Während Out-Default, Out-Null und Out-String an dieser Stelle keiner weiteren Erläuterung bedürfen, sehen Sie in den folgenden Abschnitten Detailinformationen zu den anderen Out-Cmdlets.

### Out-File – In Datei umleiten

Das Cmdlet Out-File leitet die Ausgabe in eine Datei um. In der Funktion entspricht das dem seit DOS bekannten Umleitungsparameter „>“. Das Cmdlet lässt sich durch seine optionalen Parameter fein steuern:

Out-File		
Parameter	Typ	Beschreibung
-FilePath	P (1)	Angabe des Pfads der Ausgabedatei
-Encoding	P (2)	Festlegung der Zeichencodierung für die Ausgabedatei. Standardwert ist <i>Unicode</i> .
-Append	S	Fügt die Ausgabe einer bereits vorhandenen Datei hinzu, anstatt deren Inhalt zu ersetzen
-Force	S	Eine bereits vorhandene Datei wird überschrieben, auch wenn sie schreibgeschützt ist.
-NoClobber	S	Verhindert, dass eine bereits vorhandene Datei überschrieben wird
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help -Name Out-File -Full

Das folgende Beispiel zeigt die Umleitung der Ausgabe in eine Datei:

```
Get-Process | Out-File -FilePath C:\Daten\prozesse.txt -NoClobber
```

- ✓ Die Liste der Prozesse des lokalen Rechners wird durch das Cmdlet Out-File in der angegebenen Datei gespeichert. Das Verzeichnis C:\Daten muss bereits vorhanden sein.
- ✓ Der Parameter -NoClobber sorgt dafür, dass eine bereits vorhandene Datei nicht überschrieben wird. Führen Sie die Beispielbefehlszeile zweimal hintereinander aus, erhalten Sie aus diesem Grund eine Fehlermeldung.

### Out-GridView – Ausgabe im eigenen Windows-Fenster

Das Cmdlet Out-GridView sorgt dafür, dass die Ausgabe in einem separaten Fenster angezeigt wird. Dort können interaktive Aktionen wie Sortierung und Filterung der Daten vorgenommen werden.

Für dieses Cmdlet stehen u. a. folgende Parameter zur Verfügung:

Out-GridView-Object		
Parameter	Typ	Beschreibung
-Title	N	Legt einen Text fest, der als Titel in der Titelleiste des Fensters erscheint
-PassThru	S	Legt fest, dass das Ergebnis des Cmdlets in der Pipeline an den folgenden Befehl weitergereicht wird. In der Standardeinstellung generiert das Cmdlet keine Ausgabe, die weiterverwendet werden kann.
-Wait	S	Unterdrückt im aufrufenden PowerShell-Host den Prompt, bis das Fenster des Cmdlets geschlossen wird. Gedacht ist das Verhalten primär für die Ausführung eines PowerShell-Befehls direkt aus Windows, damit das Ergebnis nicht sofort geschlossen wird.
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help -Name Out-GridView -Full

Die beiden folgenden Beispiele zeigen die Verwendung des Cmdlets:

- 1) Anzeige der Liste der Prozesse im eigenen Fenster:

```
Get-Process | Out-GridView
```

- ✓ Die Prozesse werden in einem separaten Fenster geöffnet, in dem sie sortiert oder gefiltert werden können.

- 2) Erweitertes Beispiel 1), zusätzlich wird ein Dateiexport vorgenommen:

```
Get-Process | Out-GridView -PassThru | Out-File -FilePath .\prozesse.txt
```

- ✓ Durch den Parameter –PassThru wird das Cmdlet Out-GridView angewiesen, die im Fenster vorgenommenen Auswahlen in der Pipeline als Ergebnis an das folgende Cmdlet weiterzugeben (Speichern der Auswahl in eine Textdatei im aktuellen Verzeichnis).

### **Out-Host – Standardausgabe**

Das Cmdlet Out-Host ist das Standard-Ausgabe-Cmdlet. Eine Angabe ist nur dann vonnöten, wenn Sie das Verhalten über Parameter steuern wollen.

Für dieses Cmdlet steht u. a. folgender Parameter zur Verfügung:

Out-Host		
Parameter	Typ	Beschreibung
-Paging	S	Zeigt eine Seite der Ausgabe an und wartet auf eine Benutzereingabe, um jeweils eine weitere Seite anzuzeigen
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help -Name Out-Host -Full

Das Cmdlet erklärt sich von selbst. Eine Angabe ist nur dann sinnvoll, wenn Sie die Ausgabe seitenweise wünschen:

```
Get-ChildItem -Path C:\Windows\System32 | Out-Host -Paging
```

- ✓ Die Ausgabe des Verzeichnisinhalts wird seitenweise auf dem Bildschirm vorgenommen.
- ✓ Am Fuß einer Ausgabeseite wird angezeigt, durch welche Tasten Sie das weitere Vorgehen steuern können ("<LEERTASTE> Nächste Seite, <WAGENRÜCKLAUF> Nächste Zeile, Q Beenden").

### **Out-Printer – Ausgabe zu einem Drucker senden**

Out-Printer sendet die Ausgabe an einen Drucker. Ist kein Parameter angegeben, wird der Standarddrucker angesteuert. Eine feine Steuerung des Druckers, wie z. B. Seitenausrichtung, Schriftart und -größe, ist mit diesem Cmdlet nicht möglich.

Für dieses Cmdlet steht u. a. folgender Parameter zur Verfügung:

Out-Printer		
Parameter	Typ	Beschreibung
-Name	P (1)	Als Wert wird der Name des (alternativen) Druckers erwartet, auf dem die Ausgabe gedruckt werden soll.
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help -Name Out-Printer -Full

Wollen Sie die Ausgabe an einen speziellen Drucker senden, können Sie über die folgende Zeile die Namen der installierten Drucker herausfinden:

```
Get-WmiObject -Class Win32_Printer | Select-Object -Property Name
```

- ✓ Sie erhalten über WMI (Windows Management Instrumentation) Zugriff auf installierte Drucker. Es erfolgt die Ausgabe der Namen der Drucker.

Den gewünschten Namen können Sie mit dem Cmdlet `Out-Printer` als Wert des Parameters `-Name` verwenden und so einen speziellen Drucker ansteuern:

```
Get-Process | Out-Printer -Name "Microsoft XPS Document Writer"
```

- ✓ Sie wählen für die Ausgabe einen speziellen Druckernamen, der als Text in Anführungszeichen anzugeben ist. Die Anführungszeichen müssen verwendet werden, wenn sich im Namen des Druckers ein Leerzeichen befindet.

## Export-Cmdlets – Export von Objekten

Einige Export-Cmdlets haben andere Aufgaben, als für den Export von Pipelineergebnissen zu sorgen, aber zwei dieser Cmdlets erfüllen genau diesen Zweck. Es handelt sich um folgende Cmdlets:

Cmdlet	Kurzbeschreibung
Export-Clixml	Export von Objekten in XML-Format
Export-Csv	Umwandlung der Daten in kommaseparierte Textdateien

Beide Cmdlets schreiben die konvertierten Daten in externe Dateien.

### Export-Clixml – Export in XML-Format

`Export-Clixml` wandelt die Ausgabe in XML-Format und speichert sie in einer Datei dieses Typs.

Achten Sie auf eine Einschränkung der ausgewählten Daten, da XML-Dateien sehr groß werden können. Bei einem Export aller möglichen Eigenschaften aller Prozesse auf dem lokalen Rechner ergab z. B. eine Speicherung (ohne weitere manuelle Einschränkung der Auswahl) im Test eine Dateigröße von über 10 MB.



Für dieses Cmdlet stehen u. a. folgende Parameter zur Verfügung:

Export-Clixml		
Parameter	Typ	Beschreibung
<code>-Path</code>	P (1)	Gibt den Dateipfad der XML-Exportdatei an
<code>-Depth</code>	N	Angabe, wie viele Objektebenen in die Konvertierung einbezogen werden (Standard: 2)
<code>-Encoding</code>	N	Gibt die Codierung der XML-Datei an (Standard: UTF8)
<code>-Force</code>	S	Erzwingt den Export, auch wenn die Zielfile schreibgeschützt ist
<code>-NoClobber</code>	S	Verhindert das Überschreiben einer bereits vorhandenen Zielfile
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: <code>Get-Help -Name Export-Clixml -Full</code>



Liegen XML-Daten bereits vor, sorgt das Cmdlet `Import-Clixml` dafür, dass Sie diese Daten als Objekte in die PowerShell importieren können.

Das folgende Beispiel zeigt eine Anwendungsmöglichkeit für das Cmdlet `Export-Clixml`:

- 1) Sie möchten Angaben zu einem ausgewählten Prozess in eine XML-Datei exportieren:

```
Get-Process -Name powershell | Export-Clixml -Path .\ps-prozess.xml
```

- ✓ Der Prozess *powershell* wird ausgewählt und über die Pipeline gegeben.
- ✓ Es findet ein Export in die Datei *ps-prozess.xml* im aktuellen Verzeichnis statt.

### Export-Csv – Export in CSV-Format

`Export-Csv` wandelt die Ausgabe in CSV-Format und speichert sie in einer Datei dieses Typs.

Für dieses Cmdlet stehen u. a. folgende Parameter zur Verfügung:

Export-Csv		
Parameter	Typ	Beschreibung
<code>-Path</code>	P (1)	Gibt den Dateipfad der CSV-Exportdatei an
<code>-Delimiter</code>	P (2)	Trennzeichen der Eigenschaftswerte. Standardwert ist das Komma, eigene Werte müssen in Anführungszeichen stehen.
<code>-Encoding</code>	N	Gibt die Codierung der CSV-Datei an (Standard: ASCII)
<code>-Force</code>	S	Erzwingt den Export, auch wenn die Zielfile schreibgeschützt ist
<code>-NoClobber</code>	S	Verhindert das Überschreiben einer bereits vorhandenen Zielfile
<code>-NoTypeInformation</code>	S	Die üblicherweise als erste Zeile eingefügte Typbezeichnung „#TYPE...“ wird weggelassen.
<code>-UseCulture</code>	S	Das Trennzeichen der Eigenschaftswerte für die aktuelle Kultur wird verwendet. Der Standardwert ist das Komma, das in Deutschland übliche Trennzeichen ist das Semikolon.
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: <code>Get-Help -Name Export-Csv -Full</code>



Liegen Daten bereits im CSV-Format vor (z. B. durch Speicherung in diesen Dateityp in Microsoft Excel), sorgt das Cmdlet `Import-Csv` für einen Import dieser Daten in die PowerShell. Besonders der auch für dieses Cmdlet verfügbare Parameter `-UseCulture` hilft bei deutschen Spracheinstellungen sehr, da die Daten mit dem Semikolon anstelle des Kommas getrennt wurden.

Den Umgang mit diesem Cmdlet sehen Sie in den folgenden Beispielen:

- 1) Sie möchten ausgewählte Eigenschaften eines Prozesses in eine CSV-Datei exportieren:

```
Get-Process -Name powershell | Select-Object Company, Description, Id, StartTime | Export-Csv -Path ps-prozess.csv
```

- ✓ Zuerst erfolgt die Auswahl des Beispielprozesses *powershell*.
- ✓ Im zweiten Schritt wird nach gewünschten Eigenschaften gefiltert.
- ✓ Letztlich erfolgt der Export in eine CSV-Datei.

2) In leicht abgewandelten Beispielen kommen die Parameter des Cmdlets zum Einsatz:

```
Get-Process | Export-Csv -Path .\prozesse.csv -Delimiter ";"  
-NoTypeInformation
```

- ✓ Beim Export findet der Parameter `-Delimiter` Verwendung, der im Beispiel als Trennzeichen der Daten das Semikolon festlegt.
- ✓ Durch den Parameter `-NoTypeInformation` wird in der Zielfile keine CSV-Typbezeichnung eingetragen.

```
Get-Process | Export-Csv -Path .\prozesse.csv -UseCulture
```

- ✓ In der Exportdatei wird das Listentrennzeichen gemäß den Regeln der verwendeten Kultur verwendet. Die aktuellen Einstellungen können Sie mithilfe des Cmdlets `Get-Culture` herausfinden.

## ConvertTo-Cmdlets – Umwandlung von Objekten

ConvertTo-Cmdlets funktionieren ähnlich wie Export-Cmdlets. Daten werden ebenfalls umgewandelt, aber nicht in externe Dateien geschrieben.

Folgende Cmdlets stehen zur Verfügung:

Cmdlet	Kurzbeschreibung
ConvertTo-Csv	Konvertiert Objekte in kommasseparierte Werte um. Anders als <code>Export-Csv</code> speichert das Cmdlet die Werte nicht direkt in einer Datei.
ConvertTo-Html	Umwandlung von Objekten in HTML-Inhalt.
ConvertTo-SecureString	Konvertierung von Objekten in verschlüsselte Zeichenfolgen
ConvertTo-Xml	Wandelt Objekte in XML um

Sie lernen diese Cmdlets mit ihren wichtigsten Parametern und anhand einiger Beispiele im folgenden Abschnitt genauer kennen.

### ConvertTo-Csv – Umwandlung in CSV-Format

`ConvertTo-Csv` wandelt die Ausgabe in CSV-Format um.

Für dieses Cmdlet stehen u. a. folgende Parameter zur Verfügung:

ConvertTo-Csv		
Parameter	Typ	Beschreibung
<code>-InputObject</code>	P (1)	Gibt die Objekte an, die in CSV-Zeichenfolgen umgewandelt werden sollen. Alternativ erhält das Cmdlet die umzuwandelnden Daten direkt über die Pipeline.
<code>-Delimiter</code>	P (2)	Trennzeichen der Eigenschaftswerte. Standardwert ist das Komma, eigene Werte müssen in Anführungszeichen stehen.
<code>-NoTypeInformation</code>	S	Die üblicherweise als erste Zeile eingefügte Typbezeichnung „#TYPE...“ wird weggelassen.
<code>-UseCulture</code>	S	Das Trennzeichen der Eigenschaftswerte für die aktuelle Kultur wird verwendet. Der Standardwert ist das Komma, das in Deutschland übliche Trennzeichen ist das Semikolon.
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: <code>Get-Help -Name ConvertTo-Csv -Full</code>

Die Daten werden in das CSV-Format umgewandelt und stehen weiterhin für eine Bearbeitung in der PowerShell zur Verfügung. Die folgenden Beispiele zeigen den Umgang mit dem Cmdlet:

- 1) Die Eigenschaften eines laufenden Prozesses soll in das CSV-Format umgewandelt werden:

```
Get-Process -Name powershell | ConvertTo-Csv
```

- ✓ Als Eingabeobjekt für das Cmdlet ConvertTo-Csv dient das Objekt, das über die Pipeline gesendet wird. Da das Cmdlet der letzte Befehl in der Pipeline ist, werden alle Daten in der Konsole ausgegeben.

- 2) Sie wollen die Informationen, die das Cmdlet Get-Date liefert, in das CSV-Format umwandeln:

```
ConvertTo-Csv -InputObject (Get-Date) -NoTypeInformation
```

- ✓ Da es keine Daten gibt, die über die Pipeline gesendet werden können, benötigt ConvertTo-Csv ein Eingabeobjekt, das als Wert des Parameters -InputObject angegeben wird. Das ist in diesem Fall das Cmdlet Get-Date, das in runde Klammern eingeschlossen sein muss. Die Klammern sorgen dafür, dass das Cmdlet zuerst ausgewertet wird und das Ergebnis als Wert des Parameters fungiert. (Fehlen die Klammern, erhalten Sie nur die Auswertung einer achtbuchstabigen Zeichenfolge "Get-Date".)
- ✓ -NoTypeInformation unterdrückt die Ausgabe der Typinformationen, so dass in der Konsole zwei Zeilen ausgegeben werden: Überschriften und dazugehörige Werte.

### **ConvertTo-Html – Umwandlung in HTML-Format**

ConvertTo-Html wandelt die Ausgabe in das HTML-Format um, speichert sie aber nicht. Die Speicherung in eine Datei kann über ein nachfolgendes Cmdlet in der Pipeline erfolgen.

Für dieses Cmdlet stehen u. a. folgende Parameter zur Verfügung:

<b>ConvertTo-Html</b>		
Parameter	Typ	Beschreibung
-Property	P (1)	Angegebene Objekte werden mit in HTML einbezogen.
-Head	P (2)	Angaben, die im <head>-Bereich der HTML-Datei stehen sollen. Wird dieser Parameter verwendet, wird der Parameter -Title ignoriert.
-Title	P (3)	Angabe eines Titels für die HTML-Datei (entspricht Angabe zwischen <title>...</title>-Tags)
-Body	P (4)	Definiert den Inhalt des <body>-Tags der HTML-Datei
-As	N	Legt fest, ob die konvertierten Objekte in der HTML-Datei als Tabelle oder als Liste dargestellt werden sollen. Mögliche Werte sind <i>Table</i> und <i>List</i> , wobei <i>Table</i> der Standardwert ist.
-CssUri	N	Gibt den Pfad zur CSS-Datei an, das auf die HTML-Datei angewendet werden soll
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help -Name ConvertTo-Html -Full

Sie können beliebige Ausgaben mit einfachen Mitteln in einer HTML-Datei umwandeln und somit einem breiteren Personenkreis zugänglich machen. Die folgenden Beispiele sollen dies verdeutlichen:

1) Eine Liste laufender Prozesse soll als HTML-Datei vorgelegt werden:

```
Get-Process | ConvertTo-HTML -Property Name, Path, FileVersion -Body
"

# Übersicht über die laufenden Prozesse

" | Set-Content -Path
.\prozesse.html
```

- ✓ Im ersten Schritt werden die laufenden Prozesse abgefragt und über die Pipeline an das folgende Cmdlet gesendet.
- ✓ ConvertTo-HTML konvertiert drei ausgesuchte Eigenschaften der Prozessliste in HTML-Format.
- ✓ Das letzte Cmdlet der Pipeline Set-Content verarbeitet die HTML-Ausgabe weiter und schreibt sie in die Datei prozesse.html im aktuellen Verzeichnis.
- ✓ Wollen Sie die Ausgabe überprüfen und die Datei in Ihrem Standardbrowser ansehen, reicht bereits folgende Eingabe aus:

```
Invoke-Item -Path .\prozesse.html
```

- ✓ Invoke-Item ist ein Cmdlet, das die Standardaktion für ein angegebenes Element ausführt. Hier handelt es sich um eine Datei des Typs HTML. Als Windows-Standardaktion ist definiert, dass der Standardbrowser die angegebene Datei öffnet.

2) Angaben zu Diensten, die mit dem Buchstaben *d* beginnen, sollen in HTML-Format konvertiert werden:

```
Get-Service -Name d* | ConvertTo-HTML -As List
```

- ✓ Im ersten Teil der Pipeline werden Daten gesammelt, die den geforderten Kriterien entsprechen.
- ✓ Die Daten werden an ConvertTo-HTML übergeben und in Form einer Liste präsentiert.

### ConvertTo-SecureString – Umwandlung in eine verschlüsselte Zeichenfolge

ConvertTo-SecureString wandelt die Ausgabe in eine verschlüsselte Zeichenfolge um.

Für dieses Cmdlet stehen u. a. folgende Parameter zur Verfügung:

ConvertTo-SecureString		
Parameter	Typ	Beschreibung
-String	P (1)	Zeichenfolge für die Umwandlung in eine sichere Zeichenfolge
-Force	P (3)/S	Zustimmung, dass Sie die Auswirkungen des Parameters -AsPlainText kennen
-AsPlainText	S	Erzwingt die Verwendung von Klartext, obwohl normalerweise verschlüsselter Text verwendet wird. Wenn der Parameter verwendet werden soll, muss zusätzlich der Parameter -Force angegeben werden.
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help -Name ConvertTo-SecureString -Full

Folgende Beispiele verdeutlichen die Funktion des Cmdlets:

1) Eine angegebene Zeichenkette soll in eine sichere Zeichenkette umgewandelt werden:

```
ConvertTo-SecureString -String "Hallo!" -AsPlainText -Force
```

2) Eine Zeichenkette, die vom Benutzer einzugeben ist, soll in eine sichere Zeichenkette umgewandelt werden:

```
ConvertTo-SecureString -String (Read-Host [-AsSecureString]) -AsPlainText -Force
```

- ✓ In runden Klammern wird das Cmdlet Read-Host verwendet, das auf eine Benutzereingabe wartet. Verwenden Sie den dort optional angegebenen Parameter nicht, sehen Sie Ihre Eingabe im Klartext, anderenfalls als Folge von Sternchen.
- ✓ Das Ergebnis von Read-Host wird vom Cmdlet ConvertTo-SecureString verarbeitet. Die Ausgabe der PowerShell bestätigt die Umwandlung: System.Security.SecureString.
- ✓ Diese Beispiele zeigen nur die Konvertierung in eine sichere Zeichenfolge. In der Praxis ist ein solches Vorgehen am ehesten in einem Skript zu erwarten, welches das Ergebnis in einer Variablen abspeichert. Zu diesen Verfahren lesen Sie in späteren Kapiteln des Buchs.

### ConvertTo-Xml – Umwandlung in XML-Format

Das Cmdlet ConvertTo-Xml wandelt die Ausgabe in XML-Format um.

Für dieses Cmdlet stehen u. a. folgende Parameter zur Verfügung:

ConvertTo-Csv		
Parameter	Typ	Beschreibung
-InputObject	P (1)	Gibt das Objekt an, das umgewandelt werden soll. Alternativ erhält das Cmdlet das zu verarbeitende Objekt über die Pipeline.
-As	N	Bestimmung des Ausgabenformats. Mögliche Werte sind <i>String</i> , <i>Stream</i> oder <i>Document</i> . Standardwert ist <i>Stream</i> .
-NoTypeInformation	S	Die Typbezeichnung in den Objektknoten wird weggelassen.
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help -Name ConvertTo-Xml -Full

Das folgende Beispiel zeigt eine mögliche Anwendung des Cmdlets:

①	Get-Process -Name powershell   ConvertTo-Xml -As String   Set-Content -Path .\prozesse.xml
②	Get-Content -Path .\prozesse.xml

- ① In der ersten Zeile werden Informationen zum PowerShell-Prozess ausgelesen und die Ergebnisse in XML-Format umgewandelt. Schließlich wird das Endergebnis in eine Datei im aktuellen Verzeichnis gespeichert.
- ② Zur Überprüfung des Inhalts der XML-Datei wird der Inhalt der XML-Datei angezeigt.

## 3.4 Allgemein: Schrittweise Entwicklung einer Pipeline

Nachdem Sie häufige Cmdlets einer Pipeline kennengelernt und im Beispieleinsatz gesehen haben, können Sie ermessen, wie flexibel Sie mehrere dieser Cmdlets in einer Pipeline nacheinander einsetzen können, um zu Ihrem gewünschten Ergebnis zu gelangen.

Das zeigt ein häufig verwendetes Vorgehen in der PowerShell: Die Entwicklung des verwendeten Befehls entwickelt sich schrittweise mit den Anforderungen, die Sie an die Programmierung stellen. An einem einfachen Beispiel zum Auslesen von Informationen aus dem Dateisystem soll dies verdeutlicht werden:

Schritt	Aufgabe / Entwicklung der Programmzeile
1	Auslesen eines Verzeichnisinhalts: Get-ChildItem -Path C:\Windows
1a	Filtern, nur Dateien mit der Erweiterung .exe: Get-ChildItem -Path C:\Windows\*.exe
2	Nun sollen nur die Dateien betrachtet werden, die größer als 20 kB sind: Get-ChildItem -Path C:\Windows\*.exe   Where-Object { \$_.Length -gt 20kb }
3	Sie benötigen aber nur den Dateinamen und die Dateigröße: Get-ChildItem -Path C:\Windows\*.exe   Where-Object { \$_.Length -gt 20kb }   Select-Object -Property Length, Name
4	Die Ausgabe soll noch nach Dateigröße absteigend sortiert werden: Get-ChildItem -Path C:\Windows\*.exe   Where-Object { \$_.Length -gt 20kb }   Select-Object -Property Length, Name   Sort-Object -Property Length -Descending
5	Ihre Kollegen benötigen die Ausgabe im HTML-Format: Get-ChildItem -Path C:\Windows\*.exe   Where-Object { \$_.Length -gt 20kb }   Select-Object -Property Length, Name   Sort-Object -Property Length -Descending   ConvertTo-HTML -Body "<h1>Beispielüberschrift</h1>"
6	Abschließend wird das Ergebnis in eine Datei gespeichert: Get-ChildItem -Path C:\Windows\*.exe   Where-Object { \$_.Length -gt 20kb }   Select-Object -Property Length, Name   Sort-Object -Property Length -Descending   ConvertTo-HTML -Body "<h1>Beispielüberschrift</h1>"   Set-Content -Path .\daten_kollegen.html

Am Ausgangspunkt des Beispiels stand die Anzeige eines Verzeichnisinhalts mithilfe eines kurzen Cmdlets, am Ende steht eine lange, sperrig erscheinende Pipeline, die sich in mehreren Schritten entwickelt hat. Vergleichen Sie den Code der Arbeitsschritte 1 und 6 miteinander. Mit jeder Anforderung erfolgt eine Modifikation bzw. Ergänzung des Codes. Dennoch bleibt es für Sie durch die schrittweise Bearbeitung auf einem Schwierigkeits- und Komplexitätslevel.

## 3.5 Einsatztipps für die Pipeline

### So einfach wie möglich

Das Konzept der Pipeline mit der Übergabe kompletter Objekte ist sicherlich faszinierend. Wie Sie aber auch gelesen haben, bietet es sich manchmal aus Gründen der Geschwindigkeit an, die Arbeit mit Parametern der Cmdlets dem Einsatz der Pipeline z. B. beim Filtern von Daten vorzuziehen. Generell gilt: Was Sie mit einem Cmdlet erledigen können, erledigen Sie auch mit diesem Cmdlet. Die Pipeline hat ihre Berechtigung, wenn Sie komplexe Aufgaben erfüllen müssen, die über die Fähigkeiten einzelner Cmdlets hinausgehen.

Passen Sie Ihre Arbeit den Anforderungen an:

- ✓ Reicht ein Cmdlet für die Aufgabe aus, die Sie erledigen wollen, dann beschränken Sie sich auf das Cmdlet und dessen Parameter.
- ✓ Für eine komplexere Aufgabe setzen Sie mehrere Cmdlets ein, die über die Pipeline miteinander verbunden sind. Auch hier gilt: Verwenden Sie einzelne Cmdlets, wenn sich über sie – ohne Pipeline – Einzelaufgaben erledigen lassen.
- ✓ Wollen Sie Aufgaben automatisieren, speichern Sie die Befehle bzw. Befehlsfolgen in einem Skript zur späteren Verwendung (siehe spätere Kapitel).

### Pipeline zur Absicherung verwenden

Wenn Sie z. B. einen Prozess beenden wollen, verwenden Sie das Cmdlet `Stop-Process`. Falls Sie sich nicht ganz sicher sind, dass Sie den gewünschten Prozess adressieren, lesen Sie erst über ein `Get-Cmdlet` die gesuchte Information aus.

- a) `Stop-Process -Name powershell oder`
- b) `Get-Process -Name powershell`
- c) `Get-Process -Name powershell | Stop-Process`

Bedienen Sie sich immer der Methode b) + c), wenn Sie nicht ganz sicher sind. Methode a) allein beendet alle Prozesse mit Namen `powershell`. Gab es wirklich nur den einen, den Sie beenden wollen? Schauen Sie erst, welches Ergebnis Sie erhalten, und wenden Sie dann die beabsichtigte Aktion auf dieses Ergebnis an.

## 3.6 Kurz zusammengefasst

Mithilfe der Pipeline sind Sie in der Lage, komplexe Aufgaben zu erledigen. Die Pipeline arbeitet objektorientiert. Es wird also keine Zeichenfolge übergeben, sondern jeweils komplette Objekte, die mit all ihren Eigenschaften und Methoden dem nächsten Cmdlet zur Verfügung stehen.

Im ersten Schritt werden Informationen gesammelt, die in den optionalen weiteren Arbeitsschritten bearbeitet und den eigenen Wünschen angepasst werden. Im letzten optionalen Arbeitsschritt erfolgt dann die Umwandlung oder der Export des Ergebnisses – oder einfach die Ausgabe in der Konsole.

Setzen Sie die Cmdlets in der Pipeline sinnvoll ein. Filtern Sie z. B. Daten möglichst, bevor sie über die Pipeline an das nächste Cmdlet übergeben werden. Dies erhöht die Geschwindigkeit, da nur die gewünschten Daten gesammelt und weitergeleitet werden müssen.

## 3.7 Übung

### Arbeit mit der Pipeline

Übungsdatei: –

Ergebnisdatei: **3\_ergebnisse.txt**

1. Zeigen Sie alle Dienste an, die mit dem Buchstaben s beginnen, aber beendet (`stopped`) sind.
2. Sortieren Sie den Verzeichnisinhalt des Windows-Verzeichnisses (nur Dateien) absteigend nach Größe und exportieren das Ergebnis in die CSV-Datei `dateien.csv` im aktuellen Verzeichnis.
3. Es liegen drei Dateien vor: `prozesse1.csv`, `prozesse2.csv` und `prozesse3.csv`. Die Dateien wurden mit folgenden Befehlen erstellt:
  - 1- `Get-Process | Export-Csv -Path .\prozesse1.csv`
  - 2- `Get-Process | Export-Csv -Path .\prozesse2.csv -Delimiter ";"`
  - 3- `Get-Process | Export-Csv -Path .\prozesse3.csv -UseCulture`
  - a) Welche Unterschiede weisen die Dateien -1-, -2- und -3- auf?
  - b) Was ist der Unterschied zwischen den Dateien -2- und -3-?
4. Wie viele Dateien mit der Erweiterung `.exe` (extension) weist Ihr Windows-Verzeichnis auf?  
Wie groß sind diese Dateien insgesamt und durchschnittlich?
5. a) Rufen Sie – mit Mitteln, die in diesem Kapitel besprochen wurden – zehnmal die Anwendung `notepad.exe` auf.  
b) Lassen Sie sich die zehn laufenden Prozesse anzeigen und beenden sie dann.

# 4 Datenspeicher in der PowerShell



**Beispieldatei:** 4\_kompletter code.txt (Ordner Kapitel 04)

## 4.1 PowerShell-Provider

PowerShell-Provider (manchmal auch „Anbieter“ genannt) fassen Objektmengen zu einem speziellen Thema so zusammen, dass man durch sie wie durch Laufwerke navigieren kann.

Ein Provider legt fest, innerhalb welches Datenspeichers Sie sich bewegen können. `Get-ChildItem` z. B. ist das Standard-Cmdlet zur Ermittlung einer Liste von Daten in einem bestimmten Verzeichnis. PowerShell bietet einen Provider für das Dateisystem (namens `FileSystem`), somit bewegt sich `Get-ChildItem` bei Erstellung einer Dateiliste innerhalb des genannten Providers.

Nach ähnlichem Muster können Sie innerhalb der PowerShell allerdings nicht nur im Dateisystem agieren. PowerShell bietet Provider mit ähnlichen Möglichkeiten für den Anwender auch für andere Zwecke. Es gibt folgende wichtige Provider:

PowerShell-Provider	Funktion
Alias	Bietet einen Datenspeicher für alternative Kurzbefehle (vgl. Kapitel 5)
Environment	Datenspeicher für Umgebungsvariablen (vgl. Kapitel 6)
FileSystem	Datenspeicher für das Dateisystem (Laufwerke, Freigaben) (vgl. Abschnitt 4.2)
Function	Datenspeicher für Funktionen (vgl. Kapitel 8)
Registry	Datenspeicher für zwei wichtige Hauptschlüssel der Windows-Registry (vgl. Abschnitt 4.2)
Variable	Datenspeicher für Variablen (vgl. Kapitel 7)

All diese Provider bieten eigene Laufwerke an, mit denen Sie arbeiten können, wie Sie es innerhalb des Dateisystems gewohnt sind. Wie im Dateisystem können Sie durch diese Laufwerke navigieren, Objekte abfragen, erstellen und löschen u.v.m.

### Get-PSProvider – Informationen über Provider einholen

Sie können sich über Provider der aktuellen Sitzung mithilfe des Cmdlets `Get-PSProvider` informieren.

```
Get-PSProvider [-PSProvider]
```

Sie erhalten eine Ausgabe, die der folgenden Beispieldaten ähnelt:

Name	Capabilities	Drives
---	-----	-----
Alias	ShouldProcess	{Alias}
Environment	ShouldProcess	{Env}
FileSystem	Filter, ShouldProcess, Credentials	{C, D, E}
Function	ShouldProcess	{Function}
Registry	ShouldProcess, Transactions	{HKLM, HKCU}
Variable	ShouldProcess	{Variable}
Certificate	ShouldProcess	{Cert}
WSMan	Credentials	{WSMan}

Falls Sie den optionalen Parameter `-PSProvider` verwenden, wird der Name eines oder mehrerer Provider als Wert erwartet und damit die Ausgabe auf diese Provider beschränkt.

## 4.2 PowerShell-Laufwerke

### PowerShell-Laufwerke mit `Get-PSDrive` anzeigen

Die PowerShell bietet die Möglichkeit, auch mit Laufwerken zu arbeiten, die über das Dateisystem mit seinen lokalen und Netzlaufwerken hinausgehen. Dies mag auf den ersten Blick ungewöhnlich erscheinen, stellt aber einen effektiven Weg dar, auf Informationen außerhalb des Dateisystems mit bekannten Mitteln zuzugreifen.

Das Cmdlet `Get-PSDrive` ruft die PowerShell-Laufwerke in der aktuellen Sitzung ab:

```
Get-PSDrive
```

Sie erhalten eine Ausgabe, die der folgenden Beispieldaten ähnelt:

Name	Used (GB)	Free (GB)	Provider	Root
---	-----	-----	-----	----
Alias			Alias	
C	16,87	182,79	FileSystem	C:\
Users\AD				
Cert			Certificate	\
D	3,44	0,00	FileSystem	D:\
Daten			FileSystem	\Client1\Datens
E	413,92	586,69	FileSystem	\vboxsrv\Herd
Env			Environment	
Function			Function	
HKCU			Registry	HKEY_CURRENT_USER
HKLM			Registry	HKEY_LOCAL_MACHINE
Variable			Variable	
WSMan			WSMan	

Folgende Laufwerkstypen werden angezeigt:

- ✓ Alle logischen Windows-Laufwerke und Netzwerkfreigaben  
Diese gehören zum Provider `FileSystem` und haben im Beispiel die Namen *C*, *D*, und *E*. *C* ist die lokale Festplatte, *D* das DVD-Laufwerk und *E* eine Netzwerkfreigabe.
- ✓ Zwei Laufwerke der Windows-Registry  
Es handelt sich um die Laufwerke `HKCU`: (`HKEY_CURRENT_USER`) und `HKLM`: (`HKEY_LOCAL_MACHINE`) des Providers `Registry`.
- ✓ Von weiteren PowerShell-Providern zur Verfügung gestellte Laufwerke  
Im Beispiel sehen Sie die Laufwerke `Alias`: (Provider `Alias`), `Env`: (Provider `Environment`), `Function`: (Provider `Function`), `Variable`: (Provider `Variable`) und `WSMan`: (Provider `WSMan`).
- ✓ Weitere Laufwerke, die in der PowerShell mithilfe des Cmdlets `New-PSDrive` erstellt wurden  
Im Beispiel ist das auf diese Weise erstellte Laufwerk `Daten`: zu sehen (Hinweise zur Verwendung von `New-PSDrive` folgen später in diesem Kapitel).

Für das Cmdlet Get -PSDrive stehen u. a. folgende Parameter zur Verfügung:

Get-PSDrive		
Parameter	Typ	Beschreibung
-Name	P (1)	Die Ausgabe wird auf die angegebenen Laufwerke beschränkt. Als Wert wird der Name oder der Buchstabe des Laufwerks ohne Doppelpunkt erwartet.
-PSPrinter	N	Nur die vom angegebenen Provider unterstützten Laufwerke werden abgerufen.
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help -Name Get-PSDrive -Full

Die folgenden Beispiele verdeutlichen den Einsatz des Cmdlets:

- 1) Sie möchten Informationen zu einem Laufwerk namens *Daten* abrufen:

```
Get-PSDrive -Name Daten
```

- ✓ Durch die Verwendung des Parameters –Name begrenzen Sie die Ausgabe auf Informationen zum namentlich angegebenen Laufwerk.
- ✓ Ausführlichere Informationen als in der Standardausgabe der PowerShell erhalten Sie durch die Verwendung der Pipeline, indem Sie alle verfügbaren Eigenschaften anzeigen lassen:

```
Get-PSDrive -Name Daten | Select-Object -Property *
```

- 2) Sie möchten alle Laufwerke des Providers *FileSystem* auflisten:

```
Get-PSDrive -PSPrinter FileSystem
```

- ✓ Sie erreichen dies, indem Sie den Parameter –PSPrinter verwenden.

## Weitere PSDrive-Cmdlets

Neben dem Cmdlet Get -PSDrive, mit dem Sie die PowerShell-Laufwerke der aktuellen Sitzung anzeigen lassen können, kennt die PowerShell zwei weitere PSDrive-Cmdlets:

Cmdlet	Aufgabe des Cmdlets
New-PSDrive	Erstellt ein PowerShell-Laufwerk in der aktuellen Sitzung
Remove-PSDrive	Löscht ein PowerShell-Laufwerk, das mit New-PSDrive erstellt wurde

## New-PSDrive – neue Laufwerke anlegen

Mit dem Cmdlet New-PSDrive legen Sie eigene Laufwerke in der aktuellen PowerShell-Sitzung an. Es stehen u.a. folgende Parameter zur Verfügung:

Get-PSDrive		
Parameter	Typ	Beschreibung
-Name	P (1)	Angabe der Bezeichnung des neuen Laufwerks. Sie sind nicht an Laufwerksbuchstaben gebunden, sondern können eine beliebige Zeichenkette eingeben.
-PSPrinter	P (2)	Als Wert geben Sie den PowerShell-Provider an, der den beabsichtigten Laufwerkstyp unterstützt, z. B. <i>FileSystem</i> für ein Verzeichnis oder eine Netzwerkfreigabe.
-Root	P (3)	Gibt den Speicherort des neuen Laufwerks an. Bei einer Netzwerkfreigabe verwenden Sie einen UNC-Pfad der Form <code>\ Server\Freigabe</code> , für ein lokales Verzeichnis den absoluten Pfad (z. B. <code>C:\Daten\FuerAlle</code> ) oder einen Schlüssel in der Registry (wie <code>HKCU:\SOFTWARE\Microsoft</code> ). Sollte sich im absoluten Pfad ein Leerzeichen befinden, müssen Sie die komplette Zeichenkette in Anführungszeichen setzen (z. B. "C:\Programme (x86)\Mozilla Firefox").
-Description	N	Eingabe einer Kurzbeschreibung für das Laufwerk
-Persist	S	Erstellt ein dauerhaftes Laufwerk, das nicht an die Lebensdauer der PowerShell-Sitzung gebunden ist. Das Laufwerk wird von Windows verwaltet und kann in allen dafür vorgesehenen Systemprogrammen verwendet werden. Es existieren einige Bedingungen, an die Sie sich bei Verwendung des Parameters halten müssen: <ul style="list-style-type: none"><li>✓ Der Name eines solchen Laufwerks muss ein Laufwerksbuchstabe sein, eine Zeichenkette ist nicht erlaubt.</li><li>✓ Der im Parameter <code>-PSPrinter</code> angegebene Provider muss <i>FileSystem</i> lauten.</li><li>✓ Das im Parameter <code>-Root</code> spezifizierte Verzeichnis muss ein UNC-Pfad zu einem Remoterechner sein.</li></ul>
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: <code>Get-Help -Name New-PSDrive -Full</code>

Die folgenden Beispiele zeigen das Cmdlet und seine Parameter im Einsatz:

- 1) Sie möchten das Verzeichnis `C:\Daten` als Laufwerk *Daten* in der aktuellen PowerShell-Sitzung zur Verfügung stellen:

```
New-PSDrive -Name Daten -PSPrinter FileSystem -Root \\Fileserver1\Daten
```

- 2) Sie möchten das Verzeichnis aus dem obigen Beispiel dauerhaft und unabhängig von der PowerShell-Sitzung zur Verfügung stellen:

```
New-PSDrive -Name Z -PSPrinter FileSystem -Root \\Fileserver1\Daten  
-Persist
```

- ✓ Da der Parameter `-Persist` verwendet wird, ist zu beachten, dass der Name des neuen Laufwerks ein Buchstabe ist, eine Netzwerkfreigabe eines Remoterechners in UNC-Form verwendet wird und der Wert des Parameters `-PSPrinter` *FileSystem* ist.
- ✓ Das Laufwerk steht ab nun dauerhaft über das PowerShell-Sitzungsende hinaus zur Verfügung.

- 3) Sie möchten ein Laufwerk erstellen, das auf einen bestimmten Schlüssel in der Registry zeigt:

```
New-PSDrive -Name Microsoft -PSPrinter Registry -Root  
HKLM:\SOFTWARE\Microsoft
```

- ✓ Sie erhalten durch diesen Befehl ein Laufwerk `Microsoft`:, das direkt auf den angegebenen Schlüssel in der Registry zeigt.

### Remove-PSDrive – Laufwerke entfernen

Mit dem Cmdlet Remove-PSDrive entfernen Sie mit dem Cmdlet New-PSDrive angelegte Laufwerke. Auch die Zuordnungen zu Windows- oder Netzwerklaufwerken können mit diesem Cmdlet getrennt werden. Das momentan aktive Laufwerk lässt sich nur durch die Angabe eines zusätzlichen Parameters (-Force) entfernen.

Es stehen u. a. folgende Parameter zur Verfügung:

Remove-PSDrive		
Parameter	Typ	Beschreibung
-Name	P (1)	Angabe des Namens des Laufwerks, das gelöscht werden soll. Der Name ist ohne den folgenden Doppelpunkt einzugeben.
-PSProvider	N	Entfernt alle Laufwerke, die dem angegebenen Provider zugeordnet sind
-Force	S	Durch Angabe dieses Parameters kann auch das aktuell verwendete PowerShell-Laufwerk gelöscht werden.
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help -Name Remove-PSDrive -Full

Die folgenden Beispiele zeigen den möglichen Einsatz des Cmdlets:

- 1) Sie möchten das vorher mit New-PSDrive angelegte Laufwerk *Daten* löschen:

```
Remove-PSDrive -Name Daten
```

- ✓ Die Angabe des eindeutigen Namens des Laufwerks reicht aus, um das entsprechende Laufwerk zu löschen.
- ✓ Wollen Sie mehr als ein Laufwerk löschen, verwenden Sie das Komma zur Trennung der Laufwerknamen.

- 2) Sie möchten das Laufwerk *Microsoft* : löschen, das Sie in einem früheren Beispiel angelegt haben.

Allerdings ist *Microsoft* : das momentan aktive Laufwerk:

```
Remove-PSDrive -Name Microsoft -Force
```

- ✓ Da *Microsoft* : das aktive Laufwerk ist, muss der Parameter -Force angegeben werden. Wird dieser Parameter nicht angegeben, erhalten Sie folgende Fehlermeldung: Das Laufwerk "Microsoft" kann nicht entfernt werden, da es gerade verwendet wird.

## 4.3 Cmdlets für die Arbeit mit Verzeichnissen

Für die Arbeit mit Datenspeichern – also den Laufwerken der unterschiedlichen Provider, nicht nur dem Dateisystem – stellt die PowerShell vielfältige Cmdlets zur Verfügung. Sie kennen bereits das Cmdlet Get-ChildItem, mit dem Sie Elemente an einem Speicherort abrufen können. Einfacher formuliert: Wie die bekannten Befehle *dir* oder *ls* ruft das Cmdlet den Inhalt eines Verzeichnisses ab.

Für die Navigation in PowerShell-Laufwerken können Sie vorhandene Kenntnisse nutzen. Auch die PowerShell kann mit absoluten und relativen Pfaden arbeiten und kennt die gängigen Zeichen für das Stammverzeichnis eines Laufwerks `\`, das aktuelle Verzeichnis `.`, das übergeordnete Verzeichnis `..` sowie das zu Beginn einer Sitzung automatisch gewählte Basisverzeichnis `\`.

Folgende Cmdlets stellt die PowerShell in diesem Zusammenhang zur Verfügung:

Cmdlet	Erläuterung
Get-Location	Ruft Informationen über das aktuelle Verzeichnis ab
Set-Location	Wechselt in das angegebene Verzeichnis
Push-Location	Fügt das aktuelle Verzeichnis in eine Liste von Verzeichnissen (Stapel, <i>stack</i> ) ein, die Sie sich merken wollen
Pop-Location	Ändert das aktuelle Verzeichnis in ein Verzeichnis, das Sie mithilfe von Push-Location einer „Merkliste“ hinzugefügt haben

Diese Cmdlets unterstützen die Arbeit mit Verzeichnissen und Laufwerken, verfügen aber durch ihre Parameter über mehr Flexibilität als die entsprechenden Befehle in früheren Eingabeaufforderungen.

### Get-Location – Informationen über das aktuelle Verzeichnis abrufen

Das Cmdlet `Get-Location` dient der Informationsbeschaffung über das aktuelle Verzeichnis. Es kann auch durch u. a. folgende Parameter zusätzliche Informationen liefern:

Get-Location		
Parameter	Typ	Beschreibung
-PSDrive	N	Liefert das aktuelle Verzeichnis auf dem angegebenen PowerShell-Laufwerk zurück.
-PSPrinter	N	Gibt das aktuelle Verzeichnis des Laufwerks an, das vom angegebenen PowerShell-Provider unterstützt wird
-StackName	N	Zeigt die Speicherorte an, die sich im angegebenen Stapel befinden (Stapel legen Sie mit <code>Push-Location</code> an.)
-Stack	S	Zeigt die Speicherorte, die sich im Standardstapel ( <i>default</i> ) befinden
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: <code>Get-Help -Name Get-Location -Full</code>

Wenn Sie `Get-Location` ohne Parameter verwenden, wird das aktuelle Verzeichnis angezeigt. Doch das Cmdlet kann noch mehr:

- 1) Sie wollen erfahren, welches Verzeichnis das aktuelle Arbeitsverzeichnis in der Registry ist:

```
Get-Location -PSPrinter Registry
```

- ✓ Haben Sie bereits in der aktuellen Sitzung in der Registry gearbeitet, wird das Arbeitsverzeichnis ausgegeben, ansonsten das Stammverzeichnis.

- 2) Ähnliches erreichen Sie, wenn Sie direkt nach einem bestimmten Laufwerk fragen:

```
Get-Location -PSDrive HKLM
```

- ✓ Da der Provider *Registry* mehrere Laufwerke anbietet, handelt es sich hierbei um eine feinere Abfrage, da Sie explizit nach einem bestimmten Laufwerk fragen.

## Set-Location – das aktuelle Verzeichnis wechseln

Set-Location wechselt in ein angegebenes Verzeichnis (Parameter –Path). Zusätzlich können Sie eine Ausgabe des Pfades erzeugen (Parameter –PassThru).

Set-Location		
Parameter	Typ	Beschreibung
-Path	P (1)	Pfadangabe des Verzeichnisses, in das Sie wechseln wollen
-PassThru	S	Übergibt das Objekt an die Pipeline. Standardmäßig erfolgt keine Ausgabe.
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help -Name Set-Location -Full

## Push-Location – Verzeichnisse für später merken

Das Cmdlet bietet Ihnen die Möglichkeit, die Pfadangabe z. B. häufiger verwendeter Verzeichnisse in einen oder mehrere Stapel zu schreiben, so dass Sie sie mit einem anderen Cmdlet (Pop-Location) schnell aufrufen können.

Push-Location bietet u. a. folgende Parameter:

Push-Location		
Parameter	Typ	Beschreibung
-Path	P (1)	Pfadangabe des Verzeichnisses, in das Sie wechseln wollen Zusätzlich wird das Verzeichnis, in dem Sie sich gerade befinden, an oberster Stelle eines angegebenen Stapels abgelegt (Parameter –StackName). Wird dieser Parameter nicht angegeben, wird der Standardstapel verwendet.
-StackName	N	Gibt den Namen des Stapels an, in den Sie den aktuellen Pfad speichern wollen
-PassThru	S	Übergibt das Objekt an die Pipeline. Standardmäßig erfolgt keine Ausgabe.
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help -Name Push-Location -Full

Die folgenden Beispiele zeigen Ihnen die mögliche Verwendung des Cmdlets:

- 1) Sie befinden sich im Verzeichnis C:\Windows und möchten in das Verzeichnis C:\ wechseln. Zusätzlich soll das aktuelle Verzeichnis in den Stapel wichtig zur späteren schnellen Ansteuerung gelegt werden:

```
Push-Location C:\ -StackName wichtig
```

- ✓ Sie erhalten – außer dem Verzeichniswechsel nach C:\ – keine Rückmeldung der PowerShell. Prüfen Sie den Stapel **wichtig** mithilfe des Befehls Get-Location –StackName **wichtig**, erhalten Sie folgende bestätigende Ausgabe:  

```
Path
-----
C:\Windows
```

- 2) Sie möchten den aktuellen Pfad einem Stapel hinzufügen:

```
Push-Location -StackName wichtig
```

- ✓ Für das Beispiel ist das aktuelle Verzeichnis C:\Users\Administrator.
- ✓ Durch den Befehl findet kein Verzeichniswechsel statt, sondern nur eine Zuordnung des aktuellen Verzeichnisses zum Stapel *wichtig*. Dieser Pfad wird als oberstes Element auf den Stapel gelegt.
- ✓ Eine Prüfung mithilfe des Befehls Get-Location -StackName wichtig, ergibt dann folgende Ausgabe:

```
Path
-----
C:\Users\Administrator
C:\Windows
```

### **Pop-Location – schneller Wechsel in ein Verzeichnis**

Pop-Location ändert das aktuelle Verzeichnis in ein Verzeichnis, das an oberster Stelle in einem Stapel („Merkliste“) steht. Dieses Verzeichnis wird mit dem Aufruf vom Stapel entfernt. Das Cmdlet bietet u. a. folgende Parameter:

Pop-Location		
Parameter	Typ	Beschreibung
-StackName	N	Gibt den Namen des Stapels an, von dem Sie den zuletzt abgelegten Pfad aufrufen wollen. Ohne diesen Parameter wird der zuletzt hinzugefügte Pfad des aktuellen Stapels aufgerufen.
-PassThru	S	Übergibt das Objekt an die Pipeline. Standardmäßig erfolgt keine Ausgabe.
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help -Name Pop-Location -Full

Angenommen, der Stapel *wichtig* verfügt über folgende Einträge:

```
Path
-----
C:\Windows
C:\Users\Administrator
```

Sie geben nun folgenden Befehl ein:

```
Pop-Location -StackName wichtig
```

- ✓ Das Verzeichnis wird gewechselt. Das aktuelle Verzeichnis ist C:\. Der Stapel besteht nur noch aus dem zweiten und dritten Eintrag.
- ✓ Wenn Sie ein zweites Mal den Befehl ausführen, wechselt die PowerShell in das Verzeichnis C:\Windows. Der Stapel besteht nun aus dem letzten Eintrag.
- ✓ Bei der dritten Ausführung des Befehls wechseln Sie in das Verzeichnis C:\Users\Administrator. Der Stapel ist leer bzw. nicht mehr vorhanden.

## 4.4 Cmdlets für die Arbeit mit Elementen und ihren Eigenschaften

Mit dem Begriff Element (*item*) sind hier Verzeichnisse eines Laufwerks, Dateien im Dateisystem oder Schlüssel in der Registry gemeint. Sie kennen ähnliche Befehle, seit Sie mit Computern arbeiten. Die PowerShell stellt ihren eigenen Befehlssatz für die Arbeit auf den Laufwerken der verschiedenen Provider. Sie erkennen die Cmdlets an der Verwendung des Substantivs *Item* im Namen.

Die PowerShell unterstützt ähnliche Cmdlets auch für die Arbeit mit den Eigenschaften der genannten Elemente. Cmdlets dieses Themengebiets verwenden in ihrem Namen das Substantiv *ItemProperty*.

Folgende Cmdlets gehören in diese Gruppen:

Cmdlet	Erläuterung und Syntax (mit einer Auswahl wichtiger Parameter)
Clear-Item	Löscht den Inhalt eines Elements, das Element selbst bleibt erhalten. Verwendet wird das Cmdlet z. B. für das Löschen von Werten einer Variablen oder Zweige in der Registry. <code>Clear-Item [-Path] &lt;String[]&gt; [-Force] [-Filter &lt;String&gt;] [-Include &lt;String[]&gt;] [-Exclude &lt;String[]&gt;]</code>
Clear-ItemProperty	Löscht den Wert einer Eigenschaft eines Elements, nicht aber die Eigenschaft selbst <code>Clear-ItemProperty [-Path] &lt;String[]&gt; [-Name] &lt;String&gt; [-PassThru] [-Force] [-Filter &lt;String&gt;] [-Include &lt;String[]&gt;] [-Exclude &lt;String[]&gt;]</code>
Copy-Item	Kopiert ein Element an einen anderen Ort <code>Copy-Item [-Path] &lt;String[]&gt; [[-Destination] &lt;String&gt;] [-Container] [-Force] [-Filter &lt;String&gt;] [-Include &lt;String[]&gt;] [-Exclude &lt;String[]&gt;] [-Recurse] [-PassThru]</code>
Copy-ItemProperty	Kopiert die Eigenschaft eines Elements an einen anderen Ort <code>Copy-ItemProperty [-Path] &lt;String[]&gt; [-Destination] &lt;String&gt; [-Name] &lt;String&gt; [-PassThru] [-Force] [-Filter &lt;String&gt;] [-Include &lt;String[]&gt;] [-Exclude &lt;String[]&gt;]</code>
Get-Item	Ruft Informationen eines Elements ab <code>Get-Item [-Path] &lt;String[]&gt; [-Filter &lt;String&gt;] [-Include &lt;String[]&gt;] [-Exclude &lt;String[]&gt;] [-Force]</code>
Get-ItemProperty	Ruft die Eigenschaften eines Elements ab <code>Get-ItemProperty [-Path] &lt;String[]&gt; [[-Name] &lt;String[]&gt;] [-Filter &lt;String&gt;] [-Include &lt;String[]&gt;] [-Exclude &lt;String[]&gt;]</code>
Invoke-Item	Führt die im Betriebssystem festgelegte Standardaktion für das angegebene Element aus. Auf diesem Weg können Sie ausführbare Dateien starten oder Dateien mit registrierten Dateiendungen in ihrem zugewiesenen Standardprogramm öffnen. <code>Invoke-Item [-Path] &lt;String[]&gt; [-Filter &lt;String&gt;] [-Include &lt;String[]&gt;] [-Exclude &lt;String[]&gt;]</code>
Move-Item	Verschiebt ein Element an einen anderen Ort <code>Move-Item [-Path] &lt;String[]&gt; [[-Destination] &lt;String&gt;] [-Force] [-Filter &lt;String&gt;] [-Include &lt;String[]&gt;] [-Exclude &lt;String[]&gt;] [-PassThru]</code>

Cmdlet	Erläuterung und Syntax (mit einer Auswahl wichtiger Parameter)
Move-ItemProperty	Verschiebt die Eigenschaft eines Elements an einen anderen Ort <pre>Move-ItemProperty [-Path] &lt;String[]&gt; [-Destination] &lt;String&gt; [-Name] &lt;String[]&gt; [-PassThru] [-Force] [-Filter &lt;String&gt;] [-Include &lt;String[]&gt;] [-Exclude &lt;String[]&gt;]</pre>
New-Item	Erstellt ein neues Element <pre>New-Item [-Path] &lt;String[]&gt; [-ItemType &lt;String&gt;] [-Value &lt;Object&gt;] [-Force]</pre>
New-ItemProperty	Erstellt für ein vorhandenes Element eine neue Eigenschaft und legt dessen Wert fest <pre>New-ItemProperty [-Path] &lt;String[]&gt; [-Name] &lt;String&gt; [-.PropertyType &lt;String&gt;] [-Value &lt;Object&gt;] [-Force] [-Filter &lt;String&gt;] [-Include &lt;String[]&gt;] [-Exclude &lt;String[]&gt;]</pre>
Remove-Item	Löscht ein Element <pre>Remove-Item [-Path] &lt;String[]&gt; [-Filter &lt;String&gt;] [-Include &lt;String[]&gt;] [-Exclude &lt;String[]&gt;] [-Recurse] [-Force]</pre>
Remove-ItemProperty	Löscht eine Eigenschaft eines Elements <pre>Remove-ItemProperty [-Path] &lt;String[]&gt; [-Name] &lt;String[]&gt; [-Force] [-Filter &lt;String&gt;] [-Include &lt;String[]&gt;] [-Exclude &lt;String[]&gt;]</pre>
Rename-Item	Benennt ein Element um <pre>Rename-Item [-Path] &lt;String&gt; [-NewName] &lt;String&gt; [-Force] [-PassThru]</pre>
Rename-ItemProperty	Benennt eine Eigenschaft eines Elements um <pre>Rename-ItemProperty [-Path] &lt;String&gt; [-Name] &lt;String&gt; [-NewName] &lt;String&gt; [-PassThru] [-Force] [-Filter &lt;String&gt;] [-Include &lt;String[]&gt;] [-Exclude &lt;String[]&gt;]</pre>
Set-Item	Ändert den Wert eines Elements <pre>Set-Item [-Path] &lt;String[]&gt; [-Value] &lt;Object&gt; [-Force] [-PassThru] [-Filter &lt;String&gt;] [-Include &lt;String[]&gt;] [-Exclude &lt;String[]&gt;]</pre>
Set-ItemProperty	Ändert bzw. erstellt den Wert einer Eigenschaft eines Elements <pre>Set-ItemProperty [-Path] &lt;String[]&gt; [-Name] &lt;String&gt; [-Value] &lt;Object&gt; [-PassThru] [-Force] [-Filter &lt;String&gt;] [-Include &lt;String[]&gt;] [-Exclude &lt;String[]&gt;]</pre>

Die Cmdlets weisen starke syntaktische Parallelien auf:

- ✓ Alle Cmdlets verfügen über den Parameter `-Path`, der als Wert den Pfad des gewünschten Elements erwartet.
- ✓ Bei Aktionen wie Kopieren oder Verschieben, die zwingend die Angabe eines Ziels erwarten, bieten die Cmdlets den dafür vorgesehenen Parameter `-Destination`.
- ✓ Viele der Cmdlets bieten darüber hinaus Parameter zum Ein- bzw. Ausschluss von Elementen (`-Include` und `-Exclude`) oder für den Einsatz eines Filters (`-Filter`).
- ✓ Ist ein Element schreibgeschützt, kann die Aktion erzwungen werden (Parameter `-Force`).

Einige Beispiele zeigen den Einsatz dieser Gruppe von Cmdlets:

- 1) Sie wollen aus der PowerShell heraus a) eine Anwendung und b) eine Excel-Datei mit der Anwendung Excel starten:

```
a) Invoke-Item -Path .\update.exe
b) Invoke-Item -Path "C:\Meine Dateien\Bearbeiten\rechnung.xlsx"
```

- ✓ In Beispiel a) geben Sie als Wert des Parameters –Path die ausführbare Datei *update.exe* im aktuellen Verzeichnis an. Das Programm startet daraufhin in einem eigenen Fenster.
- ✓ In Beispiel b) wird die Datei *rechnung.xlsx* geöffnet. Sie befindet sich in einem Verzeichnis, das als absoluter Pfad angegeben ist. Die Pfadangabe ist von Anführungszeichen umschlossen, weil sie ein Leerzeichen beinhaltet. Die Datei wird mit dem Standardprogramm geöffnet, das in Windows mit der Dateiendung \*.xlsx verknüpft ist. Standardmäßig handelt es sich dabei um Microsoft Excel.

- 2) Sie wollen mit der PowerShell a) eine einzelne Datei und b) ein komplettes Verzeichnis an einen anderen Ort kopieren:

```
a) Copy-Item -Path C:\Windows\Logs\DISM\dism.log -Destination C:\MeineLogs
b) Copy-Item -Path C:\Windows\Logs -Destination C:\MeineLogs -Recurse
```

- ✓ In Beispiel a) wird die Datei *dism.log* in das Verzeichnis *C:\MeineLogs* kopiert. Sollte das Zielverzeichnis nicht existieren, wird die Quelldatei unter dem Namen *MeineLogs* in das Stammverzeichnis des Laufwerks *C:* gespeichert.
- ✓ Beispiel b) zeigt das Kopieren eines kompletten Verzeichnisses inklusive aller enthaltenen Unterverzeichnisse und Dateien (Parameter –Recurse). Sollte das Zielverzeichnis nicht existieren, wird es erstellt. Sind im Zielverzeichnis bereits vor dem Kopieren gleichnamige Dateien oder Verzeichnisse enthalten, erhalten Sie für jedes Auftreten eine Fehlermeldung („Das Element mit dem angegebenen Namen ist bereits vorhanden.“).

- 3) Sie möchten aus einem Verzeichnis alle temporären Dateien (mit der Dateiendung \*.tmp) entfernen:

```
Get-ChildItem -Path C:\Test -Include *.tmp | Remove-Item
```

- ✓ In diesem Beispiel sucht das Cmdlet *Get-ChildItem* nach allen Dateien mit der Endung *\*.tmp* im angegebenen Verzeichnis. Die Ergebnisse werden über die Pipeline gegeben und vom Cmdlet *Remove-Item* gelöscht.
- ✓ Falls Sie nicht ganz sicher sind, ob diese Befehle unbeabsichtigte Aktionen ausführen, ergänzen Sie das Cmdlet *Remove-Item* um den Parameter *-WhatIf*. Damit wird die Ausführung der Aktionen nur simuliert und am Bildschirm ausgegeben.
- ✓ Sollen auch schreibgeschützte Dateien gelöscht werden, fügen Sie beim zweiten Cmdlet den Parameter *-Force* hinzu. Andernfalls erhalten Sie die Fehlermeldung: „Das Element [...] kann nicht entfernt werden: Sie besitzen keine ausreichenden Berechtigungen zum Ausführen dieses Vorgangs.“

- 4) Sie wollen alle Eigenschaften einer Datei anzeigen lassen:

```
Get-ItemProperty -Path C:\Users\Administrator\test.txt | Select-Object -Property *
```

- ✓ Wenn Sie nur den ersten Befehl eingeben, erhalten Sie eine Ausgabe wie die folgende Beispielausgabe:

Mode	LastWriteTime	Length	Name
---	-----	-----	---
-a--	03.05.2014	12:15	911 test.txt
- ✓ Setzen Sie allerdings die Pipeline ein und ergänzen den zweiten Befehl, erhalten Sie eine Liste mit mehr als 20 Eigenschaften und ihren Werten.

- 5) Sie wollen eine Datei mit einem Schreibschutz versehen:

```
Set-ItemProperty -Path .\fertig.pptx -Name IsReadOnly -Value $true
```

- ✓ Mit dem gezeigten Befehl setzen Sie die Eigenschaft *IsReadOnly* der Datei *fertig.pptx* auf den Wert `$true`. Die Datei ist nun mehr schreibgeschützt, was Sie durch einen Schreibversuch oder durch einen Blick in die Eigenschaften der Datei überprüfen können.

Die vorgestellten Cmdlets zur Arbeit mit Dateispeichern sind allgemeingültig und anwendbar auf die Laufwerke der verschiedenen Provider. In den folgenden Abschnitten lernen Sie weitere Cmdlets für den Einsatz mit speziellen Laufwerken kennen bzw. die vorgestellten Cmdlets in ihrer spezifischen Anwendung auf einzelne Laufwerke.

## 4.5 Laufwerke im Dateisystem (Provider FileSystem)

Die PowerShell stellt Ihnen für die Arbeit im Dateisystem einige Cmdlets zur Verfügung, mit denen Sie den Inhalt von Dateien (*content*) bearbeiten bzw. auslesen oder schreiben können.

Cmdlet	Erläuterung
Add-Content	Fügt einem oder mehreren Elementen den angegebenen Inhalt hinzu
Clear-Content	Löscht den Inhalt eines Elements, z. B. den Text einer Datei
Get-Content	Ruft den Inhalt eines Elements ab
Set-Content	Schreibt Inhalt in ein Element neu oder ersetzt vorhandenen Inhalt

### Get-Content – Auslesen von Inhalten

Ein wichtiges und häufig verwendetes Cmdlet, das in den Laufwerken des Dateisystems die Inhalte von Textdateien in die PowerShell einliest, ist Get-Content, das u. a. folgende Parameter unterstützt:

Get-Content		
Parameter	Typ	Beschreibung
-Path	P (1)	Gibt den Pfad zu einem Element an
-Exclude	N	Lässt angegebene Elemente aus; wird häufig mit einem Muster als Wert verwendet, wie z. B. <code>*.log</code> oder <code>windows*</code>
-Include	N	Beschränkt die Ausführung auf angegebene Elemente; wird häufig mit einem Muster als Wert verwendet, wie z. B. <code>*.log</code> oder <code>windows*</code>
-ReadCount	N	Legt fest, wie viele ausgelesene Zeilen gleichzeitig über die Pipeline gesendet werden. Standardwert ist <code>1</code> , bei dem Wert <code>0</code> wird der komplette Inhalt auf einmal übergeben. Der Parameter steuert die Menge an Daten, die gleichzeitig übergeben werden, nimmt inhaltlich keine Änderungen vor.
-TotalCount	N	Steuert, wie viele Zeilen (vom Beginn) des Elements abgerufen werden. Der Standardwert ist <code>-1</code> (= alle Zeilen).
-Tail	N	Legt die Anzahl der Zeilen fest, die vom Ende des Elements abgerufen werden

Get-Content		
Parameter	Typ	Beschreibung
-Force	S	Erzwingt die Ausführung, falls z. B. Schreibschutz oder fehlende Verzeichnisse eines Dateipfads die Ausführung verhindern würden
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help -Name Get-Content -Full

Die folgenden Beispiele zeigen das Cmdlet im Einsatz:

- 1) Sie möchten den Inhalt der Datei *notiz.txt* am Bildschirm anzeigen:

```
Get-Content -Path .\notiz.txt
```

- ✓ Der Inhalt der Datei wird am Bildschirm ausgegeben. Häufig ist eine Weiterverarbeitung über die Pipeline das eigentliche Ziel.

- 2) Sie wollen aus allen Textdateien des Laufwerks C: die ersten beiden Zeilen anzeigen:

```
Get-ChildItem -Path C:\ -Filter *.txt -Recurse | Get-Content -TotalCount 2
```

- ✓ Je nach Rechner und Größe des Laufwerks C: sowie der Anzahl der dort gespeicherten Dateien kann die Ausführung dieses Cmdlets einige Zeit in Anspruch nehmen. Modifizieren Sie bei Bedarf die Pfadangabe.
- ✓ Bei der Ausgabe werden Sie etliche Fehlermeldungen sehen, da Ihnen das System in der Regel den Zugriff auf einige Ordner nicht gestattet (Meldung: „Der Zugriff auf den Pfad [...] wurde verweigert.“). Um die Fehlermeldungen auszublenden, ergänzen Sie den Befehl Get-ChildItem um -ErrorAction SilentlyContinue.

### Set-Content – Inhalt in eine Datei schreiben

Set-Content schreibt gewünschte Inhalte in ein Element. Das Cmdlet wird häufig am Ende einer Pipeline verwendet, um vorher optimierte Ergebnisse permanent zu speichern. Das Cmdlet unterstützt u. a. folgende Parameter:

Set-Content		
Parameter	Typ	Beschreibung
-Path	P (1)	Gibt den Pfad zu einem Element an
-Value	P(2)	Gibt den Inhalt für das angegebene Element an. Lassen Sie den Parameter weg, werden die Objekte verwendet, die über die Pipeline gereicht werden.
-Exclude	N	Lässt angegebene Elemente aus; wird häufig mit einem Muster als Wert verwendet, wie z. B. *.log oder windows*
-Include	N	Beschränkt die Ausführung auf angegebene Elemente; wird häufig mit einem Muster als Wert verwendet, wie z. B. *.log oder windows*
-Force	S	Erzwingt die Ausführung z. B. bei Schreibschutz des angegebenen Elements
-PassThru	S	Sorgt dafür, dass am Bildschirm der Inhalt ausgegeben wird. Standardmäßig erfolgt keine Ausgabe.
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help -Name Set-Content -Full

Die folgenden Beispiele zeigen, wie Set-Content eingesetzt werden kann:

- 1) Sie wollen eine Liste der aktuellen Prozesse auf Ihrem Rechner in die Datei *prozesse.txt* speichern:

```
Get-Process | Select-Object -Property Id, Name | Set-Content -Path .\prozesse.txt
```

- ✓ Die Liste der Prozesse wird an das Cmdlet Select-Object übergeben. Dort wird die Auswahl auf die Eigenschaften *ID* und *Name* begrenzt.
- ✓ Die Ergebnisse werden über die Pipeline an das Cmdlet Set-Content gereicht und in die angegebene Datei geschrieben.

- 2) Sie möchten eine Zeichenfolge in eine Datei schreiben und gleichzeitig am Bildschirm ausgeben:

```
Set-Content -Path .\notiz.txt -Value "Wichtiger Termin 14 Uhr!" -PassThru
```

- ✓ In diesem Beispiel wird eine selbst definierte, kurze Zeichenfolge in die angegebene Datei geschrieben.
- ✓ Der Parameter *-PassThru* sorgt für eine zusätzliche Ausgabe der Zeichenfolge am Bildschirm.

### Add-Content – Inhalt hinzufügen

Add-Content fügt Inhalt an ein Element, z. B. eine Datei, an. Der gewünschte Inhalt wird manuell eingegeben oder von einem Objekt übernommen. Das Cmdlet unterstützt dieselben Parameter wie das Cmdlet Set-Content.

Die folgenden Beispiele zeigen den möglichen Einsatz des Cmdlets Add-Content:

- 1) Sie möchten einer Textdatei das aktuelle Datum hinzufügen:

```
Add-Content -Path .\notiz.txt -Value (Get-Date)
```

- ✓ Der Wert, der hinzugefügt werden soll, ergibt sich aus dem Cmdlet Get-Date. Um zu vermeiden, dass nur die Zeichenfolge „Get-Date“ an das Ende der Datei geschrieben wird, müssen Sie die Berechnung zuerst ausführen. Um dies zu erreichen, setzen Sie das Cmdlet in runde Klammern.

- 2) Sie möchten den Inhalt einer bestehenden Datei in eine noch nicht vorhandene Datei schreiben:

```
Add-Content -Path .\neu.txt -Value (Get-Content -Path .\notiz.txt)
```

- ✓ Sie lesen mithilfe des Cmdlets Get-Content den Inhalt der Datei *notiz.txt* aus. Diese Daten werden in die Datei *neu.txt* eingefügt. Ist diese Datei nicht vorhanden, wird sie automatisch angelegt.

### Clear-Content – Inhalt löschen

Clear-Content löscht den Inhalt eines Elements, z. B. den Text in einer Datei. Das Element selbst wird dabei nicht gelöscht. Clear-Content ist vergleichbar mit dem bereits vorgestellten Cmdlet Clear-Item. Während Clear-Item für Aliase und Variablen (siehe weiter unten) verwendet wird, wird das Cmdlet Clear-Content für Dateien eingesetzt.

Das Cmdlet unterstützt dieselben Parameter wie das Cmdlet Set-Content. Beispiele für den Einsatz des Cmdlets finden Sie nachfolgend:

- 1) Sie möchten den Inhalt aller Protokolldateien eines bestimmten Verzeichnisses löschen:

```
Clear-Content -Path C:\Protokolldateien -Filter *.log -Force
```

- ✓ Die Angabe des Pfads kann auf zwei Arten geschehen: Zum einen können Sie wie im Beispiel mit einem Filter arbeiten, der den Befehl auf Dateien mit der Erweiterung `*.log` beschränkt. Zum anderen können Sie direkt im Pfad `C:\Protokolldateien\*.log` angeben. Die erste Variante bringt Geschwindigkeitsvorteile in der Ausführung des Befehls.
  - ✓ Der Parameter `-Force` erzwingt die Anwendung des Befehls auf schreibgeschützte Dateien.
- 2) Sie möchten den Inhalt von Dateien im aktuellen Verzeichnis löschen, deren Dateiname mit `abc` beginnt und nicht die Zeichenfolge `234` enthält:

```
Clear-Content -Path * -Include abc* -Exclude *234*
```

- ✓ Wenn z. B. im aktuellen Verzeichnis die Dateien `abc.txt` und `abc234.log` vorhanden sind, wird mit diesem Befehl der Inhalt der Datei `abc.txt` gelöscht.

## 4.6 Registry-Laufwerke (Provider Registry)

Der Provider Registry bietet die beiden Laufwerke `HKCU`: (für `HKEY_CURRENT_USER`) und `HKLM`: (für `HKEY_LOCAL_MACHINE`). Weitere Cmdlets speziell für die Registry bietet die PowerShell nicht. Alle vorgestellten Cmdlets eignen sich für einen Einsatz in diesem Bereich.

Anhand eines etwas umfangreicherer, aufeinander aufbauenden Beispiels lernen Sie den Umgang mit diesem Datenspeicher. Sie können die jeweiligen Ergebnisse im Registry-Editor (`regedit.exe`) nachvollziehen:

- 1) Sie wollen im Laufwerk `HKLM`: im Schlüssel `SOFTWARE` einen eigenen Unterschlüssel `Meine Software` anlegen.

```
Set-Location -Path HKLM:\SOFTWARE
```

- ✓ Häufig wechseln Sie in den Schlüssel, in dem Sie den Unterschlüssel erstellen wollen, auch wenn dies nicht unbedingt nötig ist.

```
New-Item -Path "Meine Software" -ItemType Container
```

- ✓ Nun legen Sie den Schlüssel `Meine Software` an. Der Wert ist in Anführungszeichen zu setzen, da der Wert ein Leerzeichen enthält.
- ✓ Da ein Schlüssel in der Registry vergleichbar mit einem Verzeichnis im Dateisystem ist, lautet der Wert des Parameters `-ItemType Container`.

- 2a) Sie wollen in diesem Schlüssel einen neuen Eintrag `Version` erstellen, der als Wert die Zeichenfolge `5.02 Professional` erhält.

```
New-ItemProperty -Path "HKLM:\SOFTWARE\Meine Software" -Name Version -Value "5.02 Professional" -PropertyType String
```

- ✓ Sie verwenden das Cmdlet `New-ItemProperty`, um einen Eintrag in einem Registry-Schlüssel anzulegen. In diesem Beispiel wird der Pfad absolut angegeben.
- ✓ Im Wert des Parameters `-Name` legen Sie die Bezeichnung des Eintrags fest, im Wert des Parameters `-Value` den dazugehörigen Wert des Eintrags.
- ✓ Mit dem Parameter `-PropertyType` legen Sie den Eintragstyp fest. In diesem Fall ist es eine Zeichenfolge (`string`).

2b) Ein weiterer Eintrag *Hexwert* soll den Wert *0xABCD* von Typ *DWORD* erhalten:

```
New-ItemProperty -Path "HKLM:\SOFTWARE\Meine Software" -Name Hexwert -Value 0xABCD -PropertyType DWORD
```

- ✓ In diesem Befehl verwenden Sie den Parameter *-PropertyType* mit dem Wert *DWORD*.
- ✓ Ein Hexadezimalwert wird mit der Zeichenfolge *0x* eingeleitet.

3) Der Eintrag *Version* soll verändert werden: Der neue Wert ist *5.23 Professional*:

```
Set-ItemProperty -Path "HKLM:\SOFTWARE\Meine Software" -Name Version -Value "5.23 Professional"
```

- ✓ Für diese Aktion verwenden Sie das Cmdlet *Set-ItemProperty* und legen den neuen Wert fest.

4) Schließlich wird der Eintrag *Hexwert* gelöscht:

```
Remove-ItemProperty -Path "HKLM:\SOFTWARE\Meine Software" -Name Hexwert
```

- ✓ Mithilfe des Cmdlets *Remove-ItemProperty* können Sie den angegebenen Eintrag löschen.

Wie Sie an diesem Beispiel sehen, reichen die allgemeinen Cmdlets für die Arbeit mit Datenspeichern für die Arbeit mit den Laufwerken der Registry aus. Die PowerShell liefert darüber hinaus durch Skripting die Möglichkeit, solche Aufgaben für Ihr komplettes Netzwerk recht einfach und schnell umzusetzen.

## 4.7 Laufwerk für Umgebungsvariablen (Provider Environment)

Der Provider *Environment* stellt das Laufwerk *Env:* zur Verfügung. In ihm befinden sich als Einzelobjekte die Windows-Umgebungsvariablen. Die Objekte können Sie mit den vorgestellten Mittel auslesen, wie z. B.:

```
Get-ChildItem -Path Env:\
```

Wenn Sie Umgebungsvariablen verwenden wollen, sprechen Sie die gewünschte Variable mit folgender Syntax an:

```
$Env:<Variablenname>
```

- ✓ Das Dollarzeichen (\$) kennzeichnet eine Variable. Es folgt die Bezeichnung des Laufwerks mit abschließendem Doppelpunkt. An letzter Stelle folgt der Name der Variablen.
- ✓ \$Env:windir (Windows-Installationsverzeichnis), \$Env:USERNAME für den aktuellen Benutzer und \$Env:USERPROFILE (Pfad zum Benutzerprofil) sind einige häufiger verwendete Beispiele.

Die vorgestellten Cmdlets können Sie wie bei anderen Laufwerken einsetzen. Folgendes müssen Sie allerdings berücksichtigen:

- ✓ Wenn Sie eigene Elemente in diesem Laufwerk definieren, können Sie keine Verzeichnisse anlegen. Alle Definitionen bestehen aus Name und Wert.
- ✓ Eigene Definitionen sind nur bis zum Beenden der aktuellen PowerShell-Sitzung gültig.
- ✓ Vordefinierte Variablen können gelöscht und in der aktuellen PowerShell-Sitzung nicht mehr verwendet werden. Beim Start einer anderen PowerShell-Sitzung sind sie wieder vorhanden.

## 4.8 Kurz zusammengefasst

Die PowerShell versteht es, über verschiedene Provider Laufwerke anzubieten, die weitgehend nach einem identischen Muster funktionieren. Der Zugriff auf Aliase, Variablen, Windows-Umgebungsvariablen und sogar auf die Registry wird über diese virtuellen Laufwerke realisiert.

Diese Herangehensweise bzw. „Normierung“ vereinfacht die Arbeit mit diesen Daten enorm. Die grundlegenden Funktionen sind allseits bekannt, da sie sich an der Arbeit mit den Laufwerken des Dateisystems orientieren.

## 4.9 Übung

### Mit Laufwerken arbeiten

**Übungsdatei:** –

**Ergebnisdatei:** *4\_ergebnisse.txt*

1. Lassen Sie sich alle PowerShell-Laufwerke des Dateisystems anzeigen.
2. a) Legen Sie (mit der PowerShell) ein Verzeichnis *C:\Testdaten* an.  
b) Erstellen Sie ein neues PowerShell-Laufwerk mit Namen *Testdaten* und dem gerade erstellten Verzeichnis als Stammverzeichnis.  
c) Wechseln Sie in das neue PowerShell-Laufwerk.
3. Erstellen Sie im Stammverzeichnis des Laufwerks *Testdaten*: zwei Dateien:
  - a) eine neue, leere Textdatei mit Namen *notiz.txt*
  - b) eine Datei *prozesse.txt*, die als Inhalt den Namen und die ID der aktuellen Prozesse Ihres Rechners erhält, deren Name mit s oder w beginnt
4. Schreiben Sie in die Datei *notiz.txt*
  - a) aktuelle Datumsinformationen sowie
  - b) die Liste der Dateinamen im Laufwerk *Testdaten*:
5. Versehen Sie die Datei *prozesse.txt* mit einem Schreibschutz.
6. Löschen Sie den Inhalt der Datei *prozesse.txt*.
7. Löschen Sie das virtuelle PowerShell-Laufwerk *Testdaten*: aus der Liste der PowerShell-Laufwerke.

# 5 Aliase – alternative Kurzbefehle



Beispieldatei: 5\_kompletter code.txt (Ordner Kapitel 05)

## 5.1 Alias (Spitzname)

### Was ist ein Alias?

Ein Alias ist ein alternativer Kurzbefehl für ein Cmdlet, ein externes Kommando oder eine Funktion. Wenn Sie sich z. B. ein sperriges Cmdlet nicht merken können, verwenden Sie stattdessen einen definierten Kurzbefehl (oder „Spitznamen“). Ein Alias ist kein eigenständiger Befehl, sondern gibt einem bereits (vor-)definierten Befehl eine weitere Bezeichnung, unter der er angesprochen werden kann. Befehlsparameter können nicht in die Aliasdefinition integriert werden.

Aliase werden vornehmlich aus zwei Gründen eingesetzt:

- ✓ Erleichterung des Einstiegs in die PowerShell durch die Verwendung bekannter Befehle, die z. B. noch nicht bekannte Cmdlets aufrufen
- ✓ Arbeitserleichterung durch Definition kurzer Befehlsnamen für sperrige oder nicht leicht zu merkende Befehle

## 5.2 Vordefinierte und eigene Aliase

### Mit vordefinierten Aliasen arbeiten

Vordefinierte Aliase stehen Ihnen jederzeit in PowerShell zur Verfügung. Sie werden automatisch bei jedem Start der PowerShell geladen. Um eine Liste vordefinierter Aliase anzeigen zu lassen, geben Sie folgendes Cmdlet ein:

```
Get-Alias
```

- ✓ Sie erhalten eine Liste mit etwa 150 vordefinierten Aliasen. Die Liste ist alphabetisch nach Aliasname sortiert. Es sind ausschließlich Cmdlets mit Aliasen belegt, manche auch mit mehreren.

Wenn Sie einen bestimmten Alias suchen oder sich informieren wollen, ob ein bestimmtes Cmdlet einen Alias besitzt, setzen Sie das Cmdlet mit den entsprechenden Parametern ein:

```
Get-Alias [-Name] <string>
```

- ✓ Verwenden Sie den Parameter `-Name`, um herauszufinden, für welche Aktion der angegebene Alias definiert wurde.

### Beispiele

`Get-Alias -Name gal` führt zur (verkürzten) Ausgabe: `gal -> Get-Alias`.

`Get-Alias gcm` ergibt die Ausgabe: `gcm -> Get-Command`.

```
Get-Alias -Definition <string>
```

- ✓ Verwenden Sie den Parameter `-Definition`, wenn Sie wissen wollen, ob einer Aktion ein Alias zugewiesen wurde.
- ✓ Existiert kein gesuchter Alias, reagiert die PowerShell mit einer aussagekräftigen Fehlermeldung.

## Beispiele

`Get-Alias -Definition Get-Service` führt zur Ausgabe: *gsv* -> *Get-Service*.  
`Get-Alias -Definition Get-ChildItem` ergibt die Ausgabe: *dir, gci, ls* -> *Get-ChildItem*.

Bereits beim Auflisten von Aliasen können Sie auch mit dem Alias `dir` (alternativ für das PowerShell-Cmdlet `Get-ChildItem`) arbeiten und es im Zusammenhang mit dem virtuellen Laufwerk *Alias* des PowerShell-Providers Alias anwenden:

`dir Alias:`

- ✓ Der Befehl listet alle Aliase auf, indem es alle Objekte des Informationsspeichers *Alias* wie in einem Laufwerk durchläuft.

Die PowerShell hat Ihnen den Start vereinfacht, indem viele aus „alten Tagen“ bekannte Befehle auch in der PowerShell funktionieren. All diese bekannten Befehle sind Aliase für PowerShell-Cmdlets, z. B. `cls`, `cd` und `dir`.

Wollen Sie mit einem Alias arbeiten, geben Sie ihn anstelle des Cmdlets ein, wie z. B.:

`start notepad.exe`

- ✓ Der Alias `start` ersetzt hier das Cmdlet `Start-Process`.
- ✓ Im Beispiel wird der Windows-Editor als neuer Prozess gestartet.

## Fortgeschrittenes Beispiel



So können Sie herausfinden, welchen Cmdlets wie viele und welche Aliase zugewiesen wurden:

`Get-Alias | Group-Object -Property definition | Sort-Object -Property count`

## Eigene Aliase definieren

Um einen eigenen Alias zu definieren, stellt die PowerShell zwei Cmdlets zur Verfügung: `Set-Alias` und `New-Alias`. Die Unterschiede zwischen beiden Cmdlets sind marginal: `Set-Alias` erstellt einen neuen Alias oder überschreibt einen bereits vorhandenen Alias gleichen Namens. `New-Alias` erzeugt im letzten Fall einen Fehler.

`Set-Alias [-Name] <String> [-Value] <String>`  
`New-Alias [-Name] <String> [-Value] <String>`

- ✓ Der gewünschte Name des Alias ist als Wert des Parameters `-Name` einzutragen.
- ✓ Das Cmdlet, die Funktion oder das externe Kommando, das einen Alias erhalten soll, ist als Wert des Parameters `-Value` einzutragen.

## Beispiele

- ✓ `Set-Alias -Name hilfmir -Value Get-Help`  
erstellt einen Alias namens *hilfmir* als Alternative zum Cmdlet `Get-Help`. Mit `Get-Alias -Name hilfmir` können Sie überprüfen, ob die Aktion erfolgreich war.
- ✓ `New-Alias paint C:\Windows\System32\mspaint.exe`  
definiert einen neuen Alias *paint*, der die windowsinterne Anwendung Microsoft Paint aufruft.

Eigens definierte Aliase stehen nur in der aktuellen Instanz der PowerShell zur Verfügung. Wenn Sie die PowerShell schließen und erneut öffnen, stehen die von Ihnen definierten Aliase nicht mehr zur Verfügung.



## 5.3 Ex- und Import von Aliasen

### Aliase exportieren

Um die Aliase der aktuellen Sitzung in einer Textdatei zu speichern, verwenden Sie das Cmdlet `Export-Alias`. Den Namen der Exportdatei geben Sie nach dem optionalen Parameter `-Path` an.

Setzen Sie das folgende Cmdlet zum Export von Aliasen der aktuellen Sitzung in eine Datei ein:

#### `Export-Alias`

Das Cmdlet bietet folgende wichtige Parameter:

Export-Alias		
Parameter	Typ	Beschreibung
<code>-Path</code>	P (1)	Angabe des Dateipfades der Exportdatei
<code>-Name</code>	P(2)	Filtermöglichkeit für die Aliase, die exportiert werden sollen. Verwenden Sie Platzhalter oder trennen Sie mehrere Argumente durch das Zeichen <code>,</code> voneinander ab, z. B.: <code>h*</code> (alle Aliase, die mit „x“ beginnen) oder <code>*ri*</code> (alle Alias, die im Namen an beliebiger Stelle die Zeichenfolge „ri“ aufweisen).
<code>-Append</code>	S	Bei Angabe des Parameters werden Daten an eine Datei angehängt.
<code>-As</code>	N	Mögliche Werte: Csv oder Script. Bei der Auswahl Script werden der Exportdatei Set-Alias-Cmdlets hinzugefügt, die Datei sollte mit der Endung <code>.ps1</code> als Power-Shell-Skriptdatei gespeichert werden.
<code>-NoClobber</code>	S	Die Angabe des Parameters verhindert das Überschreiben einer bereits vorhandenen Datei gleichen Namens.
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: <code>Get-Help Export-Alias -Full</code>

### Beispiele

- ✓ `Export-Alias -Path alias-export.csv`  
erstellt im Arbeitsverzeichnis eine Textdatei `alias-export.csv`, die alle Aliase der Sitzung speichert. Darin enthalten sind alle in der aktuellen Sitzung der PowerShell vorhandenen Aliase, also auch stets die vordefinierten Aliase. Eine Beispielzeile der Exportdatei sieht wie folgt aus:  
`"nal","New-Alias","","ReadOnly, AllScope"`.
- ✓ `Export-Alias -Path alias-export.ps1 -As Script -NoClobber`  
bewirkt, dass die Datei `alias-export.ps1` mit allen Aliasen der aktuellen Sitzung neu erstellt wird. Existiert bereits eine Datei gleichen Namens, bricht PowerShell mit einer Fehlermeldung ab. Der Parameter `-As` sorgt dafür, dass die Einträge als Skript gespeichert werden, z. B.:  
`set-alias -Name:"nal" -Value:"New-Alias" -Description:"" -Option:"ReadOnly, AllScope"`.



Es empfiehlt sich, nicht alle vorhandenen Aliase zu exportieren, sondern sich nach und nach eine Datei mit eigenen Aliasen aufzubauen. Zu diesem Zweck sind der Parameter `-Name` für einen zielgerichteten Export einzelner Aliase sowie der Parameter `-Append` zum Anhängen von Daten an eine bestehende Alias-Exportdatei zu empfehlen.

## Aliase importieren

Setzen Sie das folgende Cmdlet zum Import von Aliasen aus einer Datei in die aktuelle PowerShell-Sitzung ein:

### Import-Alias

Das Cmdlet bietet folgende wichtige Parameter:

Import-Alias		
Parameter	Typ	Beschreibung
-Path	P (1)	Angabe des Dateipfades der Importdatei
-Force	S	Bei Angabe des Parameters werden bereits definierte oder schreibgeschützte Aliase überschrieben.
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help Import-Alias -Full

## Beispiel

- ✓ Import-Alias -Path alias-export.csv -Force  
importiert die in der Textdatei *alias-export.csv* vorhandenen Aliase in die aktuelle PowerShell-Sitzung. Der Parameter *-Force* sorgt dafür, dass vorhandene Aliase gleichen Namens überschrieben werden. Ohne diesen Parameter werden keine Aliase überschrieben, sondern jeweils Fehlermeldungen ausgegeben.

## 5.4 Aliase löschen

### Eigene Aliase löschen

Sobald Sie die PowerShell-Sitzung beenden, verschwinden automatisch alle selbst angelegten Aliase. Doch Sie können auch Aliase manuell löschen. Suchen Sie das geeignete Cmdlet, werden Sie feststellen, dass für diese Tätigkeit kein spezielles Cmdlet existiert.

### Get-Command -Noun alias

- ✓ Das Cmdlet liefert alle Cmdlets, deren Substantiv exakt aus der Zeichenfolge *alias* besteht. Gesucht werden alle Cmdlets, die Tätigkeiten rund um die Aliase anbieten.
- ✓ Als Ergebnis werden die Cmdlets Export-Alias, Get-Alias, Import-Alias, New-Alias und Set-Alias angezeigt.

Zum Löschen müssen Sie einen anderen Weg gehen. Sie verwenden das allgemeine Cmdlet Remove-Item zum Löschen von Elementen. Sie wenden den Löschkvorgang auf das gewünschte Objekt im virtuellen Laufwerk *Alias*: an.

### Remove-Item -Path Alias:\<Name>

- ✓ Setzen Sie als <Name> z. B. den im Kapitel definierten Alias *paint* an, wird er gelöscht. Sollte in Ihrer aktuellen PowerShell-Sitzung kein Alias *paint* existieren, erhalten Sie die folgende Fehlermeldung: Remove-Item: Der Pfad "Alias:\paint" kann nicht gefunden werden, da er nicht vorhanden ist.

Sollten Sie von PowerShell vordefinierte Aliase löschen, sind diese beim Start einer neuen Instanz der PowerShell wieder vorhanden. Der Löschkvorgang betrifft auch hier nur die aktuelle Sitzung.

### Exkurs: Schreibgeschützte Aliase

Manche vordefinierten Aliase sind schreibgeschützt. Der Versuch, sie zu löschen, mündet in einer Fehlermeldung: Der Alias wurde nicht entfernt, da der Alias [...] konstant oder schreibgeschützt ist. Um herauszufinden, welche Aliase schreibgeschützt sind, geben Sie – in einer Zeile – Folgendes ein:

```
① Get-Alias |  
② Where-Object Options -Match "ReadOnly" |  
③ Select-Object -Property Name, Options
```

- ① Mit dem Cmdlet Get-Alias lassen Sie sich die vorhandenen Aliase anzeigen.
- ② Sie filtern die zurückgegebenen Objekte durch das Cmdlet Where-Object und der Angabe, dass Sie in der Eigenschaft *Options* eines jeden Alias nach einer Übereinstimmung mit der Zeichenkette *ReadOnly* suchen. In alter – PowerShell 2.0-kompatibler – Schreibweise ist das Cmdlet so zu formulieren:  
`Where-Object {$_.Options -Match „ReadOnly“}.`
- ③ Über das Cmdlet Select-Object filtern Sie, welche Eigenschaften der Aliase Sie sehen wollen.

## 5.5 Kurz zusammengefasst

### Aliase erleichtern das Leben

Aliase erleichtern die Arbeit in der PowerShell. Zum einen können Sie aus anderen Umgebungen bekannte Befehle weiterhin verwenden, zum anderen können Sie sich Tipparbeit sparen bzw. müssen nicht nach einem Cmdlet suchen, das Sie immer wieder vergessen.

Die PowerShell bringt selbst viele vordefinierte Aliase mit, die Sie um Ihre eigenen Aliase erweitern können. Auch für eine Möglichkeit des Imports und Exports von Aliasen ist gesorgt, da Ihre Definitionen das Sitzungsende nicht überdauern.

Wie Sie mit Profilen arbeiten und dafür sorgen, dass Ihnen ausgewählte Definitionen jederzeit zur Verfügung stehen, erfahren Sie im folgenden Kapitel.

### Was Sie beim Einsatz von Aliassen beachten sollten

- ✓ Seien Sie sich zuerst sicher in der Verwendung der Cmdlets inklusive der dazugehörigen Parameter, bevor Sie Aliase verwenden. Damit vermeiden Sie die Verwechslung mit „alten“, bekannten Befehlen und setzen die richtigen Parameter ein.
- ✓ Wenn Sie mit selbstdefinierten Aliasen arbeiten, sind diese auf anderen Rechnern nicht bekannt (oder im Extremfall dort definiert, aber mit einer anderen Funktion versehen).
- ✓ Verwenden Sie beim Schreiben von Skripten (vgl. Kapitel 8) keine Aliase. Je sauberer Sie ein Skript verfassen, desto verständlicher und nachvollziehbarer bleibt es. Bei Verwendung eigener Aliase funktioniert Ihr Skript möglicherweise nur auf Ihrem eigenen Rechner.



Wissenstest: PowerShell-Grundlagen-1

## 5.6 Übung

### Mit Aliasen arbeiten

Übungsdatei: –

Ergebnisdatei: 5\_ergebnisse.txt

1. Listen Sie alle Aliase auf, deren Name mit dem Buchstaben *H* beginnen.
2. Listen Sie alle Aliase für das Cmdlet Remove-Item auf.
3. Definieren Sie drei eigene Aliase mit den Namen *aaa*, *bbb* und *ccc*, die stellvertretend für die Cmdlets Get-Process, Get-Random und Get-Service verwendet werden sollen.  
Macht es einen Unterschied, ob Sie für die Definition der Aliase das Cmdlet New-Alias oder das Cmdlet Set-Alias verwenden?
4. Exportieren Sie die drei gerade definierten Aliase in eine Textdatei *eigene-alias.txt* im aktuellen Verzeichnis.
5. Definieren Sie einen vierten Alias *ddd* für das Cmdlet Get-PSProvider.
6. Exportieren Sie den vierten Alias in die vorhandene Textdatei *eigene-alias.txt*. Überschreiben Sie dabei nicht die bereits vorhandenen Daten.
7. Schließen Sie die PowerShell und öffnen Sie sie erneut. Importieren Sie Ihre eigenen Aliase aus der Datei *eigene-alias.txt*. Überprüfen Sie dann, ob die Aliase vorhanden sind.
8. Löschen Sie manuell die vier von Ihnen erstellten Aliase.

# 6 Profile



**Beispieldatei:** 6\_kompletter code.txt und profil.txt (Ordner Kapitel 06)

## 6.1 Profil als Gedächtnis der PowerShell

### Profile laden Definitionen beim Start

Wenn Sie in einer PowerShell-Sitzung auf der Kommandozeile Definitionen vornehmen, sind diese Definitionen nur in der aktuellen Sitzung gültig. Dies gilt gleichermaßen für Aliase, Variablen, eigene Cmdlets und Funktionen oder die von der PowerShell geführte Befehlshistorie.

- ▶ Definieren Sie in einer PowerShell-Sitzung eine Variable \$a und geben ihr den Wert 1:

```
$a = 1
```

- ▶ Überprüfen Sie durch Eingabe des Variablenamens den Wert der Variablen:

```
$a
```

Die PowerShell gibt als Wert 1 zurück.

- ▶ Schließen Sie die PowerShell und öffnen Sie anschließend die Anwendung erneut.
- ▶ Geben Sie erneut den Variablenamen ein.

In der PowerShell erfolgt nun keine Ausgabe. Die Variable ist nicht bekannt. Die Gültigkeit ist – wie bei anderen direkten Definitionen – auf die Dauer einer Sitzung beschränkt.

Sollen Ihre eigenen Definitionen permanent zur Verfügung stehen, können Sie sie in eine Datei exportieren und nach Bedarf wieder importieren. Oder aber Sie nehmen wichtige Definitionen in eine Profildatei auf, die beim Start einer jeden Sitzung geladen wird.

Häufige Aufgaben, die sich in einer Profildatei finden, sind:

- ✓ Definition eigener Aliase
- ✓ Einrichtung der Arbeitsumgebung wie Festlegung z. B. von Farben und Definition eines eigenen Prompts
- ✓ Anlegen eigener Laufwerke
- ✓ Importieren häufig verwendeter Module
- ✓ Definition häufig verwendeter Variablen und Funktionen

### Es existieren mehrere Profildateien

Profile sind eigentlich Skripte, die beim Start der PowerShell-Konsolenanwendung ausgeführt werden. Es werden insgesamt vier Profildateien unterstützt (sortiert nach der Rangfolge):

Profil	PowerShell-Profilvariable und Dateipfad
1) Aktueller Benutzer, aktueller Host	Variable: \$PROFILE.CurrentUserCurrentHost oder kurz \$PROFILE Pfad: \$HOME\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1
2) Aktueller Benutzer, alle Hosts	Variable: \$PROFILE.CurrentUserAllHosts Pfad: \$HOME\Documents\WindowsPowerShell\profile.ps1

Profil	PowerShell-Profilvariable und Dateipfad
3) Alle Benutzer, aktueller Host	<i>Variable:</i> \$PROFILE.AllUsersCurrentHost <i>Pfad:</i> \$PSHOME\Microsoft.PowerShell_profile.ps1
4) Alle Benutzer, alle Hosts	<i>Variable:</i> \$PROFILE.AllUsersAllHosts <i>Pfad:</i> \$PSHOME\profile.ps1

- ✓ Sie haben Einstellmöglichkeiten für den aktuellen Benutzer oder für alle Benutzer sowie für den aktuellen PowerShell-Host oder für alle PowerShell-Hosts (z. B. PowerShell-Konsole und PowerShell ISE).
- ✓ Die PowerShell-Variable \$HOME bezeichnet das Stammverzeichnis des aktuellen Benutzers.
- ✓ Die PowerShell-Variable \$PSHOME steht für das Installationsverzeichnis der PowerShell.
- ✓ PowerShell definiert Dateinamen und Speicherort der Profildateien, erstellt sie allerdings nicht selbst für Sie.
- ✓ Die in der Tabelle genannten Variablen zeigen auf den Speicherort des jeweiligen Profils, auch wenn die Datei nicht vorhanden sein sollte. Sie können die Variablen wie angegeben direkt in der PowerShell verwenden.
- ✓ Die Verarbeitung der Profildateien erfolgt von der untersten Tabellenzeile bis zur obersten (4, 3, 2, 1). Aus diesem Grund hat die oberste Profildatei Vorrang, da sie widersprüchliche Einträge überschreibt.
- ✓ Alle vier Profildateien werden beim Start gesucht und verarbeitet. Existiert eine Datei nicht, wird die Verarbeitung dieser Profildatei einfach übersprungen. Eine Fehlermeldung wird nicht generiert.

Lesen Sie in der Literatur vom gängigen PowerShell-Profil, ist das Profil „aktueller Benutzer, aktueller Host“ gemeint.

Auch die PowerShell ISE verfügt über zwei eigene Profildateien, die sich wie erläutert verhalten. In der folgenden Tabelle sehen Sie die entsprechenden Angaben:

Profil ISE	PowerShell-Profilvariable innerhalb PowerShell ISE und Dateipfad
1) Aktueller Benutzer, aktueller Host	<i>Variable:</i> \$PROFILE.CurrentUserCurrentHost oder kurz \$PROFILE <i>Pfad:</i> \$HOME\Documents\WindowsPowerShell\Microsoft.PowerShellISE_profile.ps1
2) Alle Benutzer, aktueller Host	<i>Variable:</i> \$PROFILE.AllUsersCurrentHost <i>Pfad:</i> \$PSHOME\Microsoft.PowerShellISE_profile.ps1

### Die richtige Profildatei auswählen

Die Wahl der passenden Profildatei für Ihre Definitionen hängt von Ihrer Arbeitsweise und Ihren Aufgaben ab. Sie können sich an folgenden Regeln orientieren:

- ✓ Nehmen Sie nur für sich selbst Einstellungen vor, wählen Sie einen Profiltyp für „aktuelle Benutzer“.
- ✓ Wollen Sie Definitionen für die Verwendung mehrerer Hostanwendungen vornehmen, verwenden Sie ein Profil, das für „alle Hosts“ gedacht ist.
- ✓ Einstellungen für eine spezielle Hostanwendung speichern Sie in einem Profil für den „aktuellen Host“.
- ✓ Allgemeine Definitionen legen Sie im Profil „Alle Benutzer, alle Hosts“ ab.

Für erste Tests empfiehlt sich, mit dem Standardprofil für den aktuellen Benutzer und den aktuellen Host zu experimentieren.

## 6.2 Anlegen eines Benutzerprofils

### Testen, ob das Profil vorhanden ist

Als Profildatei erwartet die PowerShell gemäß ihrer Definition eine bestimmte Datei in einem festgelegten Verzeichnis. Testen Sie zu Beginn, ob die Profildatei, mit der Sie arbeiten möchten, bereits vorhanden ist.

Für diese Aufgabe verwenden Sie das Cmdlet `Test-Path`, das bestimmt, ob alle Elemente eines Pfades vorhanden und u. a. folgende Parameter erlaubt:

Test-Path		
Parameter	Typ	Beschreibung
<b>-Path</b>	<b>P (1)</b>	Angabe des Pfades, der getestet werden soll. Platzhalter sind erlaubt.
-PathType	N	Prüft den Typ des letzten Elements der Pfadangabe. Stimmt der Typ des Elements mit dem Wert des Parameters überein, wird <code>TRUE</code> ausgegeben, ansonsten <code>FALSE</code> . Gültige Werte sind: <ul style="list-style-type: none"> <li>✓ <code>Container</code>: Element, das andere Elemente enthalten kann, z. B. Verzeichnis oder Registrierschlüssel</li> <li>✓ <code>Leaf</code>: Element, das keine anderen Elemente enthält, z. B. eine Datei</li> <li>✓ <code>Any</code>: Container- oder Leaf-Objekt</li> </ul>
-IsValid	S	Prüft, ob die Syntax des angegebenen Pfades korrekt ist. Dabei ist es unerheblich, ob die Elemente tatsächlich vorhanden sind. Ist die Syntax korrekt, gibt der Parameter <code>TRUE</code> zurück, andernfalls <code>FALSE</code> .
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: <code>Get-Help Test-Path -Full</code>

- ✓ Sind alle Elemente eines Pfades vorhanden, gibt das Cmdlet `TRUE` zurück, ansonsten `FALSE`. Bei einer Prüfung des Beispieldpfades `C:\Users\AD` werden einzeln `C:\`, `Users` und `AD` geprüft. Nur wenn alle Elemente vorhanden sind, erfolgt die Ausgabe `TRUE`.

Am Beispiel des Benutzerprofils für den aktuellen Benutzer, das auf den aktuellen Host angewendet wird, durchlaufen Sie alle Schritte bis zur Anwendung des Profils.

- Geben Sie folgende Zeile in der PowerShell-Konsole ein:

```
Test-Path -Path $PROFILE.CurrentUserCurrentHost
```

- ✓ Mit dieser Eingabe testen Sie, ob das Profil für den aktuellen Benutzer für den aktuellen Host existiert.
- ✓ Die PowerShell gibt daraufhin `TRUE` aus, wenn die Profildatei existiert, und `FALSE`, wenn sie nicht existiert.

Es ist Standard und an dieser Stelle anzunehmen, dass die Datei noch nicht existiert. Sollte die Datei dennoch bereits existieren, überspringen Sie den nächsten Schritt.

### Erstellen einer neuen Profildatei

Sie verwenden zum Erstellen eines Elements das Cmdlet `New-Item`. Dieses Cmdlet ist eines aus einer ganzen „Familie“ von Cmdlets, die verschiedene Aufgaben rund um Elemente (wie Erstellen, Löschen, Ändern, Kopieren, Auslesen etc.) erfüllen. `Get-Command -Noun Item` liefert Ihnen eine Liste dieser Cmdlets.

New-Item akzeptiert u. a. folgende Parameter:

New-Item		
Parameter	Typ	Beschreibung
-Path	P (1)	Pfadangabe des Speicherorts für das neue Element. Platzhalter sind erlaubt.
-Name	N	Angabe des Namens des neuen Elements, falls nicht bereits im Parameter -Path durch eine vollständige Pfadangabe angegeben
-Type	N	Angabe des Typs des neuen Elements, i. d. R. <i>file</i> oder <i>directory</i>
-Value	N	Angabe des Inhalts des neuen Elements, z. B. eine Zeichenfolge, die in eine Datei geschrieben wird
-Force	S	Gibt an, dass das neue Element ein vorhandenes – auch schreibgeschütztes – Element überschreiben darf
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help New-Item -Full

- ▶ Legen Sie die (leere) Profildatei durch die folgende Zeile in der PowerShell-Konsole an:

```
New-Item -Path $PROFILE.CurrentUserCurrentHost -Type file -Force
```

- ✓ Das standardmäßig fehlende Verzeichnis wird angelegt (Parameter -Force).
- ✓ Es wird eine Profildatei für das Profil „aktueller Benutzer, aktueller Host“ angelegt.

```
Windows PowerShell
PS C:\> New-Item -Path $PROFILE.CurrentUserCurrentHost -Type File -Force

Verzeichnis: C:\Users\AD\Documents\WindowsPowerShell

Mode          LastWriteTime    Length Name
----          -----        ---- 
-a---  04.08.2016     16:12         0 Microsoft.PowerShell_profile.ps1

PS C:\>
```

Rückmeldung der PowerShell bei Erstellung der Profildatei

### Bearbeiten der Profildatei

Da es sich bei Profildateien um einfache Textdateien handelt, können Sie einen beliebigen Texteditor wie z. B. den Windows-Editor (notepad.exe) zur Bearbeitung verwenden.

Die PowerShell beherrscht den Aufruf externer Programme, so dass Sie folgende Zeile eingeben können:

```
notepad.exe $PROFILE.CurrentUserCurrentHost
```

- ✓ Mit dieser Aktion öffnet sich der Windows-Editor mit der leeren Profildatei.

Nun können Sie im Editor zeilenweise Befehle für gewünschte Definitionen eingeben. Im folgenden Beispiel sehen Sie ein einfaches Beispiel für eine Profilskriptdatei.

### Beispiel: *profil.txt*

```

① Write-Host -Object "Hallo und viel Spaß mit der PowerShell,
$env:username!`n"
②     -ForegroundColor blue -BackgroundColor yellow
③ Set-Alias -Name paint C:\Windows\System32\mspaint.exe
# und etliche weitere Definitionen

```

- ① Das Cmdlet `Write-Host` sorgt für eine Bildschirmausgabe der Zeichenkette. Innerhalb der Zeichenkette erfolgt ein Zeilenumbruch durch die Sonderzeichensequenz ``n`. Voraussetzung dafür ist, dass die Zeichenkette von doppelten Anführungszeichen begrenzt wird, da sonst die Sonderzeichensequenz nicht ausgewertet, sondern einfach mit ausgegeben wird. Zudem erfolgt durch `$env:username` die Ausgabe des aktuellen Benutzernamens.
- ② Der Alias `paint` wird definiert. Microsoft Paint kann nun auch mithilfe des Kurzbefehls `paint` geöffnet werden.
- ③ Das Zeichen `#` sorgt dafür, dass die PowerShell die folgenden Zeichen (bis Zeilenende) nicht auswertet. Es handelt sich um einen Kommentar.

Wurde die Datei gespeichert, kommt es beim nächsten Start der PowerShell zu folgender Fehlermeldung:

The screenshot shows a Windows PowerShell window with the title bar 'Windows PowerShell'. The content of the window is as follows:

```

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

Lernen Sie das neue plattformübergreifende PowerShell kennen – https://aka.ms/pscore6

. : Die Datei
"C:\Users\Andreas\OneDrive\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1" kann
nicht geladen werden, da die Ausführung von Skripts auf diesem System deaktiviert ist. Weitere
Informationen finden Sie unter "about_Execution_Policies"
(https://go.microsoft.com/fwlink/?LinkID=135170).
In Zeile:1 Zeichen:3
+ . 'C:\Users\Andreas\OneDrive\Documents\WindowsPowerShell\Microsoft.Po ...
+ ~~~~~
+ CategoryInfo          : Sicherheitsfehler: () [], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess

```

Start der PowerShell bei vorhandener Profildatei: Skriptausführung deaktiviert

## 6.3 Hintergrund: Ausführungsrichtlinie für Skripte

Wie die Fehlermeldung in der obigen Abbildung richtig hinweist, ist anfangs auf einem System die Ausführung von Skripten blockiert. Da Skripte unter Umständen viel Schaden anrichten können, ist dies aus Sicherheitsgründen die Standardeinstellung.

Seit der PowerShell 4.0 hat sich die Standardeinstellung für die Ausführung von Skripten verändert. Auf Clientsystemen wie Windows 10 gilt wie zuvor, dass im Auslieferungszustand die Ausführung von Skripten blockiert ist. Unter Windows Server 2016 hingegen ist die Ausführung lokaler Skripte erlaubt (Einstellung `RemoteSigned`, siehe weiter unten).

## Herausfinden, welche Ausführungsrichtlinie gilt

Welche Ausführungsrichtlinie gilt, können Sie durch die Verwendung des Cmdlets `Get-ExecutionPolicy` herausfinden.

```
Get-ExecutionPolicy
```

- Die Ausgabe lautet bei einem Rechner mit dem Betriebssystem Windows 10, bei dem noch keine eigene Einstellung vorgenommen wurde: `Restricted`. Skripte dürfen nicht ausgeführt werden.

## Anwendungsbereiche und Einstellungen

Es existieren insgesamt fünf Ausführungsrichtlinien, wie Sie durch Verwendung des Parameters `-List` zusammen mit dem Cmdlet `Get-ExecutionPolicy` erkennen können:

```
Get-ExecutionPolicy -List
```

- Sie erhalten folgende Ausgabe:

Scope	ExecutionPolicy
-----	-----
MachinePolicy	Undefined
UserPolicy	Undefined
Process	Undefined
CurrentUser	Undefined
LocalMachine	Undefined

PowerShell geht die fünf Einstellungen (für fünf Anwendungsbereiche) von oben nach unten durch, bis eine Einstellung gefunden wird, die nicht `Undefined` lautet. Diese Einstellung ist dann die effektive Ausführungsrichtlinie, unabhängig von den weiteren Einstellungen in den Zeilen darunter. Im vorliegenden Fall stehen alle Einstellungen auf `Undefined`, in diesem Fall blockiert die PowerShell die Skriptausführung.

Zur Abfrage einer Einstellung eines bestimmten Anwendungsbereichs verwenden Sie `Get-ExecutionPolicy` zusammen mit dem Parameter `-Scope` und einem der Werte `MachinePolicy`, `UserPolicy`, `Process`, `CurrentUser` und `LocalMachine`.

Die Anwendungsbereiche bedeuten im Einzelnen:

Bereich	Erläuterung
MachinePolicy	Wird von einer Gruppenrichtlinie für den Computer festgelegt und damit auf alle Benutzer des Rechners angewendet; kann nicht manuell angepasst werden
UserPolicy	Wird von einer Gruppenrichtlinie für den aktuellen Benutzer festgelegt; kann nicht manuell angepasst werden
Process	Ist nur für die aktuelle PowerShell-Sitzung gültig, z. B. wenn Sie testen wollen
CurrentUser	Die Ausführungsrichtlinie stellt der aktuelle Benutzer selbst in der PowerShell ein.
LocalMachine	Basiseinstellung für alle Benutzer

Beachten Sie, dass die Tabellenzeilen nach Priorität sortiert sind. Liegt eine Einstellung für den Bereich `LocalMachine` vor, wird sie durch eine Einstellung eines einzelnen Users (`CurrentUser`) überschrieben.



Für alle Bereiche können Sie aus den folgenden Einstellungen wählen:

Einstellung	Erläuterung
Restricted	Standardwert; es werden keine Skripte ausgeführt. Die PowerShell kann nur interaktiv verwendet werden.
AllSigned	Jedes Skript muss von einem vertrauenswürdigen Herausgeber signiert sein.
RemoteSigned	Alle Skripte fremder Quellen müssen digital von einem vertrauenswürdigen Herausgeber signiert sein. Lokale Skripte werden ausgeführt.
Unrestricted	Skripte werden ausgeführt. Bei Skripten nicht vertrauenswürdiger Quellen erfolgt eine Abfrage.
Bypass	Skripte werden ohne Blockierung, Nachfragen oder Warnungen ausgeführt.
Undefined	Setzt die Einstellung auf einen undefinierten Stand (zurück); funktioniert nicht bei Ausführungsrichtlinien, die durch Gruppenrichtlinien vorgegeben werden

### Ausführungsrichtlinie ändern

Sie wollen erreichen, dass die erstellte Profildatei beim Start der PowerShell ausgeführt wird. In späteren Kapiteln sollen auch andere lokale Skripte ausgeführt werden. Die vorgeschlagene Einstellung lautet *RemoteSigned* für den Bereich des aktuellen Benutzers (*CurrentUser*).

Zu diesem Zweck verwenden Sie das Cmdlet `Set-ExecutionPolicy`, das Sie u. a. mit folgenden Parametern verwenden können:

Set-ExecutionPolicy		
Parameter	Typ	Beschreibung
<code>-ExecutionPolicy</code>	P (1)	Definiert eine neue Ausführungsrichtlinie mit den oben gezeigten Einstellungen als möglichen Werten
<code>-Scope</code>	N	Gibt den Anwendungsbereich der Ausführungsrichtlinie an. Die möglichen Werte sind in der obigen Tabelle angegeben.
<code>-Force</code>	S	Unterdrückt die Standard-Nachfrage, ob die Richtlinie wirklich geändert werden soll
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: <code>Get-Help Set-ExecutionPolicy -Full</code>

- ▶ Öffnen Sie eine PowerShell-Konsole als Administrator.  
Sie benötigen – bis auf die Einstellungen für den aktuellen Benutzer – erhöhte Rechte für die beabsichtigte Änderung an der Ausführungsrichtlinie. Andernfalls erhalten Sie eine Fehlermeldung, die auf mangelnde Rechte hinweist ("Der Zugriff [...] wurde verweigert .").
- ▶ Geben Sie die folgende Zeile ein, um dem aktuellen Benutzer zu ermöglichen, lokale Skripte auszuführen:

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
```

- ✓ In der nachfolgenden Abbildung sehen Sie das Verhalten der PowerShell. Die Titelleiste zeigt, dass die PowerShell als Administrator ausgeführt wird. Noch erhalten Sie eine Fehlermeldung bei dem Versuch, das Profilskript automatisch beim Start auszuführen. Nach Eingabe der Befehlszeile erfolgt eine Nachfrage, ob Sie die Aktion wirklich ausführen wollen. Die Nachfrage können Sie durch zusätzliche Angabe des Switch-Parameters `-Force` unterdrücken.
- ✓ Eine weitere Bestätigung vonseiten der PowerShell erfolgt nicht.

```

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

Lernen Sie das neue plattformübergreifende PowerShell kennen - https://aka.ms/pscore6

. : Die Datei
"C:\Users\Andreas\OneDrive\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1" kann
nicht geladen werden, da die Ausführung von Skripts auf diesem System deaktiviert ist. Weitere
Informationen finden Sie unter "about_Execution_Policies"
(https://go.microsoft.com/fwlink/?LinkID=135170).
In Zeile:1 Zeichen:3
+ . 'C:\Users\Andreas\OneDrive\Documents\WindowsPowerShell\Microsoft.Po ...
+ ~~~~~
+ CategoryInfo          : Sicherheitsfehler: () [], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\Andreas> Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser

Ausführungsrichtlinie ändern
Die Ausführungsrichtlinie trägt zum Schutz vor nicht vertrauenswürdigen Skripts bei. Wenn Sie die
Ausführungsrichtlinie ändern, sind Sie möglicherweise den im Hilfethema "about_Execution_Policies"
unter "https://go.microsoft.com/fwlink/?LinkID=135170" beschriebenen Sicherheitsrisiken ausgesetzt.
Möchten Sie die Ausführungsrichtlinie ändern?
[J] Ja [A] Ja, alle [N] Nein [K] Nein, keine [H] Anhalten [?] Hilfe (Standard ist "N"):
```

Anpassen der Ausführungsrichtlinie von Skripten für den aktuellen Benutzer

- ▶ Beenden Sie die aktuelle PowerShell-Sitzung und starten Sie die PowerShell erneut.

```

Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. Alle Rechte vorbehalten.

Hallo und viel Spaß mit der PowerShell, AD!

Das Laden von persönlichen und Systemprofilen dauerte 1634 ms.
PS C:\Users\AD> Get-Alias -Name paint

 CommandType      Name                                     Version
-----          ----
 Alias           paint -> mspaint.exe

PS C:\Users\AD> Get-ExecutionPolicy -List

 Scope ExecutionPolicy
----- -----
 MachinePolicy   Undefined
 UserPolicy     Undefined
 Process        Undefined
 CurrentUser    RemoteSigned
 LocalMachine   Undefined

PS C:\Users\AD>
```

Bestätigung, dass die Profildatei abgearbeitet wurde und lokale Skripte ausgeführt werden dürfen

- ✓ Die Eingangsmeldung aus dem Beispiel-Profilskript wird angezeigt.
- ✓ Der Alias *paint* ist vorhanden. Die Überprüfung erfolgt durch die Eingabe von `Get-Alias -Name paint`.
- ✓ Die Ausführungsrichtlinien für Skripte wird mit `Get-ExecutionPolicy -List` angezeigt. Wie erwartet, zeigt der Bereich *CurrentUser* die Einstellung *RemoteSigned*.

Das Anpassen der Ausführungsrichtlinie für Skripte muss in der PowerShell nur einmalig vorgenommen werden, da PowerShell diese Einstellung (permanent) in die Registry schreibt. Ab diesem Zeitpunkt erfolgt bei jedem Start der PowerShell die Ausführung der vorhandenen Profildateien.



Die vorgenommene Änderung betrifft nicht nur Profildateien, sondern die Ausführung von Skripten allgemein. Auch wenn Sie momentan keine Verwendung für ein Profilskript haben sollten, ändern Sie die Ausführungsrichtlinie für Skripte entsprechend ab, damit Sie in den nächsten Kapiteln problemlos mit Skripten arbeiten können.

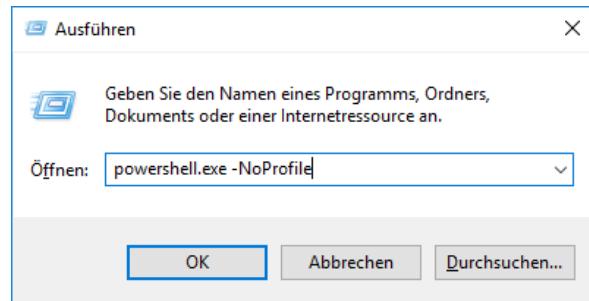
## PowerShell ohne Profil starten

Wollen Sie die PowerShell starten, vorhandene Profile aber nicht ausführen, können Sie die PowerShell mit dem Parameter `-NoProfile` starten.

Drücken Sie die Tasten zum Öffnen des Ausführen-Dialogs und geben Sie `powershell.exe -NoProfile` ein.

Es stehen weitere interessante Parameter für den Start der PowerShell zur Verfügung. Informieren Sie sich bei Bedarf über die Möglichkeiten unter [https://docs.microsoft.com/de-de/powershell/module/microsoft.powershell.core/about/about\\_powershell\\_exe](https://docs.microsoft.com/de-de/powershell/module/microsoft.powershell.core/about/about_powershell_exe).

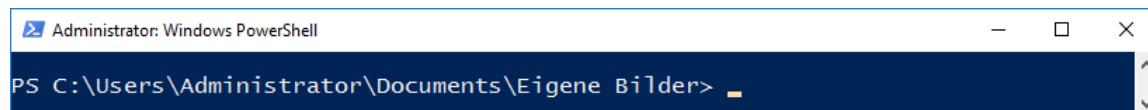
An dieser Stelle sei auf den Parameter `-ExecutionPolicy` hingewiesen, mit dem Sie die Ausführungsrichtlinie für die aktuelle PowerShell-Sitzung steuern.



Starten der PowerShell ohne Profil

## Anwendungsbeispiel: Definition eines eigenen Prompts

Der Prompt (die „Eingabeaufforderung“) der PowerShell zeigt standardmäßig den Pfad des aktuellen Ordners an. Je nach Fensterbreite und Position im Dateisystem verwendet die PowerShell einen großen Teil der zur Verfügung stehenden Zeilenbreite zur Anzeige des Prompts.



Vorher: Stecken Sie tief im Dateisystem, nimmt der Prompt viel Platz ein.

Um einen kurzen Prompt zu erhalten und dennoch nicht auf den Luxus der Anzeige der aktuellen Position im Dateisystem zu verzichten, können Sie die folgende Zeile in der PowerShell eingeben:

```
function prompt { 'Shell> ' ; $Host.UI.RawUI.WindowTitle = Get-Location }
```

- ✓ Die Funktion mit dem vordefinierten Namen `prompt`, legt das Verhalten der Eingabeaufforderung fest (und überschreibt in diesem Fall den vom System vordefinierten Standard). Die Funktion wird jeweils nach Fertigstellung eines Befehls erneut ausgeführt. Weitere Informationen zu Funktionen erhalten Sie in Kapitel 8.
- ✓ In einfachen Anführungszeichen wird ein statischer Text definiert, der als Prompt erscheinen soll.
- ✓ Ein Semikolon wird verwendet, damit Sie mehrere Befehle hintereinander ohne Zeilenumbruch schnell in der Konsole verwenden können.
- ✓ Die Variable `$Host.UI.RawUI.WindowTitle` legt das Aussehen der Titelzeile fest. Als Wert wird über `Get-Location` das jeweils aktuelle Verzeichnis zugewiesen.



Nachher: Die Position im Dateisystem steht im Titel, der Prompt besteht aus einem kurzen, statischen Text.

Wenn Sie diese Beispielzeile in Ihr Profilskript einfügen, führt die PowerShell die Funktion bei jedem Start aus. Damit verändern Sie den Standard-Prompt für jede Sitzung der PowerShell.

## 6.4 Kurz zusammengefasst

### Eigene Definitionen dauerhaft zur Verfügung stellen

Profile runden die Arbeit mit der PowerShell ab. Mit ihrer Hilfe „retten“ Sie Ihre individuellen Definitionen von Sitzung zu Sitzung. Sie stehen nunmehr stets zur Verfügung. Mögliche Inhalte der Profildateien decken verschiedene Einsatzgebiete ab, die sich an Ihrer individuellen Arbeitsweise orientieren können.

Sie bereiten einmalig Ihren Rechner für den Einsatz von (lokalen) Skripten – und damit auch Profilen – vor. Dabei haben Sie die Wahl, Ausführungsrichtlinien für Skripte an Ihre Bedürfnisse anzupassen.

Kehren Sie nach Abschluss des kompletten Buchs nochmals zu dieser Thematik zurück. Mit wachsender Erfahrung im Umgang mit der PowerShell ergeben sich fast automatisch Ideen für Ihre persönlichen Profile.

## 6.5 Übung

### Mit Benutzerprofilen arbeiten

Übungsdatei: –

Ergebnisdatei: 6\_ergebnisse.txt

1. Lesen Sie die Skript-Ausführungsrichtlinie für Ihren Rechner aus.
2. Ändern Sie die Ausführungsrichtlinie nur für Sie persönlich auf die Einstellung *RemoteSigned* und überprüfen Sie, ob die Änderungen erfolgreich vorgenommen wurden. Erinnern Sie sich an die nötigen Arbeitsschritte?
3.
  - a) Prüfen Sie, ob eine Profildatei für den aktuellen Benutzer und Host vorhanden ist.
  - b) Erstellen Sie bei Bedarf eine neue Datei und öffnen Sie sie zur Bearbeitung.
  - c) Füllen Sie die Datei mit Definitionen Ihrer Wahl und speichern Sie die Datei ab.
  - d) Wie prüfen Sie, ob die Einstellungen angewendet werden?
4. Vorausgesetzt, Sie verwenden eine oder mehrere Profildateien: Wie starten Sie die PowerShell ohne Verarbeitung von Profildateien?
5. Welche Auswirkungen auf Ihre Profile hat es, wenn Sie die PowerShell über den Befehl `powershell.exe -ExecutionPolicy Restricted` starten?

# 7 Programmiergrundlagen



Beispieldatei: 7\_kompletter code.txt (Ordner Kapitel 07)

## 7.1 Variablen

### Mit Variablen arbeiten

Variablen dienen dazu, Informationen für die spätere Verwendung zu speichern. Besonders nützlich ist die Speicherung von Werten für die spätere Verwendung im Rahmen der Programmierung, um flexible Skripte zu erstellen, die wechselnden Bedingungen entsprechen sollen.

Der Datentyp einer Variablen ergibt sich bei der PowerShell automatisch durch den Datentyp des Wertes, der der Variablen zugewiesen wird. Sie müssen nicht vor der Verwendung einer Variablen – wie bei anderen Programmiersprachen üblich – einen Datentyp definieren und sich daran halten. Der Datentyp einer Variablen kann im Betrieb geändert werden, ohne dass es zu einem Fehler kommt.

Eine Variable in der PowerShell besteht aus einer Zeichenfolge mit vorangestelltem Dollarzeichen. Es wird nicht zwischen Groß- und Kleinschreibung unterschieden. Die Zeichenkette kann aus Zahlen, Buchstaben und Sonderzeichen bestehen. Die Verwendung von Sonderzeichen und dem Bindestrich wird allerdings nicht empfohlen. \$x, \$service, \$MeineEinstellung oder \$meine\_einstellung sind typische Variablennamen.

### Variablentypen in der PowerShell

Auch wenn Sie selbst noch keine eigenen Variablen definiert haben, arbeitet die PowerShell bereits mit einer Reihe vordefinierter Variablen. Die PowerShell kennt insgesamt drei Typen von Variablen:

- ✓ Einstellungsvariablen
- ✓ Automatische Variablen
- ✓ Benutzerdefinierte Variablen

### Einstellungsvariablen

In diesen Variablen sind Benutzereinstellungen für die PowerShell gespeichert. Sie steuern das Standardverhalten der PowerShell. Die Variablen werden automatisch erstellt und mit Standardwerten versehen, die aber von Ihnen verändert werden können.

Einige Beispiel-Einstellungsvariablen finden Sie in der folgenden Tabelle:

Variable	Funktion
\$ErrorActionPreference	Steuert das Verhalten im Falle eines Fehlers, der die Ausführung des Cmdlets nicht beendet. Der Standardwert <i>Continue</i> erlaubt eine Fortsetzung der Cmdlet-Ausführung. Sie können alternativ ein Anhalten der Ausführung ( <i>Stop</i> ), eine Benutzernachfrage über das weitere Vorgehen ( <i>Inquire</i> ) oder ein Fortsetzen ohne Anzeige der Fehlermeldung ( <i>SilentlyContinue</i> ) voreinstellen. Änderungsbeispiel: <code>\$ErrorActionPreference = SilentlyContinue</code>
\$MaximumAliasCount	Legt fest, wie viele Aliase Sie in einer PowerShell-Sitzung maximal festlegen können. Der Standardwert ist <i>4096</i> . Änderungsbeispiel: <code>\$MaximumAliasCount = 512</code>

Variable	Funktion
\$MaximumHistoryCount	Legt fest, wie viele eingegebene Befehle der aktuellen Sitzung in einer Liste gespeichert werden. Der Standardwert ist ebenfalls 4096. Änderungsbeispiel: <code>\$MaximumHistoryCount = 10000</code>
\$WhatIfPreference	Steuert das Verhalten von Cmdlets, die den Parameter –WhatIf unterstützen <ul style="list-style-type: none"> <li>✓ Der Standardwert 0 legt fest, dass <i>die Funktion</i> des Parameters –WhatIf nicht automatisch aktiviert ist. Zur Verwendung müssen Sie den Parameter –WhatIf manuell mit einem Cmdlet verwenden.</li> <li>✓ Der Wert 1 legt fest, dass automatisch alle Cmdlets, die den Parameter unterstützen, ihre Ausführung nur simulieren und eine entsprechende Meldung anzeigen. Wollen Sie bei der Einstellung diese Cmdlets wirklich ausführen, müssen Sie –WhatIf:\$false manuell mit dem Cmdlet verwenden.</li> </ul> Änderungsbeispiel: <code>\$WhatIfPreference = 1</code>

Genauere Informationen erhalten Sie im Internet unter der Adresse [https://docs.microsoft.com/de-de/powershell/module/microsoft.powershell.core/about/about\\_preference\\_variables](https://docs.microsoft.com/de-de/powershell/module/microsoft.powershell.core/about/about_preference_variables) oder durch Eingabe des Befehls:

```
Get-Help about_Preference_Variables
```

### Automatische Variablen

Die PowerShell speichert in diesen Variablen Statusinformationen. Die Variablen werden von der PowerShell automatisch erstellt und deren Werte bei Bedarf von der PowerShell verändert. Die Werte dieses Variablen Typs können von Ihnen nicht verändert werden.

Einige Beispiel-Variablen dieser Kategorie finden Sie in der folgenden Tabelle:

Variable	Funktion
\$?	Zeigt den Status der letzten Befehlausführung. Der Rückgabewert TRUE deutet auf eine erfolgreiche Ausführung hin, der Rückgabewert FALSE auf einen Fehlschlag.
\$Error	Protokolliert die Fehlermeldungen der aktuellen PowerShell-Sitzung. Bei den Variablen handelt es sich um ein Array, einen speziellen Variablen Typ, der beliebig viele Werte speichern kann. Mit \$Error [0] erhalten Sie die letzte Fehlermeldung, mit \$Error [1] die vorletzte usw. Mit \$Error .count können Sie die Anzahl der gespeicherten Fehlermeldungen ermitteln.
\$HOME	Enthält die Pfadangabe des Benutzerverzeichnisses, üblicherweise C:\Users\<Benutzername>

Genauere Informationen erhalten Sie im Internet unter der Adresse [docs.microsoft.com/de-de/powershell/module/microsoft.powershell.core/about/about\\_automatic\\_variables](https://docs.microsoft.com/de-de/powershell/module/microsoft.powershell.core/about/about_automatic_variables) oder durch Eingabe des folgenden Befehls:

```
Get-Help about_Automatic_Variables
```

### Benutzerdefinierte Variablen

Diesen Typ von Variablen erstellen und verwalten Sie selbst. Standardmäßig sind die in einer PowerShell-Sitzung erstellten Variablen nur in dieser Sitzung gültig. Am häufigsten werden Sie mit Variablen in Verbindung mit der Erstellung eigener Skripte arbeiten.

In diesem und den nachfolgenden Kapiteln lernen Sie den Umgang mit Variablen und erstellen und verwalten eigene Variablen.

## Vorhandene Variablen anzeigen

Sie können das Cmdlet Get-Variable verwenden, um eine Variablen-Liste der aktuellen PowerShell-Sitzung anzeigen zu lassen. Standardmäßig wird eine Tabelle mit den Spalten für den Namen und den Wert der Variablen angezeigt.

Das Cmdlet Get-Variable wird ohne Parameter verwendet, um alle Variablen anzeigen zu lassen. Verwenden Sie das Cmdlet mit dem Parameter –Name, werden nur die dort spezifizierte(n) Variable(n) angezeigt. Sie dürfen Platzhalter verwenden bzw. mehrere Variablennamen – durch Komma getrennt – angeben:

```
Get-Variable [-Name]
```

- ✓ z. B.: Get-Variable
- ✓ z. B.: Get-Variable -Name ps\_test
- ✓ z. B.: Get-Variable -Name ps\_test, test, s\*

Eine weitere Möglichkeit, Variablen anzuzeigen, bietet der PowerShell-Provider *Variable*. Der Provider bietet das PowerShell-Laufwerk **Variable:**, das Sie wie ein Laufwerk im Dateisystem verwenden können:

```
Get-ChildItem -Path Variable:
```

- ✓ In einer flachen Struktur innerhalb des Laufwerks **Variable:** sind alle Variablen der aktuellen PowerShell-Sitzung als Einzelobjekte verfügbar.



Wenn Sie weitere Eigenschaften einer Variablen sehen, verwenden Sie in der Pipeline einen Ausgabebefehl, der alle verfügbaren Eigenschaften anzeigt (und nicht nur die von der PowerShell vorausgewählten), wie z. B.:

```
Get-Variable -Name PSUICulture | Format-List -Property *
```

- ✓ Informationen zur Variablen **PSUICulture** werden über die Pipeline an das Cmdlet **Format-List** weitergereicht. Der Parameter **-Property** mit seinem Wert „\*“ (für „alle“) sorgt dafür, dass alle verfügbaren Eigenschaften der Variablen angezeigt werden.

## Variablen auf einfachem Weg erstellen

Die PowerShell kennt zwei Wege, um eine neue Variable zu erstellen. Am einfachsten geben Sie den gewünschten Variablenamen ein und initialisieren die Variable. Das bedeutet, dass Sie der Variablen gleich einen Wert zuweisen. Die Wertzuweisung geschieht mithilfe des Gleichheitszeichens **=**. Dieses Zeichen wird nicht wie in der Schulmathematik bei der Frage nach Gleichheit verwendet, sondern bei der Zuweisung eines Wertes (auf der rechten Seite) zu einer Variablen (auf der linken Seite). Aus diesem Grund wird das Zeichen als Zuweisungsoperator bezeichnet:

### Beispiel

```
$test = 45.3
```

- ✓ Sie schreiben den Namen der Variablen mit führendem Dollarzeichen in die Konsole. Durch den Zuweisungsoperator **=** erfolgt die Zuweisung des angegebenen Wertes **45,3**.
- ✓ Als Dezimaltrennzeichen für Zahlwerte mit Nachkommastellen ist in der PowerShell das Zeichen **.** zu verwenden.
- ✓ Wenn Sie Variablen auf diese Weise definieren, müssen Sie eine Wertzuweisung vornehmen. Verwenden Sie in einer Zeile nur den Variablenamen, passiert nichts, falls die Variable nicht bereits definiert ist. Es wird in dem Fall nichts ausgegeben, auch keine Fehlermeldung.

```
$test
```

- ✓ Ist die Variable bereits definiert, wird ihr Wert in der Konsole ausgegeben.

## Die PowerShell-Syntax zur Erstellung von Variablen verwenden

Neben der vorgestellten einfachen Methode zum Erstellen von Variablen beherrscht die PowerShell auch die Verwendung des Cmdlets `New-Variable` zur Erstellung von Variablen. Mit dieser Methode sind Sie durch den möglichen Einsatz von Parametern flexibler:

New-Variable		
Parameter	Typ	Beschreibung
<code>-Name</code>	<code>P(1)</code>	Angabe des Namens der Variablen (ohne führendes Dollarzeichen)
<code>-Value</code>	<code>P(2)</code>	Legt den Anfangswert der spezifizierten Variablen fest
<code>-Description</code>	<code>N</code>	Eingabe einer Beschreibung für die Variable
<code>-Force</code>	<code>S</code>	Erzwingt die Erstellung der neuen Variablen, auch wenn bereits eine Variable dieses Namens vorhanden ist; überschreibt zudem schreibgeschützte Variablen
<code>-Option</code>	<code>N</code>	Legt zusätzliche Optionen fest. Mögliche Werte sind u. a.: <ul style="list-style-type: none"> <li>✓ None: Keine Option wird definiert (Standardwert).</li> <li>✓ ReadOnly: Kann gelöscht werden, der Wert der Variablen kann nur mithilfe des Parameters <code>-Force</code> verändert werden.</li> </ul> Constant: Verhindert Löschung oder Änderung der Variablen. Kann nur bei Erstellung einer Variablen verwendet werden. Für bereits existierende Variablen können Sie die Option nicht setzen.
<code>-PassThru</code>	<code>S</code>	Definiert nicht nur die Variable, sondern gibt auch zusätzlich Informationen zur Variablen in der Konsole aus
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: <code>Get-Help -Name New-Variable -Full</code>

Wenn Sie einer neuen Variablen keinen Wert zuweisen, erhält die Variable standardmäßig von der PowerShell den Wert `$null` (leer).

### Beispiele

- 1) Erstellen einer Variablen mithilfe des Cmdlets `New-Variable` (Minimaleingabe):

```
New-Variable -Name ps_test
```

- ✓ Sie verwenden das Cmdlet `New-Variable` mit dem Parameter `-Name`, um den Namen einer neuen Variablen festzulegen.
- ✓ Die neue Variable erhält keinen Anfangswert. Wenn Sie in der Konsole `$ps_test` eingeben, erfolgt keine Ausgabe.

Beachten Sie, dass selbst erstellte Variablen nur in der aktuellen PowerShell-Sitzung gültig sind. Sobald Sie die aktuelle PowerShell-Sitzung beenden, stehen die von Ihnen definierten Variablen nicht weiter zur Verfügung. Sie können wichtige Variablen, die laufend verfügbar sein sollen, z. B. in Ihrer persönlichen Profildatei definieren.



- 2) Eine Variable soll bei der Erstellung initialisiert werden:

```
New-Variable -Name ps_test2 -Value "Herdt-Verlag"
```

- ✓ Die Variable `$ps_test2` erhält bei ihrer Definition den Ausgangswert `Herdt-Verlag`.

Bei der Wertzuweisung (Parameter `-Value`) müssen Sie beachten, dass PowerShell den angegebenen Wert als Zeichenkette versteht, wenn es sich nicht um eine Ziffer handelt. Die folgende Tabelle zeigt an einigen Beispielen das Verhalten bei der Wertzuweisung:

Beispielwert <code>-Value</code>	Wert der Variablen	Erläuterung
"Herdt-Verlag"	Herdt-Verlag	Der Wert ist als Zeichenkette gekennzeichnet und wird entsprechend übernommen.
37 * 11	Fehler	Die PowerShell sucht vergeblich nach Positionsparametern, die das Argument „*“ unterstützen. Durch die Leerzeichen vermutet die PowerShell an dieser Stelle die Übergabe von drei Argumenten.
37*11	37*11	PowerShell erkennt den Wert als Zeichenkette.
(37 * 11)	407	Durch die Klammersetzung ist PowerShell angehalten, erst das Ergebnis der Multiplikation zu bilden. Das Ergebnis 407 wird dann als Zahlwert der Variablen zugewiesen.
Get-Date	Get-Date	Eine Auswertung des Cmdlets findet nicht statt, es wird als Zeichenfolge interpretiert.
(Get-Date)	z.B.: Sonntag, 23. Mai 2021 14:31:02	Durch die Klammersetzung wird das Cmdlet Get-Date ausgeführt. Es ergibt sich damit als Wert der Variablen das aktuelle Datum.

## Wert einer Variablen ändern

Wenn Sie den Wert einer Variablen ändern wollen, können Sie das durch eine erneute Wertzuweisung nach folgendem Muster erreichen:

```
$<Variablename> = (neuer) Wert
```

- ✓ Das Vorgehen ist identisch zu dem, als Sie eine neue Variable erstellt haben. Der Unterschied besteht nur darin, dass bei einer Wertänderung die Variable bereits definiert ist. Ist die Variable nicht bekannt, wird sie erstellt.

Alternativ dazu bietet die PowerShell auch hier ein eigenes Cmdlet: `Set-Variable`. Das Cmdlet unterstützt dieselben Parameter wie das weiter oben vorgestellte Cmdlet `New-Variable`.

## Beispiel

- 1) Der Wert der Variablen `$ps_test2` soll verändert werden:

```
Set-Variable -Name ps_test2 -Value "Bodenheim"
```

- ✓ Falls Sie in der aktuellen PowerShell-Sitzung bereits die Variable `$ps_test` definiert hatten, erfolgt nun eine neue Wertzuweisung.
- ✓ Ist die Variable noch nicht vorhanden, wird sie angelegt und mit dem angegebenen Wert versehen.

## Variablen löschen

Die PowerShell bietet Cmdlets zum Löschen von Variablen: zum einen `Clear-Variable` zum Löschen des Wertes einer Variablen, zum anderen `Remove-Variable` zum vollständigen Entfernen der Variablen inklusive ihres Werts.

Bei beiden Cmdlets wird der Positionsparameter `-Name` verwendet, um den Namen der Variablen anzugeben. Hier sind auch Platzhalter erlaubt, so dass Sie die Aktion auf mehrere Variablen gleichzeitig anwenden können.

### Beispiele

Sie haben bereits eine Variable `$a` definiert und ihr den Wert 5 zugewiesen (`$a = 5`) sowie eine Variable `$b` mit Wert 10 (`$b = 10`).

- 1) Sie möchten den Wert der Variablen `$a` löschen, nicht aber die Variable selbst:

```
Clear-Variable -Name a
```

- ✓ Mit diesem Cmdlet löschen Sie den Wert der Variablen `$a`. Die Variable selbst bleibt erhalten, ist aber leer.

Alternativ zu dieser Vorgehensweise können Sie folgenden Befehl eingeben:

```
$a = $null
```

- ✓ Das Ergebnis ist identisch: Die Variable `$a` bleibt erhalten, hat aber keinen zugewiesenen Wert.

- 2) Sie möchten die Variable `$b` komplett löschen:

```
Remove-Variable -Name b
```

Auch hier existiert eine Alternative, dieses Ziel zu erreichen. Sie löschen das Objekt `b` auf dem Laufwerk `Variable:` (des Providers `Variable`):

```
Remove-Item Variable:\b
```

## Typisierte Variablen verwenden

Wie bisher vorgestellt, können in der PowerShell Variablen verwendet werden, bei denen Sie nicht explizit einen Datentyp für die Variable angeben. Die PowerShell kann aus eingegebenen Werten den Typ ableiten, der dann verwendet wird.

Sie können auch selbst explizit eine Variable typisieren (d. h., den Typ vorgeben). Das bedeutet, dass diese Variable nur noch Werte des angegebenen Typs akzeptiert. Soll die Variable einen Wert eines anderen Datentyps speichern, versucht sie implizit eine Umwandlung in den vorgegebenen Variablentyp. Sollte dies nicht möglich sein, wird ein Fehler ausgegeben.

Die folgende Tabelle zeigt einige wichtige Variablentypen, die bei der Typisierung zur Verfügung stehen:

Typ	Beschreibung
[int]	Ganzzahlwert (32 bit)
[long]	Ganzzahlwert (64 bit)
[string]	Zeichenfolge (Unicode)
[char]	16-bit-Zeichen (Unicode)

Typ	Beschreibung
[byte]	Zeichen (8 bit)
[bool]	Boolscher Wert (True/False-Wert)
[decimal]	128-bit-Dezimalwert
[single]	Gleitkommazahl (32 bit)
[double]	Gleitkommazahl (64 bit)
[xml]	XML-Objekt
[array]	Array
[hashtable]	Hashtable

Sie legen einen Variabtentyp fest, indem Sie den Variabtentyp aus der oben stehenden Tabelle direkt vor die Variable schreiben, der Sie diesen Typ zuweisen wollen.

### Beispiele

```
[int]$zahl = 42
```

- ✓ Als Variabtentyp wird ein Ganzahlwert vorgegeben. Die Wertzuweisung mit dem Zahlwert 42 entspricht dieser Vorgabe.

```
[int]$zahl = "42"
```

- ✓ Auch diese Definition kann durchgeführt werden. Es wird zwar einer als Ganzahl definierten Variablen eine Zeichenkette zugewiesen, sie kann aber von der PowerShell in einen Ganzahlwert umgewandelt werden.

```
[int]$zahl = "Tokio"
```

- ✗ Da die Zeichenkette Tokio nicht in einen Ganzahlwert umgewandelt werden kann, erhalten Sie eine Fehlermeldung: Der Wert "Tokio" kann nicht in den Typ "System.Int32" konvertiert werden. Fehler: "Die Eingabezeichenfolge hat das falsche Format."



Zur Prüfung, ob eine Variable einem bestimmten Variabtentyp entspricht, können Sie den Operator `-Is` einsetzen. Als Gegenstück existiert ein Operator `- IsNot`, der prüft, ob eine Variable dem angegebenen Typ nicht entspricht. Ist die Prüfung erfolgreich, erhalten Sie von der PowerShell den Rückgabewert `TRUE`, anderenfalls `FALSE`. Für das obige Beispiel ergibt `$zahl -Is [int]` die Rückgabe `TRUE`, da Sie diesen Variabtentyp festgelegt haben.

## 7.2 Arrays – spezielle Variablen mit Wertelisten

Mit Variablen können Sie einzelne Werte speichern. Eine spezielle Art der Variablen sind Arrays (auch Felder oder Feldvariablen genannt), die eine Werteliste (gleichen oder unterschiedlichen Datentyps) aufnehmen können. Der Zugriff auf die einzelnen Werte erfolgt über einen Index.

Beispielsweise erhalten Sie in einer Berechnung 150 Ergebnisse. Die Ausgabe oder Weiterverarbeitung der Werte soll erst erfolgen, wenn alle Berechnungen abgeschlossen wurden. Sie könnten 150 Variablen (`$ergebnis1`, `$ergebnis2`, `$ergebnis3`, `$ergebnis4` ...) definieren, um alle Ergebnisse der Berechnungen zu speichern. Das ist sehr umständlich und arbeitsaufwendig, aussagekräftige Bezeichnungen sind in größerer Anzahl schwer zu realisieren. Einfacher und schneller kann die Aufgabe gelöst werden, wenn die Ergebnisse der Berechnung in einem Array abgelegt werden.

## Eine Array-Variable erstellen

Zum Erstellen eines Arrays gehen Sie genauso vor wie bei der Erstellung einer einfachen Variablen. Im Unterschied dazu weisen Sie der Variablen mehrere, durch Komma voneinander getrennte Werte zu.

```
$a_var = 2, 66, 43, 5.4, 13, "Wolpertinger", 307
```

```
New-Variable -Name a_var -Value 2, 66, 43, 5.4, 13, "Wolpertinger", 307
```

- ✓ Beide Varianten zur Erstellung von Variablen sind bereits bekannt. In den Beispielen werden mehrere Werte der Variablen zugewiesen und damit eine Array-Variable erstellt.
- ✓ Verwenden Sie `$a_var.Count`, um die Anzahl der Werte zu bestimmen, die die Variable enthält.

Es gibt noch einen weiteren Weg, wie Sie eine Array-Variable definieren können:

```
$<name> = @( [Wert] )
```

- ✓ Sie geben den Variablenamen und den Zuweisungsoperator „=“ an.
- ✓ Es folgt das Zeichen `@` und ein rundes Klammernpaar. Das Zeichen `@` wird in diesem Zusammenhang als *Array-Operator* bezeichnet.
- ✓ In den Klammern können Werte eingetragen werden, die der Array-Variablen zugewiesen werden. Sie können keinen, einen oder mehrere Werte angeben.

Bei dieser Syntax geht es darum, dass – unabhängig von der eingegebenen Anzahl der Werte – zwingend eine Array-Variable definiert werden kann. Die Definition mittels Array-Operator wird besonders dann verwendet, wenn Sie nicht wissen, wie viele Objekte Sie zur Speicherung in einer Variablen erwarten dürfen.

## Beispiele

```
$a = @() # $a.Count ergibt 0
```

```
$a = @("Hallo") # $a.Count ergibt 1
```

```
$a = @(Get-Process -Name svchost) # $a.Count ergibt ?
```

- ✓ Im letzten Beispiel ist in einem Programmablauf nicht unbedingt bekannt, wie viele Instanzen des Prozesses vorhanden sind.

## Werte eines Arrays lesen

Um alle vorhandenen Werte eines Arrays anzeigen zu lassen, geben Sie wie bei allen Variablen einfach den Variablenamen ein, z. B. `$a`.

Um einen speziellen Wert anzuzeigen, geben Sie einen numerischen Index in eckigen Klammern direkt hinter dem Variablenamen an. Die Zählung des Index beginnt bei *0*. Der erste Wert der Variablen `$a` lässt sich über `$a[0]` ansprechen, der zweite Wert über `$a[1]`, der dritte über `$a[2]` usw.

## Werte eines Arrays ändern

Wenn Sie Werte eines Arrays ändern wollen, müssen Sie die Array-Variablen mit dem entsprechenden Index ansprechen.

```
$a[7] = 25
```

- ✓ Der achte Wert der Array-Variablen \$a wird auf 25 gesetzt.

```
$a.SetValue(25, 7)
```

- ✓ Hier wird die identische Aktion ausgeführt. Dabei wird die Methode `SetValue()` auf die Variable \$a angewendet. Als Argumente in Klammern werden der Wert an erster Stelle und der Indexwert an zweiter Stelle angegeben. Getrennt werden die Werte durch ein Komma.

Wollen Sie das Array erweitern und einen weiteren Wert hinzufügen, verwenden Sie den Operator „`+=`“:

```
$a += 19
```

Um das Array schließlich komplett zu löschen, entfernen Sie die Variable wie bekannt.

## 7.3 Konstanten

### Fast wie eine Variable

Eine Konstante ist eine Sonderform einer Variablen. Die Besonderheit besteht darin, dass eine Konstante nicht gelöscht und ihr Wert nicht mehr verändert werden kann.

Sie kennen bereits das Cmdlet, mit dem eine Konstante erstellt werden kann. Es handelt sich um das Cmdlet `New-Variable`, bei dem Sie den Parameter `-Option` mit dem Wert `Constant` nach folgendem Muster angeben:

```
New-Variable -Name <Name> -Value <Wert> -Option Constant
```

Für die Definition der Konstanten `PI` zur Kreisumfangberechnung können Sie folgende Zeile verwenden:

```
New-Variable -Name PI -Value 3.14159265 -Option Constant
```

- ✓ \$PI wird definiert, ein Wert wird zugewiesen. Der Wert `Constant` des Parameters `-Option` sorgt dafür, dass \$PI als Konstante definiert wird.

## 7.4 Arithmetische Operatoren

Mit arithmetischen Operatoren können mathematische Berechnungen durchgeführt werden. Sie erwarten entweder Ganzzahl- bzw. Gleitkommazahl-Variablen oder feste Werte als Parameter und liefern ein numerisches Ergebnis zurück. Sie können folgende Operatoren verwenden:

Operator	Name	Bedeutung	Beispiel	Wert nach der Operation
+	Addition	\$a + \$b ergibt die Summe von \$a und \$b.	\$a = 10 \$b = 2 \$c = \$a + \$b	\$c = 12
-	Subtraktion	\$a - \$b ergibt die Differenz von \$a und \$b.	\$a = 10 \$b = 2 \$c = \$a - \$b	\$c = 8
*	Multiplikation	\$a * \$b ist das Produkt aus \$a und \$b.	\$a = 10 \$b = 2 \$c = \$a * \$b	\$c = 20

Operator	Name	Bedeutung	Beispiel	Wert nach der Operation
/	Division	$\$a / \$b$ ist der Quotient von $\$a$ und $\$b$ .	$\$a = 10$ $\$b = 2$ $\$c = \$a / \$b$	$\$c = 5$
%	Modulo	$\$a \% \$b$ ist der Rest der ganzzahligen Division von $\$a$ und $\$b$ .	$\$a = 10$ $\$b = 3$ $\$c = \$a \% \$b$	$\$c = 1$
++	Präinkrement	$++\$a$ erhöht die Variable $\$a$ um 1 <b>vor</b> der weiteren Verwendung.	$\$a = 10$ $\$b = 2$ $\$c = ++\$a + \$b$	$\$a = 11$ $\$c = 13$
--	Prädekrement	$--\$a$ verringert die Variable $\$a$ um 1 <b>vor</b> der weiteren Verwendung.	$\$a = 10$ $\$b = 2$ $\$c = --\$a + \$b$	$\$a = 9$ $\$c = 11$
++	Postinkrement	$\$a++$ erhöht die Variable $\$a$ um 1 <b>nach</b> der Verwendung.	$\$a = 10$ $\$b = 2$ $\$c = \$a++ + \$b$	$\$a = 11$ $\$c = 12$
--	Postdecrement	$\$a--$ verringert die Variable $\$a$ um 1 <b>nach</b> der Verwendung.	$\$a = 10$ $\$b = 2$ $\$c = \$a-- + \$b$	$\$a = 9$ $\$c = 12$
+=	Zuweisungsoperator	$\$a += \$b$ weist der Variablen $\$a$ den Wert $\$a + \$b$ zu (Kurzschreibweise für $\$a = \$a + \$b$ ).	$\$a = 10$ $\$a += 5$	$\$a = 15$
-=	Zuweisungsoperator	$\$a -= \$b$ ist die Kurzschreibweise für $\$a = \$a - \$b$ .	$\$a = 10$ $\$a -= 5$	$\$a = 5$
*=	Zuweisungsoperator	$\$a *= \$b$ ist die Kurzschreibweise für $\$a = \$a * \$b$ .	$\$a = 10$ $\$a *= 5$	$\$a = 50$
/=	Zuweisungsoperator	$\$a /= \$b$ ist die Kurzschreibweise für $\$a = \$a / \$b$ .	$\$a = 10$ $\$a /= 5$	$\$a = 2$

Werden mehrere Berechnungen in einer Zuweisung durchgeführt (z. B.  $\$a = \$b - \$c * \$d$ ), werden die üblichen mathematischen Rechenregeln angewandt:

- ✓ Bearbeitung der Berechnung von links nach rechts
- ✓ Punkt- vor Strichrechnung
- ✓ Geklammerte Ausdrücke (gemeint sind runde Klammern) werden zuerst ausgewertet.

Für eine Verknüpfung von Zeichenketten verwenden Sie ebenfalls den Operator **[+]**:



```
$a = "Dies wird"
$b = "komplett"
$a + " ein " + $b + "er Satz. "
```

- ✓ Sie definieren die Variablen  $\$a$  und  $\$b$  und weisen Werte zu, die aus Zeichenketten bestehen.
- ✓ In der letzten Zeile verwenden Sie die Variablen und verknüpfen sie und eingegebene Zeichenfolgen mithilfe des Zeichens **[+]**. Sie erhalten die Ausgabe **Dies wird ein kompletter Satz.**

## 7.5 Vergleichsoperatoren

Sie müssen für viele fortgeschrittene Tätigkeiten innerhalb der PowerShell in der Lage sein, Werte miteinander zu vergleichen. Bedingte Anweisungen benötigen beispielsweise diese Möglichkeit, damit Sie weitere Befehle abhängig vom Ergebnis des Vergleichs ausführen können. Neben der Kenntnis von Variablen ist dies ein weiterer wichtiger Baustein auf dem Weg zur Erstellung eigener Skripte.

Die PowerShell stellt wie andere Skriptsprachen eine Reihe von Vergleichsoperatoren zur Verfügung, allerdings unterscheiden sich diese deutlich von den meisten anderen Sprachen. Die Operatoren bestehen meist aus einer Abkürzung des englischen Begriffs und werden wie Parameter von einem Minuszeichen (-) eingeleitet.

Vergleichsoperatoren liefern einen booleschen Wert zurück: TRUE, wenn der Vergleich zutrifft, und FALSE, wenn der Vergleich nicht zutrifft. Sie stellen durch ihre Antwort sozusagen die Weichen für Ihr weiteres Vorgehen.

Folgende Vergleichsoperatoren stellt die PowerShell zur Verfügung (ohne Operatoren zum binären Vergleich):

Operator	Bedeutung	Beispiele
-eq	Ist gleich (equal to)	2 -eq 3 (Ergebnis: FALSE) "Harry" -eq "harry" (Ergebnis: TRUE)
-ne	Ist ungleich (not equal to)	2 -ne 3 (Ergebnis: TRUE) "Harry" -ne "harry" (Ergebnis: FALSE)
-gt	Größer als (greater than)	2 -gt 3 (Ergebnis: FALSE) "Harry" -gt "harry" (Ergebnis: FALSE)
-ge	Größer oder gleich (greater than or equal to)	2 -ge 3 (Ergebnis: FALSE) "Harry" -ge "harry" (Ergebnis: TRUE)
-lt	Kleiner als (less than)	2 -lt 3 (Ergebnis: TRUE) "Harry" -lt "harry" (Ergebnis: FALSE)
-le	Kleiner oder gleich (less than or equal to)	2 -le 3 (Ergebnis: TRUE) "Harry" -le "harry" (Ergebnis: TRUE)
-Contains	Prüft eine gegebene Werteliste, ob ein Wert enthalten ist	"abc", "def" -Contains "abc" (Ergebnis: TRUE) "abc", "def" -Contains "bc" (Ergebnis: FALSE)
-NotContains	Umkehrung des Operators -Contains	"abc", "def" -NotContains "abc" (Ergebnis: FALSE) "abc", "def" -NotContains "bc" (Ergebnis: TRUE)
-In	Wie -Contains, nur in umgekehrter Reihenfolge	"abc" -In "abc", "def" (Ergebnis: TRUE) "bc" -In "abc", "def" (Ergebnis: FALSE)
-NotIn	Umkehrung des Operators -In	"abc" -NotIn "abc", "def" (Ergebnis: FALSE) "bc" -NotIn "abc", "def" (Ergebnis: TRUE)
-Is	Prüfung auf Typgleichheit	[string]\$string = "Herdt-Verlag" \$string -Is [string] (Ergebnis: TRUE)
- IsNot	Umkehrung des Operators -Is	[string]\$string = "Herdt-Verlag" \$string -IsNot [string] (Ergebnis: FALSE)

Operator	Bedeutung	Beispiele
-Like	Einfacher Text-abgleich; prüft, ob sich eine Zeichenkette in einer anderen befindet	"abcdef" -Like "abc" (Ergebnis: FALSE) "abcdef" -Like "abc*" (Ergebnis: TRUE) "abcdef" -Like "*c*" (Ergebnis: TRUE)
-NotLike	Umkehrung des Operators -Like	"abcdef" -NotLike "abc" (Ergebnis: TRUE) "abcdef" -NotLike "abc*" (Ergebnis: FALSE) "abcdef" -NotLike "*c*" (Ergebnis: FALSE)
-Match	Suche nach Substring ohne Wildcards (auch Mustervergleich mit regulären Ausdrücken)	"abcdef" -Match "c" (Ergebnis: TRUE)
-NotMatch	Umkehrung des Operators -Match	"abcdef" -NotMatch "c" (Ergebnis: FALSE)

Alle in der vorherigen Tabelle vorgestellten Operatoren liegen in drei Varianten vor:

- ✓ So wie in der Tabelle vorgestellt:  
Die Vergleichsoperatoren unterscheiden standardmäßig nicht Groß- und Kleinschreibung.  
Beispiel: -eq
- ✓ Mit vorangestelltem Buchstaben **C** (*case-sensitive*):  
Diese Vergleichsoperatoren unterscheiden ausdrücklich Groß- und Kleinschreibung.  
Beispiel: -ceq
- ✓ Mit vorangestelltem Buchstaben **I** (*case-insensitive*):  
Diese Vergleichsoperatoren unterscheiden ausdrücklich nicht Groß- und Kleinschreibung.  
Beispiel: -ieq

Wollen Sie das boolesche Ergebnis eines Vergleichs umkehren, z. B. weil es besser in Ihre Programmierlogik passt? Auch da hilft die PowerShell und stellt den Operator **-not** zur Verfügung. Analog zu anderen Programmiersprachen dürfen Sie den Operator verkürzt als Ausrufezeichen **!** schreiben. So ergibt der Vergleich **3 -gt 2** das Ergebnis **TRUE**, der Vergleich **-not (3 -gt 2)** aber das Ergebnis **FALSE**.

### Komplexere Vergleiche mit mehreren Einzelvergleichen

Bei komplexeren Fragestellungen können Sie mehrere Vergleiche miteinander verknüpfen. So können Sie beispielsweise abfragen, ob ein Prozess vorhanden ist *und* die befürchtete Menge an Ressourcen benötigt.

Auch zu diesem Thema stellt die PowerShell Operatoren zur Verfügung, die Sie nach Wunsch verknüpfen können:

Operator	Bedeutung	Beispiele
-and	Alle Einzelbedingungen müssen erfüllt sein.	( (2 -gt 3) -and (2 -gt 1) -and (0 -gt -1) ) (Ergebnis: FALSE)
-or	Mindestens eine der Einzelbedingungen muss erfüllt sein.	( (2 -gt 3) -or (2 -gt 1) -or (0 -gt -1) ) (Ergebnis: TRUE)
-xor	Genau eine der Einzelbedingungen muss erfüllt sein, nicht aber mehrere oder alle.	( (2 -gt 3) -xor (2 -gt 1) -xor (0 -gt 1) ) (Ergebnis: TRUE)

Wie Sie in den Beispielen sehen, werden die Einzelbedingungen in Klammern gesetzt, damit sie zuerst verarbeitet werden. Es werden nur die Ergebnisse der einzelnen Bedingungen miteinander verknüpft und das Gesamtergebnis berechnet.

## 7.6 Kontrollstrukturen in der PowerShell

Herzstück einer Programmiersprache ist die Möglichkeit, eine Programmierung individuell nach eigenen Vorstellungen zu gestalten. Kontrollstrukturen steuern dabei den Ablauf einer Programmierung durch Verzweigungen und Schleifen.

Verzweigungen führen – abhängig vom Ergebnis der Prüfung von Bedingungen – Anweisungsblöcke aus. Schleifen steuern die wiederholte Ausführung von Anweisungsblöcken, bis eine Bedingung erfüllt ist.

Die PowerShell stellt sowohl Verzweigungen als auch Schleifen zur Verfügung. In den folgenden Abschnitten lernen Sie die Syntax als letzten Schritt vor der Programmierung eigener Skripte kennen.

## 7.7 Die einfache If-Anweisung

In der PowerShell erreichen Sie auf verschiedene Arten eine Verzweigung des Programmablaufs. Die einfachste und häufig eingesetzte Variante ist die Bedingungsprüfung mit der **If**-Anweisung. **Wenn** die angegebene Bedingung erfüllt ist, **dann** wird der in geschweiften Klammern angegebene Anweisungsblock ausgeführt. Ist die Bedingung nicht erfüllt, wird der Anweisungsblock übersprungen.

### Syntax und Bedeutung der If-Anweisung

- ✓ Die If-Anweisung beginnt mit dem Schlüsselwort **if**.
- ✓ Die Bedingung steht in runden Klammern.
- ✓ Der Anweisungsblock steht in geschweiften Klammern.
- ✓ Als Bedingung ist ein logischer Ausdruck anzugeben, der einen der beiden Zustände TRUE (wahr) oder FALSE (falsch) zurück liefert.
- ✓ Liefert die Bedingung TRUE zurück, werden die Anweisungen ausgeführt. Ist die Bedingung FALSE, werden die Anweisungen ignoriert.
- ✓ Nach dem Ende der If-Anweisung werden alle weiteren Anweisungen unabhängig von der Bedingung abgearbeitet.

```
If (Bedingung)
{
    Anweisungsblock
}
```

Die auszuführenden Anweisungen werden Anweisungsblock genannt, auch wenn sie nur aus einer Anweisung bestehen.

### Beispiel

```
If ($x -eq 27) { Write-Host "Der Wert der Variablen x ist 27." }
```

- ✓ Wenn Sie eine Variable \$x in der aktuellen PowerShell-Sitzung definiert haben, die den Wert 27 hat, wird der Anweisungsblock in geschweiften Klammern ausgeführt.
- ✓ In allen anderen Fällen geschieht nichts, da keine Handlungsanweisung für diesen Fall gegeben wurde.



Wenn Sie ein ähnliches Beispiel in einem Skript verwenden, achten Sie aus Gründen der Übersichtlichkeit und Nachvollziehbarkeit auf Zeilenumbrüche wie bei der Vorstellung der Syntax. Bei direkter Eingabe in ein Konsolenfenster zum Testen belassen Sie es bei der einzeiligen Schreibweise (wie auch in den folgenden Beispielen, auch wenn sie komplexer sind).



### Mehr als eine Zeile? PowerShell ISE

Es folgen nun komplexere Beispiele, die sich nicht auf die Eingabe in einer Zeile beschränken. Verwenden Sie dafür den Skriptbereich der PowerShell ISE.

```
If ($x -eq 27)
{
    Write-Host "Der Wert der Variablen x ist 27."
}
Else
{
    Write-Host "Der Wert von x ist - sofern vorhanden - nicht 27."
}
```

Für die Eingabe längerer Passagen bietet sich die Verwendung der PowerShell ISE an.

Sie müssen den Code nicht einmal als Datei speichern, sondern einfach die auszuführende Passage markieren und dann die Taste **F8** drücken. Damit wird die Auswahl ausgeführt, und Sie können die folgenden Beispiele testen.

## 7.8 Die **If**-Anweisung mit **Else**-Zweig

Soll nicht nur ein Anweisungsblock durchgeführt werden, falls die Bedingung erfüllt ist, sondern ein anderer, sofern die Bedingung nicht erfüllt ist, verwenden Sie die **If-Else**-Anweisung.

### Syntax und Bedeutung der **If-Else**-Anweisung

- ✓ Den alternativen Befehlszweig leiten Sie mit dem Schlüsselwort **Else** ein.
- ✓ Danach folgen in geschweiften Klammern die alternativen Anweisungen.
- ✓ Falls die Bedingung erfüllt ist, wird Anweisungsblock 1 ausgeführt, sonst Anweisungsblock 2 (nach dem Prinzip *wenn-dann-sonst*).

```
If (Bedingung)
{
    Anweisungsblock 1
}
Else
{
    Anweisungsblock 2
}
```

### Beispiel

```
$x = 27          # Wert zum Testen anpassen

If ($x -eq 27)
{
    Write-Host "Der Wert der Variablen x ist 27."
}
Else
{
    Write-Host "Der Wert von x - sofern vorhanden - ist nicht 27."
}
```

- ✓ Der erste Teil des Beispiels wurde nicht verändert.
- ✓ Der **Else**-Zweig mit einem Anweisungsblock wurde hinzugefügt, falls der Wert der Variablen \$x nicht 27 ist. Damit erhalten Sie in jedem Fall eine Ausgabe.

## 7.9 Erweiterte If-Anweisung mit ElseIf

Zusätzlich zu verschachtelten If-Anweisungen können Sie mit ElseIf eine weitere Bedingung prüfen. Da Sie ElseIf beliebig oft einsetzen können, ist es möglich, eine beliebige Anzahl von zusätzlichen Bedingungen zu prüfen. Erst dann wird der Else-Zweig aufgerufen, falls keine der vorherigen Bedingungen zutrifft.

```
If (Bedingung 1)
{
    Anweisungsblock 1
}
ElseIf (Bedingung 2)
{
    Anweisungsblock 2
}
[ElseIf (Bedingung n)
{
    Anweisungsblock n
}]
Else
{
    Anweisungsblock Else
}
```

Allgemeines Beispiel für eine If-Anweisung mit ElseIf

- ✓ Für die einzelnen If-Anweisungen und ebenso für die einzelnen ElseIf-Anweisungen gelten die gleichen Regeln wie bei einer einfachen If-Anweisung.
- ✓ Sie können in einer If-Anweisung beliebig viele ElseIf-Anweisungen verwenden.
- ✓ Der Else-Zweig wird – wie bei der einfachen If-Anweisung – nur dann ausgeführt, wenn keine der vorherigen Bedingungen zutrifft.

### Beispiel

```
$x = 19          # Wert zum Testen anpassen

If ($x -gt 27)
{
    Write-Host "Der Wert der Variablen x ist größer als 27."
}
ElseIf ($x -eq 27)
{
    Write-Host "Der Wert der Variablen x ist genau 27."
}
Else
{
    Write-Host "Der Wert von x - sofern vorhanden - ist kleiner als 27."
}
```

- ✓ An erster Stelle erfolgt die Prüfung, ob die Variable \$x größer als 27 ist. Ist dies der Fall, erfolgt eine entsprechende Ausgabe, und die restlichen Zeilen des Beispiels werden übersprungen.
- ✓ Ist die erste Bedingung nicht erfüllt, wird die nächste Bedingung (ElseIf) geprüft. Ist der Wert der Variablen genau 27, erfolgt die dazugehörige Ausgabe.
- ✓ Nur wenn beide Bedingungen nicht zutreffen, wird der Else-Zweig ausgeführt und die passende Meldung ausgegeben.

## 7.10 Fallauswahl mit der **Switch**-Anweisung

Wenn Sie eine Variable mit verschiedenen Werten vergleichen und in Abhängigkeit vom Auswertungsergebnis unterschiedlich reagieren möchten, kann die Programmierung mit **If**-Anweisungen sehr aufwendig und durch die Klammerung syntaktisch kompliziert sein. Als Alternative zur verschachtelten Auswahl mit **If** können Sie in diesem Fall die **Switch**-Anweisung einsetzen.

Bei der **Switch**-Anweisung bestimmt der Wert einer Variablen, welcher Anweisungsblock ausgeführt wird. Diese Vorgehensweise wird Fallauswahl oder Selektion genannt.

### Syntax und Bedeutung der **Switch**-Anweisung

```
switch ($variable)
{
    Wert-1 {
        Anweisungsblock 1
        [break]
    }
    Wert-2 {
        Anweisungsblock 2
        [break]
    }
    Default {
        Anweisungsblock 3
    }
}
```

- ✓ Die Fallauswahl wird mit dem Schlüsselwort **Switch** eingeleitet.
- ✓ Danach folgt die Variable, abhängig von deren Wert eine Auswahl getroffen werden soll. Anschließend leitet die öffnende geschweifte Klammer die Fallauswahl ein.
- ✓ Jede Auswahl wird durch den Wert der Variablen eingeleitet. Stimmt der Wert der Variablen mit einem der aufgeführten Auswahlwerte überein bzw. ergibt die Bedingungsprüfung TRUE, dann wird der – in eigenen geschweiften Klammern stehende – Anweisungsblock danach bis zur optionalen Anweisung **break** ausgeführt. Die restlichen Auswahlblöcke werden nicht ausgeführt.
- ✓ Die einzelnen Zweige der **Switch**-Anweisung werden der Reihe nach geprüft. Ergibt die Prüfung eine Übereinstimmung, arbeitet die PowerShell den direkt folgenden Anweisungsblock dieses Zweigs bis zur Anweisung **break** ab. Ist kein **break** am Ende eines Zweiges angegeben, so werden auch alle nachfolgenden Blöcke ohne weitere Prüfung ausgeführt, bis eine **break**-Anweisung erfolgt. 
- ✓ Stimmt der Wert der Variablen mit keinem der angegebenen Werte überein, wird der Anweisungsblock nach der optionalen **Default**-Anweisung durchgeführt. Wird der **Default**-Auswahlblock weggelassen, führt das Programm in diesem Fall keine Anweisungen aus, sondern erst eventuell folgende Befehle nach der **Switch**-Anweisung.
- ✓ Die Fallauswahl wird nach der schließenden geschweiften Klammer beendet.

### Beispiel

```
$a = 3          # Wert zum Testen anpassen

Switch ($a)
{
    1 { Write-Host "A = 1" }
    2 { Write-Host "A = 2" }
    3 { Write-Host "A = 3" }
    Default { Write-Host "Nicht 1, nicht 2, nicht 3." }
}
```

- ✓ Eine Variable \$a wird initialisiert. Auf Basis des Wertes der Variablen soll der passende Anweisungsblock ausgeführt werden. Dies wird durch den Einsatz der Fallauswahl mit Switch auf syntaktisch einfache Weise erreicht.
- ✓ Innerhalb der geschweiften Klammern erfolgt die Fallauswahl nach dem Muster *Wert-Anweisungsblock*.
- ✓ Trifft ein Wert nicht zu, wird der dazugehörige Anweisungsblock übersprungen.
- ✓ Trifft keiner der zur Verfügung stehenden Werte zu, wird der optionale *Default-Anweisungsblock* ausgeführt.

## 7.11 Schleifen

### Schleifen verwenden

Um einen bestimmten Teil des Programms mehrfach auszuführen bzw. zu wiederholen, benötigen Sie Schleifen. Durch die Verwendung von Schleifen sparen Sie Programmcode und können variabel festlegen, wie oft oder bis zum Eintreffen welcher Bedingung die Schleife durchlaufen werden soll.

Folgende Schleifen gibt es in der PowerShell:

- ✓ While- bzw. Do-While-Schleife:  
Wenn Ihnen als Programmierer nicht bekannt ist, wie oft eine Anweisung wiederholt werden soll, bzw. Sie eine Schleife so lange ausführen möchten, bis eine bestimmte Bedingung eingetroffen ist, verwenden Sie die While- bzw. Do-While-Schleife.
- ✓ For-Schleife:  
Wenn Sie die genaue Anzahl kennen oder diese vorher in der PowerShell ermitteln können, wie oft eine Anweisung wiederholt werden soll, verwenden Sie die For-Schleife.
- ✓ForEach-Schleife:  
Wenn Sie eine Anzahl von Werten, z. B. in einer Array-Variablen, der Reihe nach lesen und weiterbearbeiten möchten, ist dieser Schleifentyp die richtige Wahl.

## 7.12 Mit der `While`-Schleife arbeiten

In einer While-Schleife prüft die PowerShell zu Beginn die vorgegebene Bedingung und führt den angegebenen Anweisungsblock so lange aus, bis die Bedingung nicht mehr erfüllt ist. Diese Prüfung wird automatisch **vor** jedem weiteren Durchlauf wiederholt. Ergibt der Test der Bedingung den Wert TRUE, wird die Schleife durchlaufen. Liefert die Bedingung den Wert FALSE zurück, werden die Anweisungen der Schleife übersprungen, und die Abarbeitung des Programms wird nach der Schleife fortgesetzt.

Eine Prüfung **vor** der Ausführung des Anweisungsblocks wird als **kopfgesteuert** bezeichnet. Ergibt bereits die erste Bedingungsprüfung den Wert FALSE, wird der Anweisungsblock gar nicht ausgeführt. Eine While-Schleife kann null Mal bis unendlich oft (sogenannte Endlosschleife) ausgeführt werden.

### Syntax und Bedeutung der `While`-Schleife

- ✓ Das Schlüsselwort `While` leitet die Schleife ein. In runden Klammern wird die Bedingung angegeben, die vor der Abarbeitung überprüft wird.
- ✓ Ist der Rückgabewert der Bedingung TRUE, wird der Anweisungsblock im Schleifenkörper ausgeführt.
- ✓ Nach der letzten Anweisung des Anweisungsblocks bzw. mit Erreichen der schließenden geschweiften Klammer springt die PowerShell zum Anfang der Schleife zurück und überprüft erneut die Bedingung. Ist das Ergebnis der Bedingungsprüfung TRUE, wird der Anweisungsblock erneut ausgeführt.
- ✓ Diese Schritte werden so lange wiederholt, bis die Auswertung der Bedingungsprüfung am Beginn der Schleife den Wert FALSE liefert. Die Schleife wird dann verlassen und weitere Befehle nach der Schleife abgearbeitet.

```
While (Bedingung)
{
    Anweisungsblock
}
```

### Beispiel

```
$a = 0          # Wert zum Testen anpassen

While ($a -le 3)
{
    Write-Host $a
    $a = $a + 1 # alternativ: $a++ oder $a += 1
}
```

- ✓ Die Variable `$a` wird initialisiert. Sie wird in der Schleife als Teil der Bedingung benötigt. Die Schleife soll so lange durchlaufen werden, bis der Wert der Variablen größer als 3 ist.
- ✓ Im Anweisungsblock stehen die Anweisungen, die ausgeführt werden, solange der Wert von `$a` kleiner oder gleich 3 ist.
- ✓ Wichtig ist es, im Anweisungsblock den Wert der Variablen, die vor jedem Schleifendurchlauf geprüft wird, zu verändern. Hier wird der Wert um 1 erhöht. Wird der Wert nicht verändert, wird die Schleife endlos durchlaufen, weil das Ergebnis der Bedingungsprüfung `$a -le 3` im Beispiel stets TRUE wäre.

### Vermeiden von Endlosschleifen

Schleifen bergen grundsätzlich die Gefahr, dass sie durch eine falsche Logik in der Programmierung in eine Endlosschleife laufen. Dies wird Ihnen nicht als Fehler gemeldet. Der Aufruf einer Endlosschleife ist möglich, es erfolgt eine Endlosausgabe, die Sie mit der Tastenkombination `Strg C` abbrechen können.



## 7.13 Mit der Do-While-Schleife arbeiten

Im Unterschied zur While-Schleife, die die Bedingung für die Wiederholung am Anfang hat, wird in der Do-While-Schleife die Bedingung erst **nach** der letzten Anweisung in der Schleife überprüft. Man spricht deshalb von einer **fußgesteuerten** Schleife. Daraus folgt auch, dass der Anweisungsblock immer mindestens einmal ausgeführt wird. Auch hier gilt wie bei der While-Schleife: Liefert die Bedingung einen Wert TRUE zurück, wird die Schleife erneut durchlaufen, ansonsten wird sie verlassen.

### Syntax und Bedeutung der Do-While-Schleife

- ✓ Das Schlüsselwort Do leitet die Schleife ein.
- ✓ Die Anweisungen im Schleifenkörper werden auf jeden Fall mindestens einmal ausgeführt.
- ✓ Am Ende der Schleife wird die Bedingung der Schleife über das Schlüsselwort While ausgewertet. Die Bedingung wird in runden Klammern angegeben.
- ✓ Trifft die Bedingung zu, wird die Schleife erneut ausgeführt.

```
Do
  {
    Anweisungsblock
  }
While (Bedingung)
```

### Beispiel

```
$a = 0          # Wert zum Testen anpassen

Do
{
    Write-Host $a
    $a = $a + 1 # alternativ: $a++ oder $a += 1
}
While ($a -le 3)
```

- ✓ Das Beispiel kennen Sie bereits von der While-Schleife.
- ✓ Der einzige Unterschied besteht darin, dass die Schleife auf jeden Fall einmal ausgeführt wird. Das trifft auch zu, wenn die Variable \$a z. B. einen Startwert 4 hat, der bei der Bedingungsprüfung zum Ergebnis FALSE führt.

### Besonderheit Do-Until-Schleife

Die PowerShell bietet zusätzlich eine Abwandlung der Do-While-Schleife. Es handelt sich um die Do-Until-Schleife, die syntaktisch identisch ist. Der Unterschied liegt im Schlüsselwort Until, das die Bedingungsprüfung am Fuß der Schleife umkehrt.

Die Schleife wird mit Until so lange ausgeführt, bis die Bedingungsprüfung TRUE ergibt. Bei der While-Schleife wird die Schleife so lange ausgeführt, bis die Bedingungsprüfung FALSE ergibt.

Im obigen Beispiel zur While-Schleife muss die Zeile `While ($a -le 3)` bei einer Until-Schleife in `Until ($a -gt 3)` umgewandelt werden, um zum identischen Verhalten der Schleifen zu gelangen.

## 7.14 Mit der For-Schleife arbeiten

Im Unterschied zur While-Schleife, die so oft durchlaufen wird, bis der Test der Bedingung FALSE ergibt, wird in der For-Schleife genau angegeben, wie oft die Schleife durchlaufen werden soll. Bei der For-Schleife kann berechnet werden:

- ✓ die Anzahl der Wiederholungen
- ✓ Beginn und Ende der Wiederholung

Übergeben werden der Start- und Endwert sowie die Bedingung, wie der Endwert erreicht werden soll.

### Syntax und Bedeutung der For-Schleife

- ✓ Die For-Schleife beginnt mit dem Schlüsselwort For.
- ✓ Mit der Anweisung *Initialisierung* wird der Anfangswert für die Schleife festgelegt.
- ✓ Die Anweisung *Bedingung* legt fest, unter welchen Konditionen die Schleife durchlaufen werden soll. Diese Bedingung, die vor jedem Durchlauf abgefragt wird, liefert als Wert TRUE oder FALSE zurück. Ist der Wert TRUE, wird die Schleife durchlaufen. Andernfalls wird das Programm mit den Anweisungen nach der For-Schleife fortgesetzt. Ist die Bedingung bereits vor dem ersten Schleifendurchlauf FALSE, wird die Schleife gar nicht durchlaufen.
- ✓ Die Anweisung *Reinitialisierung* legt fest, wie die Variable verändert werden soll, die in der Bedingung ausgewertet wird. Diese Veränderung geschieht am Ende der Schleife vor dem nächsten Schleifendurchlauf.
- ✓ Die drei Anweisungen werden jeweils durch ein Semikolon voneinander getrennt.

```
For (Initialisierung; Bedingung; Reinitialisierung)
{
    Anweisungsblock
}
```

### Beispiel

```
For ($a = 1; $a -le 5; $a++)
{
    Test-Connection -ComputerName 127.0.0.$a -Count 1
}
```

- ✓ Eine Variable \$a wird mit dem Wert 1 initialisiert. Die Bedingung, die über die Ausführung der Schleife bestimmt, lautet, dass der Wert der Variablen \$a kleiner oder gleich 5 sein muss. Schließlich erfolgt die Anweisung, dass der Wert von \$a nach jedem Schleifendurchlauf um 1 erhöht werden soll. Die Schleife wird mit diesen Angaben insgesamt fünfmal durchlaufen.
- ✓ Im Anweisungsblock wird eine Verbindungsanfrage („Ping“) an einen Rechner mit der IP-Adresse 127.0.0.x gesendet. X steht in jedem Schleifendurchlauf für den aktuellen Wert der Variablen \$a. Es erfolgt also ein Ping an die Adressen 127.0.0.1, 127.0.0.2, 127.0.0.3, 127.0.0.4 und 127.0.0.5.

## 7.15 Einsatz der ForEach-Schleife

In einer ForEach-Schleife müssen Sie keine Bedingungen oder die Anzahl der Schleifendurchläufe definieren. Wenn Sie eine Anzahl von Werten, z. B. in einer Array-Variablen, der Reihe nach lesen und weiterbearbeiten möchten, ist dieser Schleifentyp die richtige Wahl.

## Syntax und Bedeutung der **ForEach**-Schleife

- ✓ Das Schlüsselwort **ForEach** leitet die Schleife ein.
- ✓ In runden Klammern geben Sie die Datenquelle (**<Quelle>**) und den Variablennamen (**\$<name>**) an, unter dem das aktuelle Element im Anweisungsblock angesprochen werden kann.
- ✓ Im Anweisungsblock beziehen Sie sich in der Regel auf **\$<name>** und führen die beabsichtigten Operationen durch.

```
ForEach ($<name> in <Quelle>)
{
    Anweisungsblock
}
```

## Beispiel

```
$array = 2, 4, 6, 8, 10
ForEach ($wert in $array) { $wert * 5 }
```

- ✓ Es wird eine Array-Variable **\$array** mit den Werten 2, 4, 6, 8 und 10 definiert.
- ✓ In der Definition der **ForEach**-Schleife wird festgelegt, dass jedes aktuelle Element aus der Variablen **\$array** unter dem Namen **\$wert** angesprochen werden kann.
- ✓ Im Anweisungsblock erfolgt die Ansprache des jeweils aktuellen Wertes über die Variable **\$wert**. Die angegebene Operation wird ausgeführt und jeweils das Ergebnis ausgegeben.

Es gibt noch eine andere Variante der **ForEach**-Schleife, die ihre Datenquelle in Objekten findet, die über die Pipeline geliefert werden. In dem Fall können Sie die Schleife mit einer etwas anderen Syntax verwenden.

- ✓ Die **ForEach**-Schleife steht nicht an erster Stelle in der Pipeline, sondern erhält die zu verarbeitenden Daten über den vorhergehenden Befehl.
- ✓ Sie können bis zu drei Anweisungsblöcke definieren. Der erste und der letzte Block sind Anweisungen, die vor und nach Abarbeitung der Schleife ausgeführt werden.
- ✓ Der mittlere Anweisungsblock übernimmt die Verarbeitung der Objekte in einer Schleife.

```
<command> | ForEach
{
    Anweisungsblock Beginn
}
{
    Anweisungsblock Schleife
}
{
    Anweisungsblock Ende
}
```

## Beispiel

```
Get-Process | ForEach {$count = 0} {$_.Name; $count++} {"Gesamt: $count Prozesse."}
```

- ✓ Als Datenquelle kommen auch Objekte in Frage, die über die Pipeline geliefert werden. Hier sind es die Objekte, die das Cmdlet **Get-Process** liefert.
- ✓ Im ersten Anweisungsblock (vor dem Durchlaufen der Schleife) wird eine Variable **\$count** definiert, die den Ausgangswert 0 erhält.
- ✓ Im zweiten Anweisungsblock wird jeweils der Name des aktuellen Prozesses ausgegeben und der Wert der Variablen **\$count** um eins erhöht.
- ✓ Im letzten Anweisungsblock wird eine Zeichenfolge mit dem Wert der Variablen **\$count** ausgegeben.



ForEach und **ForEach-Object** scheinen auf den ersten Blick identisch zu sein. Da **ForEach** aber nicht unbedingt die nötigen Daten über die Pipeline erhält, ist die Verarbeitung über **ForEach** in der Regel schneller. Voraussetzung dafür ist, dass Daten bereits in einer Variablen vorliegen oder überaus schnell beschafft werden können. Ist dies nicht der Fall, setzen Sie **ForEach-Object** ein.

## 7.16 Anweisungen zur Ablaufsteuerung: **break** und **continue**

Die Anweisung **break** wird in den Anweisungsblock geschrieben und unterbricht an dieser Stelle die Ausführung der aktuellen Schleife. Die PowerShell fährt dann mit der Ausführung am Ende der aktuellen Schleifenanweisung fort, so als wäre die Schleifenanweisung vollständig durchlaufen worden.

Die **continue**-Anweisung überspringt die Ausführung des restlichen aktuellen Anweisungsblocks. Die PowerShell fährt dann mit dem nächsten Durchlauf der aktuellen Schleifenanweisung fort, so als wäre der Anweisungsblock vollständig durchlaufen worden.

### Schleifenabbruch mit **break**

Nicht immer ist es notwendig, eine Schleife bis zum definierten Ende zu durchlaufen. Auch innerhalb des Schleifendurchlaufs können andere Kriterien geprüft werden, die das weitere Durchlaufen der Schleife überflüssig machen. Wenn Sie z. B. 100 Schleifendurchläufe planen, das gewünschte Ergebnis bereits nach 10 Durchläufen finden, machen die restlichen 90 Schleifendurchläufe keinen Sinn mehr. Sie können in diesem Fall die Operation mit dem Schlüsselwort **break** abbrechen.

### Vorzeitiger Sprung zum nächsten Schleifendurchlauf mit **continue**

Wenn Sie erreichen möchten, dass für einen bestimmten Wert die weiteren Anweisungen innerhalb der Schleife nicht mehr ausgeführt werden, dann verwenden Sie das Schlüsselwort **continue**. Die Schleife selbst ist davon nicht betroffen und wird so oft durchlaufen, bis die von Ihnen definierte Bedingung nicht mehr erfüllt ist.

Mit **continue** springt das Programm zurück zum Anfang der Schleife und beginnt ihren nächsten Durchlauf. Bei **For**-Schleifen wird zudem die Reinitialisierungs-Anweisung ausgeführt.

## 7.17 Kurz zusammengefasst

### Nächster Schritt Skripting

Sie haben nun das nötige Grundgerüst erhalten, um im nächsten Kapitel die ersten Skripte zu schreiben. Microsoft gibt an, dass ein großer Prozentsatz der Beschäftigung mit der PowerShell interaktiv an der Konsole geschieht. Das ist sicherlich die Grundvoraussetzung für die nächsten Schritte. Programmiergrundlagen, die mehr oder minder allen Programmiersprachen gemeinsam sind, spielen auch bei der PowerShell eine gewichtige Rolle. Die PowerShell ist (auch) eine vollständige Programmiersprache, die mehr kann, als einfach eine Folge vormals interaktiver Befehle zu verarbeiten.

Sie haben Variablen kennengelernt und die für die Programmierung unumgängliche Arbeit mit Bedingungen und Schleifen zur individuellen Steuerung von Programmabläufen.

## 7.18 Übungen

### Mit Variablen, Konstanten und Arrays arbeiten

**Übungsdatei:** –

**Ergebnisdatei:** 7\_ergebnisse.txt

1. a) Erstellen Sie eine Variable \$geld und weisen Sie ihr den Wert 68 zu.  
b) Teilen Sie den Wert mit einem weiteren Befehl durch 4. Nutzen Sie dafür einen möglichst kurzen Befehl.  
c) Erstellen Sie eine Konstante \$c\_euro mit dem Wert *Euro in meiner Geldbörse*.  
d) Lassen Sie den Satz "Ich habe x Euro in meiner Geldbörse." ausgeben. Anstelle des "x" soll der Wert der Variablen \$geld stehen.
2. a) Legen Sie eine Array-Variable \$prozesse an, die als Werte die Namen der aktuell auf Ihrem System laufenden Prozesse enthält.  
b) Lassen Sie sich den 6. Eintrag des Arrays anzeigen.
3. a) Vergleichen Sie, ob "Hans" mit "hans" identisch ist (einmal mit und einmal ohne Unterscheidung von Groß- und Kleinschreibung).  
b) Prüfen Sie, ob 74 / 3 größer oder gleich 24.6 ist.  
c) Finden Sie heraus, ob im Wert der Variablen \$HOME der Buchstabe o enthalten ist.

### Mit Bedingungen und Schleifen arbeiten

**Übungsdatei:** –

**Ergebnisdatei:** 7\_ergebnisse.txt

1. a) Speichern Sie einen Wert zwischen 1 und 10 in einer Variablen. Fragen Sie ab, ob der Wert größer als 5 ist, und geben Sie den Text "Wert ist größer als 5" aus, wenn die Bedingung zutrifft.  
b) Erweitern Sie das Beispiel: Prüfen Sie, ob der Variablenwert gleich 5 ist, und geben Sie in dem Fall den Text "Wert ist gleich 5" aus.  
c) Erweitern Sie nochmals das Beispiel: Geben Sie jetzt noch für alle anderen Fälle den Text "Wert ist kleiner als 5" aus. (Prüfen Sie mit unterschiedlichen Werten.)
2. Legen Sie eine Variable \$note an und weisen Sie ihr einen Wert zwischen 1 und 6 zu. Verwenden Sie die Switch-Anweisung, um für die Werte die entsprechenden Texte gemäß der folgenden Tabelle auszugeben:

Note	Text
1	sehr gut
2	gut
3	befriedigend
4	ausreichend
5	mangelhaft
6	ungenügend

Experimentieren Sie mit verschiedenen Werten.

3. Programmieren Sie eine Schleife, die die Werte 10 bis 100 in Zehnerschritten ausgibt.
4. Geben Sie mithilfe einer For-Schleife die ganzen Vielfachen der Zahl 5 bis zum Wert 100 aus.

# 8 Skripting, Funktionen, Filter



**Beispieldatei:** 8\_kompletter code.txt (Ordner Kapitel 08)

## 8.1 PowerShell ISE verwenden

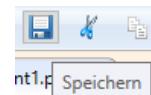
Falls Sie sich bisher für die Verwendung der PowerShell-Konsole entschieden haben, steigen Sie nun auf die PowerShell ISE um. Für die Arbeiten im Kapitel ist zumeist eine Anzahl von Zeilen einzugeben, was in der Konsole aufgrund der zeilenweisen Verarbeitung recht unkomfortabel ist.



Nehmen Sie die Eingaben im Skriptbereich im linken oberen Fensterbereich der PowerShell ISE vor. Sie sehen dort die Titelzeile mit dem Dateinamen *Unbenannt1.ps1\**. Das Zeichen **\*** zeigt einen nicht gespeicherten Zustand der eingegebenen Codes an: Entweder wurde Ihre Eingabe noch nicht als Datei gespeichert, oder eine bereits gespeicherte Datei wurde verändert.

Um eine Datei zu speichern, besitzen Sie die in Windows üblichen Möglichkeiten:

- ✓ Menü *Datei - Speichern* bzw. *Speichern unter*
- ✓ Tastenkombination **Strg S**
- ✓ Diskettensymbol in der Symbolleiste des Programms



Beim Speichern einer Datei verwenden Sie den Dateityp *.ps1*.

Um Code direkt aus dem Skriptbereich heraus auszuführen, markieren Sie den Bereich, den Sie ausführen wollen und drücken die Taste **F8**. Wollen Sie alle Eingaben im Skriptbereich ausführen, müssen Sie keine Markierung vornehmen, sondern nur die Taste **F5** drücken.

Beachten Sie, dass die Ausführung von Skripten auf Ihrem System erlaubt sein muss. Die Ausführungsrichtlinie von Skripten wurde ausführlich bei der Arbeit mit Profilen in Kapitel 6 angesprochen.



### Starten von Skripten

Um ein Skript zu starten, geben Sie den relativen oder absoluten Pfad zur Skriptdatei an:

- a) .\EigenesSkript.ps1
- b) C:\Daten\EigenesSkript.ps1

- ✓ In Beispiel a) wird ein Skript, das sich im aktuellen Verzeichnis befindet, mit einem relativen Pfadnamen adressiert.
- ✓ Beispiel b) zeigt den Aufruf eines Skripts mit absolutem Pfadnamen.

Enthält eine Pfadangabe Leerzeichen, setzen Sie sie in Anführungszeichen. Der Pfad wird nun von der PowerShell als Text verstanden, den Sie durch ein vorangestelltes „&“ ausführen können.

- a) "C:\Daten\Eigenes Skript.ps1"
- b) & "C:\Daten\Eigenes Skript.ps1"

- ✓ Beispiel a) gibt die in Anführungszeichen gesetzte Zeichenfolge aus.
- ✓ Beispiel b) führt durch das vorangestellte Zeichen „&“ die Zeichenfolge aus, startet also das Skript.

Ist in einer Pfadangabe neben einem Leerzeichen auch eine Variable enthalten, muss die Pfadangabe in doppelte Anführungszeichen gesetzt werden, da Variablen in einfachen Anführungszeichen nicht in ihren Wert umgewandelt werden:

- a) '\$HOME\Eigenes Skript.ps1'
- b) "\$HOME\Eigenes Skript.ps1"

- ✓ Beispiel a) ergibt die Zeichenfolge \$HOME\Eigenes Skript.ps1.
- ✓ Beispiel b) ergibt je nach aktuellem Benutzer z. B. die Zeichenfolge C:\Users\Administrator\Eigenes Skript.ps1.

## 8.2 Funktionen in der PowerShell

### Was sind Funktionen?

Wenn Sie einen einmal entwickelten Codeschnipsel wieder einsetzen wollen, müssen Sie sich an ihn erinnern und erneut eingeben. Um die Wiederverwendbarkeit zu erleichtern, können Sie Ihre „gute Idee“ unter einer Bezeichnung abspeichern und dann später über diese Bezeichnung aufrufen.

Cmdlets sind (kompilierte) Funktionen, die Sie als frei verfügbare, getestete Codeschnipsel wiederverwenden können.

Eine Funktion besteht aus einer Liste von Anweisungen, die Sie mit einer Bezeichnung versehen haben. Wenn Sie eine Funktion ausführen wollen, geben Sie die Bezeichnung (den sogenannten Funktionsnamen) ein. Die Liste der Anweisungen wird dann wie bei einer manuellen Eingabe an der Eingabeaufforderung ausgeführt.

Funktionen können über Parameter verfügen. Sie können Werte zurückgeben, die angezeigt, Variablen zugewiesen oder an andere Funktionen oder Cmdlets übergeben werden können.

Funktionen sind ein Teil des Skryptings. Sie erstellen ein Skript mit Funktionsdefinitionen und verwenden sie dann interaktiv oder in anderen Skripten.



Je allgemeiner und flexibler Sie Ihre Programmierung halten können, desto wertvoller wird eine Funktion für Sie sein. Überlegen Sie bei der Programmierung immer, welche Angaben Sie statisch vornehmen müssen. Es empfiehlt sich z. B., mit einer veränderbaren Variablen für ein Verzeichnis zu arbeiten, als es direkt anzugeben und damit fest zu programmieren.

## 8.3 Einfache Funktionen erstellen

Nützliche Funktionen können syntaktisch sehr einfach sein. Wenn Sie erstmals eine Funktion definieren, wird sie sich an der folgenden Syntax orientieren:

```
function <Name>
{
    <Anweisungsblock>
}
```

- ✓ Neben dem Schlüsselwort `function` und dem Namen der Funktion wird nur der Anweisungsblock in geschweiften Klammern definiert.

### Beispiel

```
function Get-Message
{
    Write-Host "Hallo, dies ist eine neue Funktion."
}
```

- ✓ Die Funktion erhält den Namen *Get-Message*. Einen Befehl dieses Namens kennt die PowerShell nicht, nach Ausführung des Codes steht dieser Befehl in der aktuellen Sitzung zur Verfügung.
- ✓ Der Anweisungsblock, der bei Aufruf der Funktion ausgeführt wird, ist einfach gehalten. Es handelt sich um eine kurze Meldung, die angezeigt werden soll.

Die komplette Funktionsdefinition muss einmal in der aktuellen PowerShell-Sitzung ausgeführt werden. Ab diesem Zeitpunkt steht Ihnen die Funktion zur Verfügung.

Um Funktionen zu verwenden, rufen Sie sie mit ihrem Namen auf. Eventuelle Parameter geben Sie jeweils durch Leerzeichen getrennt im Anschluss ein. Die Syntax eines Funktionsaufrufs gleicht der Syntax der Cmdlets:

```
<Funktionsname> [-Parameter1 [Wert1]] [-Parameter2 [Wert2]] [-ParameterN [WertN]]
```

Im obigen Beispiel wurden keine Parameter definiert. Sie geben *Get-Message* zum Ausführen der Funktion ein. Das Cmdlet basiert zwar auf einer von Ihnen definierten Funktion, Sie werden aber keinen Unterschied zur Verwendung anderer Cmdlets feststellen.

Es wird eine Namensgebung für Ihre Funktionen empfohlen, sie sich an den Vorgaben der PowerShell orientiert. Verwenden Sie die bekannte Kombination <Verb>-<Substantiv>, die Sie von den Cmdlets kennen. Orientieren Sie sich mit *Get-Verb* über die gebräuchlichen Verben und kombinieren Sie das passende Verb dann mit einem Substantiv. Dankbar ist auch eine Kennzeichnung eigener Cmdlets z. B. durch Verwendung der eigenen Initialen (hier etwa: *Get-Message\_AD*).



## 8.4 Funktionen mit Parametern erstellen

Sie können eine beliebige Anzahl von Parametern mit Funktionen verwenden, u. a. benannte Parameter und Switch-Parameter. Diese Parametertypen kennen Sie bereits von den vorgestellten Cmdlets. In den nachfolgenden Abschnitten lernen Sie, wie Sie benannte Parameter und Switch-Parameter in Funktionen definieren.

Sie können Parameter in den geschweiften Klammern mit dem Schlüsselwort `Param` mit folgender Syntax definieren:

```
function <Name>
{
    Param ($Parameter1[, $Parameter2])
    <Anweisungsblock>
}
```

Alternativ können Parameter außerhalb der geschweiften Klammern und ohne das Schlüsselwort `Param` nach folgender Syntax definiert werden:

```
function <Name> [($Parameter1[, $Parameter2])]
{
    <Anweisungsblock>
}
```

- ✓ Bestandteile einer Funktionsdefinition sind:
  - ✓ Das Schlüsselwort `function`
  - ✓ Der von Ihnen vergebene Funktionsname
  - ✓ Optional: beliebig viele Parameter der Funktion
  - ✓ Beliebig viele PowerShell-Befehle in geschweiften Klammern („{}“)

Beide Methoden sind gleichwertig, zwischen ihnen besteht kein Unterschied. Verwenden Sie die Methode, die für Sie nachvollziehbarer ist.

Wenn Sie die Funktion ausführen, wird einer Variablen mit dem Parameternamen der für einen Parameter angegebene Wert zugewiesen. Der Wert dieser Variable kann in der Funktion verwendet werden.

### Beispiel

Das folgende Beispiel stellt die selbst definierte Funktion `Get-LargeFiles` dar. Diese Funktion verfügt über einen Parameter `$length`. Mit der Funktion werden alle Dateien angezeigt, die größer als der Wert des Parameters sind:

```
function Get-LargeFiles ($length)
{
    Get-ChildItem C:\Windows | Where-Object {$_.length -ge $length}
```

- ✓ In der Funktionsdefinition wird ein Parameter `$length` definiert. Die Variable `$length` wird in der Funktionsdefinition bei der Formulierung der Bedingung `$_.length -ge $length` verwendet.
- ✓ Ein mit dem Funktionsaufruf angegebener Wert wird in der Funktionsdefinition mit diesem Variablenamen angesprochen.

Wollen Sie die Funktion verwenden, rufen Sie sie beispielsweise wie folgt auf:

- a) `Get-LargeFiles 500`
- b) `Get-LargeFiles 1MB`
- c) `Get-LargeFiles`

- ✓ Alle Beispiele rufen die Funktion `Get-LargeFiles` auf. In Beispiel a) wird eine Dateigröße von 500 Bytes vorgegeben, in Beispiel b) von 1 MB.
- ✓ Beispiel c) ruft die Funktion ohne den definierten Parameter auf. Dies führt nicht zu einer Fehlermeldung, sondern der Ausgabe des kompletten Verzeichnisinhalts. Es entspricht damit der Eingabe des Wertes 0.

## 8.5 Standardwert eines Parameters vorgeben

Um einen Standardwert für einen Parameter zu definieren, geben Sie nach dem Parameternamen den Zuweisungsoperator „=“ und den beabsichtigten Wert ein.

### Beispiel

Die Beispelfunktion `Get-LargeFiles` wird erweitert und um einen weiteren Parameter ergänzt. Beide Parameter erhalten einen vordefinierten Standardwert:

```
function Get-LargeFiles ($location = "C:\Windows", $length = 1MB)
{
    Get-ChildItem $location | Where-Object {$_.length -ge $length}
```

- ✓ Die Funktion erwartet die Eingabe eines Verzeichnisses, das durchsucht werden soll (Parameter \$location). Standardwert ist C:\Windows, der nur verwendet wird, falls keine Eingabe des Parameters erfolgt.
- ✓ Über den Parameter \$length steuern Sie, ab welcher Dateigröße Dateien im Verzeichnis angezeigt werden sollen. Geben Sie den Parameter nicht an, wird ein Wert von 1 MB verwendet.

Der Aufruf der Funktion könnte wie folgt aussehen:

- a) Get-LargeFiles
- b) Get-LargeFiles -location C:\Users\Administrator
- c) Get-LargeFiles -location F:\Daten -length 100kb

- ✓ Im Beispiel a) wird die Funktion mit den vordefinierten Standardwerten verwendet.
- ✓ In der Funktion im Beispiel b) wird der Pfad C:\Users\Administrator verwendet, als Dateigrößenvorgabe der Standardwert von 1 MB.
- ✓ Beispiel c) ist unter Verwendung aller definierten Parameter eingegeben worden. Die Funktion verwendet die im Aufruf angegebenen Werte F:\Daten und 100 kb.

## 8.6 Funktionen mit Switch-Parametern

Ein Switch-Parameter erfordert keinen Wert. Stattdessen geben Sie den Funktionsnamen und danach den Namen des Parameters ein. Wird der Parameter angegeben, erhält er den Wert \$True, wird er nicht angegeben, besitzt er den Wert \$False.

Wenn Sie einen Switch-Parameter definieren möchten, geben Sie den Typ [switch] vor dem Parameternamen an, wie Sie im folgenden allgemeinen Beispiel sehen:

```
function Test-Switch ([switch]$AnAus)
{
    If ($AnAus)
    {
        Write-Host "Eingeschaltet..."
    }
    Else
    {
        Write-Host "Ausgeschaltet..."
    }
}
```

- ✓ Der Switch-Parameter \$AnAus wird definiert. Der Typ [switch] zur Kennzeichnung eines Switch-Parameters wird angegeben.
- ✓ In der Funktionsdefinition wird mit einer If-Abfrage geprüft, ob der Parameter beim Funktionsaufruf angegeben wurde. Beim Aufruf der Funktion mit Angabe des Parameters wird die Meldung Eingeschaltet... ausgegeben, anderenfalls die Meldung Ausgeschaltet....

Sie können beim Ausführen der Funktion dem Switch-Parameter alternativ explizit einen booleschen Wert zuweisen, wie z. B.  
Test-Switch -AnAus:\$True bzw.  
Test-Switch -AnAus:\$False.

```
function1.ps1 x
1  function Test-Switch ([switch]$AnAus)
2  {
3      If ($AnAus)
4      {
5          Write-Host "Eingeschaltet..."
6      }
7      Else
8      {
9          Write-Host "Ausgeschaltet..."
10     }
11 }
```

PS C:\Users\Administrator> Test-Switch -AnAus  
Eingeschaltet...

PS C:\Users\Administrator> Test-Switch  
Ausgeschaltet...

Beispiel-Funktion mit Switch-Parameter

Erweitert man die bisher verwendete Beispielfunktion Get-LargeFiles um einen Switch-Parameter, könnte das z. B. wie folgt geschehen. Zur besseren Nachvollziehbarkeit werden die Änderungen fett gedruckt:

```
function Get-LargeFiles2 ($location = "C:\Windows", $length = 1MB,
[switch]$subdir)
{
    $schalter=$False
    If($subdir)
    {
        $schalter=$True
    }
    Get-ChildItem $location -Recurse:$schalter | Where-Object
    {$_.length -ge $length}
}
```

- ✓ Die bereits bekannte Funktion Get-LargeFiles wird um den Switch-Parameter \$subdir erweitert.
- ✓ In der Funktionsdefinition wird die Variable \$schalter mit dem Wert \$False initialisiert.
- ✓ Eine If-Abfrage ändert den Wert der Variablen \$schalter in \$True, falls der Parameter \$subdir existiert, also im Funktionsaufruf verwendet wurde.
- ✓ Die Angabe des Switch-Parameters im Funktionsaufruf steuert, ob das Cmdlet Get-ChildItem auch Unterverzeichnisse durchsucht. Wird der Switch-Parameter \$subdir verwendet, erhält der Parameter -Recurse des Cmdlets Get-ChildItem den Wert \$True. Somit werden auch Unterverzeichnisse in die Suche mit einbezogen.

## 8.7 Objekte über die Pipeline an eine Funktion übergeben

Jede Funktion kann Eingaben von der Pipeline akzeptieren. Wie eine Funktion Eingaben von der Pipeline verarbeitet, steuern Sie mit den Schlüsselwörtern Begin, Process und End. Die Schlüsselwörter werden wie folgt in der Funktion definiert:

```
function <Name>
{
    Begin
    {
        <Anweisungsblock>
    }
    Process
    {
        <Anweisungsblock>
    }
    End
    {
        <Anweisungsblock>
    }
}
```

- ✓ Zu Beginn der Funktionsausführung wird der Begin-Anweisungsblock einmalig ausgeführt.
- ✓ Der Process-Anweisungsblock wird für jedes Objekt ausgeführt, das der Funktion über die Pipeline übergeben wird. Während der Ausführung des Anweisungsblockes repräsentiert die Variable \$\_ das aktuelle Objekt.

- ✓ Nach der Verarbeitung aller Objekte wird der Anweisungsblock einmalig ausgeführt, der durch das Schlüsselwort `End` eingeleitet wird.
- ✓ Falls die Schlüsselwörter `Begin`, `Process` und `End` nicht verwendet werden, werden alle Anweisungen wie ein `End`-Anweisungsblock behandelt.

### Beispiel `Process`

Für die folgende Funktion wird nur das Schlüsselwort `Process` verwendet. Ziel der Funktion ist es, den Wert der Objekte anzuzeigen, die über die Pipeline übergeben wurden.

Die Definition der Funktion entnehmen Sie dem folgenden Beispiel:

```
function Get-Pipe1
{
    Process
    {
        Write-Host "Wert des Pipeline-Objekts: $_."
    }
}
```

Wenn die Funktion Objekte über die Pipeline empfängt und verarbeitet, könnte ein Aufruf wie folgt aussehen:

```
1, 2, 3, 4, 5, 6, 7, 8 | Get-Pipe1
```

- ✓ Es werden nacheinander die Zahlen 1 bis 8 über die Pipeline übergeben.
- ✓ Die Funktion `Get-Pipe1` führt für jedes übergebene Objekt den `Process`-Anweisungsblock aus und erzeugt eine Meldung mit dem Wert des aktuellen Objekts.

### Beispiel `Begin` und `End`

Wenn Sie eine Funktion in einer Pipeline verwenden, werden die über die Pipeline an die Funktion übergebenen Objekte automatisch der Variablen `$input` zugewiesen. Diese beinhaltet am Ende der Funktionsausführung die Werte aller übergebenen Objekte.

Das folgende Beispiel veranschaulicht die automatische Variable `$input` mit dem `End`-Schlüsselwort:

```
function Get-Pipe1a
{
    Begin
    {
        Write-Host "Pipeline-Funktion (Werte: $input)"
    }
    End
    {
        Write-Host "Verarbeitete Werte: $input"
    }
}
```

- ✓ In der Funktion wird ein `Begin`-Anweisungsblock definiert, der eine Ausgabe vor der Verarbeitung übergebener Objekte vornimmt. Der Wert der Variablen `$input` soll ausgegeben werden. Vor der Verarbeitung der einzelnen Objekte besitzt die Variable allerdings noch keinen Wert.
- ✓ Der `End`-Anweisungsblock wird nach der Verarbeitung aller Objekte ausgeführt. Die Variable `$input` wird ausgegeben.

Sie können die Funktion z. B. über die Pipeline wie folgt aufrufen:

```
1, 2, 3, 4, 5, 6, 7, 8 | Get-Pipe1a
```

- ✓ Sie erhalten die erwartete Ausgabe:

```
Pipeline-Funktion (Werte: )
Verarbeitete Werte: 1 2 3 4 5 6 7 8
```

### Beispiel Begin, Process und End

In einem abgewandelten Beispiel sehen Sie die Definition der Funktion mit Begin-, Process- und End-Anweisungsblöcken:

```
function Get-Pipe2
{
    Begin
    {
        Write-Host "Pipeline-Funktion"
    }
    Process
    {
        Write-Host "Wert des aktuellen Pipeline-Objekts: $_."
    }
    End
    {
        Write-Host "Verarbeitete Werte: $input."
    }
}
```

- ✓ Das aktuelle Objekt können Sie im Process-Anweisungsblock über die Variablen `$_` oder `$input` ansprechen.
- ✓ Falls Sie den Process-Anweisungsblock verwenden, ist die Variable `$input` im End-Anweisungsblock leer.
- ✓ Nur wenn Sie den Process-Anweisungsblock weglassen, können Sie im End-Anweisungsblock mit der Variablen `$input` auf die Werte der übergebenen Objekte zugreifen.
- ✓ Daraus folgt: Wollen Sie im End-Anweisungsblock die Variable `$input` verwenden, müssen Sie den Process-Anweisungsblock weglassen.

```
1  function Get-Pipe2
2  {
3      Begin
4      {
5          Write-Host "Pipeline-Funktion"
6      }
7      Process
8      {
9          Write-Host "Wert des aktuellen Pipeline-Objekts: $_."
10     }
11     End
12     {
13         Write-Host "Verarbeitete Werte: $input."
14     }
15 }
```

```
PS C:\Users\AD> 1, 2, 3, 4, 5, 6, 7, 8 | Get-Pipe2
Pipeline-Funktion
Wert des aktuellen Pipeline-Objekts: 1.
Wert des aktuellen Pipeline-Objekts: 2.
Wert des aktuellen Pipeline-Objekts: 3.
Wert des aktuellen Pipeline-Objekts: 4.
Wert des aktuellen Pipeline-Objekts: 5.
Wert des aktuellen Pipeline-Objekts: 6.
Wert des aktuellen Pipeline-Objekts: 7.
Wert des aktuellen Pipeline-Objekts: 8.
Verarbeitete Werte: .
```



Bei den bisherigen Beispielen orientiert sich die Benennung eigener Funktionen an der Standardbenennung von Microsoft in der Form *Verb-Substantiv*. Wenn Sie häufiger selbst oder gemeinsam mit mehreren Kollegen Funktionen erstellen, empfiehlt es sich, den Funktionsnamen z. B. um die eigenen Initialen zu ergänzen (Beispiele für individualisierte Namen: `Get-AD_Shell`, `Set-AD_Service`). So erstellen Sie sprechende Namen und vermeiden dabei Namensgleichheit mit eingebauten Funktionen. Darüber hinaus können Sie damit sehr schnell alle eigenen Funktionen (hier: `Get-Command -Noun AD_*`) finden.

## 8.8 Objekte über die Pipeline an ein Skript übergeben

Wenn Sie Objekte über die Pipeline an ein Skript übergeben wollen, erstellen Sie ein Skript nach demselben Muster wie eben für Funktionen gezeigt. Sie verwenden im Skript einen `Process`-Anweisungsblock zur Verarbeitung jedes Objekts, das über die Pipeline an das Skript übergeben wird.

**Beispiel:** *pipescript.ps1*

**Plus** **Beispieldatei:** *pipescript.ps1* (Ordner Kapitel 08)

Ein Skript soll Objekte über die Pipeline erhalten und verarbeiten. Dateien sollen mit einer anderen Hintergrundfarbe angezeigt werden, wenn die Datei mindestens 1 MB groß ist.

```

① Begin
{
    Write-Host "Große Dateien vor rotem Hintergrund:"
}

② Process
{
    ③ If ($_.length -lt 1MB)
    {
        $_
    }
    ④ Else
    {
        $Standardfarbe = $host.UI.RawUI.BackgroundColor
        $host.UI.RawUI.BackgroundColor = "Red"
        $_
        $host.UI.RawUI.BackgroundColor = $Standardfarbe
    }
}

⑤ End
{
    Write-Host "Ende der Verarbeitung."
}

```

- ① Das Skript beginnt mit einem optionalen `Begin`-Anweisungsblock. Bevor Objekte über die Pipeline empfangen werden, wird eine Meldung ausgegeben.
- ② Der `Process`-Anweisungsblock verarbeitet die einzelnen Objekte. Hier finden die eigentlichen Pipelineoperationen statt.
- ③ In einer `If`-Abfrage wird geprüft, ob das aktuell übergebene Objekt eine Dateigröße von weniger als 1 MB aufweist. Ist dies der Fall, wird das Objekt unverändert ausgegeben.
- ④ Ist das übergebene Objekt mindestens 1 MB groß, werden die Anweisungen des `Else`-Zweigs ausgeführt.
- ⑤ In der Variablen `$host.UI.RawUI.BackgroundColor` ist die aktuelle Hintergrundfarbe der Konsole hinterlegt. Diese wird der Variablen `$Standardfarbe` zugewiesen. Dann wird die Hintergrundfarbe in den Wert *rot (Red)* geändert. Das aktuelle Objekt wird auf rotem Hintergrund ausgegeben. Letztlich wird die Hintergrundfarbe wieder auf den vorher gültigen Wert gesetzt.
- ⑥ Nachdem alle über die Pipeline übergebenen Objekte verarbeitet wurden, wird am Ende des Skripts im optionalen `End`-Anweisungsblock eine weitere Meldung ausgegeben.

Sie können das Skript z. B. wie folgt aufrufen:

```
Get-ChildItem C:\Windows | .\pipescript.ps1
```

- ✓ Der Aufruf setzt voraus, dass sich das Skript im aktuellen Verzeichnis befindet.
- ✓ Jedes einzelne Objekt, das über die Pipeline übergeben wird, wird auf seine Größe geprüft und entsprechend den Anweisungen dargestellt.

## 8.9 Filter

Als Filter wird ein spezieller Funktionstyp bezeichnet, der für jedes Objekt in der Pipeline ausgeführt wird. Filter ähneln Funktionen, bei denen sich alle Anweisungen in einem `Process`-Block befinden.

Für Filter wird die folgende allgemeine Syntax verwendet:

```
filter <Name>
{
    <Anweisungsblock>
}
```

### Beispiel

Mit dem folgenden Filter werden Daten eines Verzeichnisses nur dann angezeigt, wenn sie dem angegebenen Filter entsprechen.

Die Filterdefinition sehen Sie hier:

```
filter NewFiles ([int]$stage)
{
    $jetzt = Get-Date
    $_ | Where-Object { ($jetzt - $_.LastWriteTime).Days -lt $stage}
```

- ✓ Ein Filter mit dem Namen `NewFiles` wird definiert. Bei der Nutzung des Filters wird ein Parameter `$stage` erwartet, der Ganzahlen akzeptiert (Anzahl von Tagen).
- ✓ Der Variablen `$jetzt` wird das aktuelle Datum als Wert zugewiesen.
- ✓ Jedes über die Pipeline empfangene Objekt (`$_`) wird mithilfe des Cmdlets `Where-Object` gefiltert. Es wird die Differenz aus dem aktuellen Datum und dem Datum der letzten Änderung am Objekt in Tagen berechnet. Falls dieser Wert kleiner als die im Filteraufruf eingegebene Anzahl an Tagen ist, wird das Objekt durch den Filter gelassen und angezeigt.

Ein möglicher Aufruf des Filters könnte so aussehen:

```
Get-ChildItem <Pfad> | NewFiles 20
```

- ✓ Im angegebenen Pfad werden alle Dateien angezeigt, deren letzte Aktualisierung weniger als 20 Tage her ist.

## 8.10 Das virtuelle Laufwerk *Function*:

### Funktionen und Filter anzeigen

Alle Funktionen und Filter in der PowerShell werden automatisch auf dem Laufwerk *Function*: gespeichert. Dieses Laufwerk wird vom PowerShell-Provider *Function* zur Verfügung gestellt.

Um alle Funktionen und Filter der aktuellen PowerShell-Sitzung anzuzeigen, geben Sie folgenden Befehl ein:

```
Get-ChildItem Function:
```

- ✓ In der Spalte der Eigenschaft  *CommandType* wird angezeigt, bei welchem Element es sich um einen Filter oder eine Funktion handelt.
- ✓ Die Spalte der Eigenschaft *Name* zeigt den Namen des Filters bzw. der Funktion.

### Definition von Funktionen und Filtern anzeigen

Die Definition einer Funktion wird als Skriptblock in der Eigenschaft *Definition* der Funktion gespeichert. Wenn Sie die Definition einer Funktion anzeigen möchten, geben Sie folgenden Befehl ein:

```
(Get-ChildItem Function:\<Funktionsname>) .Definition
```

- ✓ Ersetzen Sie <Funktionsname> durch den Namen der Funktion, dessen Programmierung Sie am Bildschirm sehen wollen, z. B. (Get-ChildItem Function:\Get-Pipe2) .Definition.

### Eigene Funktionen und Filter löschen

Zum Löschen eigener Funktionen und Filter verwenden Sie das bekannte Cmdlet *Remove-Item* mit dem Parameter *-Path* zur exakten Bezeichnung des gewünschten Elements.

Mit dieser Methode können Sie auch vordefinierte Funktionen und Filter für die aktuelle PowerShell-Sitzung entfernen. Diese stehen aber beim Start einer neuen PowerShell-Sitzung wieder zur Verfügung.

## 8.11 Eigene Programmierung dokumentieren

Wenn Sie etwas programmieren, sollten Sie es gut dokumentieren. Andere nutzen Ihre Programmierung und möchten sich orientieren, oder Sie selbst kennen nach einer Weile nicht mehr alle Details z. B. einer von Ihnen programmierten Funktion.

Die PowerShell bietet die Möglichkeit einer Dokumentation Ihrer Programmierung, indem Sie am Anfang Ihres Skripts, Ihrer Funktion oder Ihres Filters einen Kommentarblock nach folgendem Muster einfügen:

①	<#
②	.SYNOPSIS
③	Kurzbeschreibung
④	.DESCRIPTION
	Ausführliche Beschreibung

④	.PARAMETER <ParameterName-1> Beschreibung des ersten Parameters .PARAMETER <ParameterName-N> Beschreibung des n. Parameters
⑤	.EXAMPLE Beispielanwendung und -erläuterung .EXAMPLE Weitere Beispielanwendung und -erläuterung
⑥	.NOTES Weitere Hinweise
⑦	.LINK Angabe von URLs oder ähnlichen Cmdlets #>

- ① An den Beginn Ihres Skripts, Ihrer Funktion oder Ihres Filters wird ein Kommentarblock eingefügt:  
<# ... #>. In diesem Kommentarblock erfolgen alle weiteren Definitionen.
- ② Die Überschrift eines Eintrags wird durch einen Punkt eingeleitet, gefolgt vom Namen des Eintrags. In einer oder mehreren eigenen Zeilen tragen Sie den Text für den Eintrag ein. Beim Eintrag .SYNOPSIS handelt es sich um die Kurzbeschreibung Ihrer Programmierung.
- ③ Der Eintrag .DESCRIPTION ist der ausführlichen Beschreibung zugeordnet.
- ④ Der Eintrag .PARAMETER wird für die Beschreibung eines programmierten Parameters verwendet. Der Eintrag wird für jeden weiteren Parameter mit Angaben zu diesem Parameter wiederholt.
- ⑤ Im Eintrag .EXAMPLE präsentieren Sie ein Anwendungsbeispiel für Ihre Programmierung. Verwenden Sie die erste Zeile für das Beispiel selbst, eine weitere für die Beschreibung des Beispiels. Wenn Sie mehrere Beispiele angeben wollen, wiederholen Sie den Eintrag entsprechend.
- ⑥ .NOTES erwartet weitere Hinweise zu Ihrer Programmierung.
- ⑦ Im Eintrag .LINK geben Sie bei Bedarf Links für interessante Webseiten oder ähnliche Cmdlets an, die in diesem Zusammenhang von Interesse sind.

Wollen Sie einen Eintrag nicht verwenden, lassen Sie ihn einfach weg.

Mit dieser Dokumentation steht Ihnen die bekannte PowerShell-Hilfe auch für Ihre Programmierung zur Verfügung. Die Eingabe Get-Help <Skript/Funktion/Filter> liefert die angegebenen Einträge in der bekannten Struktur. Get-Help ... -Example z. B. liefert die von Ihnen vorgenommenen .EXAMPLE-Einträge.

**Beispiel:** *dokuscript.ps1*

**Plus** **Beispieldatei:** *dokuscript.ps1* (Ordner Kapitel 08)

Im folgenden Beispiel sehen Sie eine Dokumentation für die weiter oben im Kapitel definierte Funktion Get-LargeFiles.

①	function Get-LargeFiles (\$location = "C:\Windows", \$length = 1MB) {
---	--

```

② <#
    .SYNOPSIS
        Große Dateien anzeigen
    .DESCRIPTION
        Standardmäßig werden große Daten ab 1 MB im Windows-Verzeichnis angezeigt. Sie können das Verzeichnis und die Dateigröße durch Verwendung von Parametern selbst steuern.
    .PARAMETER location
        Angabe eines Verzeichnisses, das durchsucht werden soll.
    .PARAMETER length
        Festlegung der minimalen Dateigröße.
    .EXAMPLE
        Get-LargeFiles
        Die vordefinierten Standardwerte werden verwendet: Das Windows-Verzeichnis wird durchsucht, die minimale Dateigröße ist 1 MB.
    .EXAMPLE
        Get-LargeFiles -location C:\Users\Administrator
        In der Funktion wird der Pfad C:\Users\Administrator verwendet, als Dateigrößenvorgabe der Standardwert von 1 MB.
    .EXAMPLE
        Get-LargeFiles -location F:\Daten -length 100kb
        Die Funktion verwendet die im Aufruf angegebenen Werte F:\Daten und 100kb.
    .LINK
        http://beispiel.url
        Get-ChildItem
    #>
③     Get-ChildItem $location | Where-Object {$_ .length -ge $length}
}

```

- ① Die Funktionsdefinition wird durch das Schlüsselwort `function` eingeleitet.
- ② Zu Beginn der Funktion wird der Kommentarblock mit der Dokumentation eingegeben.
- ③ Schließlich folgt der Anweisungsblock der Funktion.

## 8.12 Funktionen und Filter allgemein verfügbar machen

Wenn Sie an der PowerShell-Eingabeaufforderung eine Funktionsdefinition oder die Definition eines Filters aufrufen, werden Funktion oder Filter Teil der aktuellen Sitzung und sind bis zum Ende der aktuellen PowerShell-Sitzung verfügbar.

Wenn Sie eine eigene Definition in allen Windows PowerShell-Sitzungen verwenden möchten, fügen Sie sie einem PowerShell-Profil hinzu.

Sie können die Definition auch in einer PowerShell-Skriptdatei speichern, die Sie während Ihrer aktuellen PowerShell-Sitzung manuell ausführen. Geben Sie die Definition in einer Textdatei ein und speichern Sie die Datei dann mit der Dateinamenerweiterung `.ps1`.

Es ist sinnvoll, mehrere Funktionen zusammen in einem Skript zu speichern. Speichern Sie in den Skripten Ihre Funktionen nach Anwendungsgebiet wie z. B. allgemein, Dateifunktionen etc. So können Sie Ihre Funktionen schnell laden, und Ihre Definitionen stehen zur Verfügung.



## 8.13 Kurz zusammengefasst

Möchten Sie Befehle oder Befehlsfolgen wieder verwenden, empfiehlt es sich, den Code in ein Skript zu schreiben oder eine Funktion zu definieren. Ziel dabei ist immer, die Programmierung möglichst allgemein zu halten. So können Sie sich im Laufe der Zeit eine Bibliothek nützlicher Codeschnipsel erstellen. Funktionen sind in dieser Hinsicht besonders flexibel, da sie sich für den Benutzer wie die bekannten Cmdlets präsentieren. Mit den Programmiergrundlagen, Funktionen und Filtern kennen Sie das gesamte Grundgerüst, das Sie für das Skripting benötigen.

Verwenden Sie zur Erstellung von Skripten und mehrzeiligen Eingaben die PowerShell ISE oder einen anderen Editor, der die PowerShell-Syntax unterstützt. Vergessen Sie zudem nicht, Ihre eigene Programmierung ausreichend zu dokumentieren.

## 8.14 Übung

### Mit Benutzerprofilen arbeiten

Übungsdatei: –

Ergebnisdatei: **8\_ergebnisse.txt, Meine Funktionen.ps1**

1. Erstellen Sie mit der PowerShell ISE ein Skript und speichern Sie es unter dem Dateinamen *Meine Funktionen.ps1* im aktuellen Benutzerverzeichnis (`$HOME`). Erstellen Sie in der Datei einen Befehl zur Ausgabe der Zeichenfolge "Hier sind meine Funktionen gespeichert." und führen Sie die Datei von der PowerShell-Konsole aus.
2. Fügen Sie Ihrem Skript eine neue Funktion `Get-MyProcess` hinzu, die Prozesse auf Ihrem lokalen Rechner unter folgenden Bedingungen anzeigt:
  - ✓ Die anzuzeigenden Prozesse beginnen mit dem Buchstaben *s*.
  - ✓ Die Spalten *ID*, *ProcessName* und *CPU* sollen angezeigt werden.
  - ✓ Die Anzeige soll absteigend nach der Eigenschaft *CPU* sortiert werden.Speichern Sie Ihr Skript, führen Sie es erneut aus und testen Sie die Funktion.
3. Kopieren Sie die Funktion `Get-MyProcess` und fügen Sie sie unter dem Namen `Get-MyProcess2` in Ihr Skript ein. Wandeln Sie die Funktion wie folgt ab:
  - ✓ Die Funktion soll einen Parameter `$FilterName` akzeptieren, der bestimmt, welche Prozesse angezeigt werden. Standardwert: Es sollen Prozesse angezeigt werden, deren Name mit dem Buchstaben *s* beginnt.
  - ✓ Über einen Switch-Parameter `$Liste` soll gesteuert werden, die Ausgabe als Liste zu sehen (Format-List) und nicht als Tabelle.Speichern Sie Ihr Skript, führen Sie es erneut aus und testen Sie die neue Funktion.
4. Kopieren Sie die Funktion `Get-MyProcess2` und fügen Sie sie unter dem Namen `Get-MyProcess3` in Ihr Skript ein.
  - ✓ Entfernen Sie die Programmierung des Switch-Parameters `$Liste`.
  - ✓ Erstellen Sie stattdessen einen Filter *Liste*, der die Aufgabe übernimmt, übergebene Objekte als Liste darzustellen.Speichern Sie Ihr Skript, führen Sie es erneut aus und testen Sie die abgeänderte Funktion zusammen mit dem definierten Filter.
5. Erstellen Sie in Ihrem Skript eine Funktion `Optimize-Output`, die Objekte des Befehls `Get-Process` über die Pipeline verarbeiten kann. Richten Sie sich nach folgenden Vorgaben:
  - ✓ Vor der Verarbeitung soll das aktuelle Datum ausgegeben werden.
  - ✓ Für die Verarbeitung der Objekte über die Pipeline übernehmen Sie die Auswahl der Eigenschaften und die Sortierung aus den bereits erstellten Funktionen des Skripts.
  - ✓ Nach der Verarbeitung soll die Meldung "Aufgabe erledigt" in schwarzer Schrift auf weißem Hintergrund ausgegeben werden.
6. Erstellen Sie eine kurze Dokumentation über die Funktion `Get-MyProcess2` und fügen Sie diese der Funktionsdefinition im Skript hinzu. Speichern Sie Ihr verändertes Skript, führen Sie es erneut aus und testen Sie die eingebaute Dokumentation.

# 9 PowerShell-Module



Beispieldatei: 9\_kompletter code.txt (Ordner Kapitel 09)

## 9.1 Was sind PowerShell-Module?

Module können als Erweiterungen der PowerShell verstanden werden. Es handelt sich um themenspezifische Sammlungen von Befehlen. Teilweise wird mit dem Laden eines Moduls auch ein PowerShell-Provider definiert, der ein neues PowerShell-Laufwerk definiert. Alle PowerShell-Cmdlets entstammen grundsätzlich Modulen.

Das Modul *Microsoft.PowerShell.Core* beinhaltet einen Kernsatz an Cmdlets, Funktionen und Providern, die direkt mit der PowerShell installiert werden und damit immer zur Verfügung stehen. Weitere Module können zu jedem Zeitpunkt einer PowerShell-Sitzung nachgeladen werden.

Microsoft hat den Anspruch formuliert, dass es mit der PowerShell möglich sein soll, sowohl das eigene System als auch das Netzwerk vollständig verwalten zu können. In den Vorgängerversionen der PowerShell ist dieser Anspruch nur zu einem Bruchteil umgesetzt worden. Mittlerweile sind viele blinde Flecken beseitigt worden. In Windows Server 2016 (oder 2019) stehen direkt nach der Installation bereits mehr als 80 Module zur Verfügung, in aktuellen Windows 10-Versionen ebenfalls knapp 80. Bei Windows 7 waren es z. B. noch 17 Module.

Der grundlegende PowerShell-Befehlssatz steht auf den modernen Betriebssystemen Windows Server 2016 (oder 2019) und Windows 10 zur Verfügung und wird bereits beim Start der PowerShell durch Basis-Module wie *Microsoft.PowerShell.Management* automatisch geladen.

Module sind in eigenen Unterverzeichnissen im Verzeichnis *Modules* des Installationsverzeichnisses der PowerShell sowie im Programme-Ordner (unter *\WindowsPowerShell\Modules*) als Dateien gespeichert. Ein zusätzlicher Pfad zu verwendbaren – vielleicht auch selbst erstellten – Modulen ist benutzerabhängig in der Struktur unterhalb des eigenen *Documents*-Verzeichnisses zu finden.

Um die möglichen Speicherorte auszulesen, lesen Sie in der PowerShell den Wert der Umgebungsvariablen *PSModulePath* aus:

```
$Env:PSModulePath -split ";"
```

In der Befehlszeile sorgt der Parameter *-split* dafür, die jeweils durch Semikolon getrennte Liste an diesem Zeichen zu trennen und die einzelnen Werte zeilenweise darzustellen.

## 9.2 Mit Modulen arbeiten

### Get-Module: Welche Module sind geladen bzw. standardmäßig vorhanden?

Geben Sie nun den folgenden Befehl ein, sehen Sie die aktuell geladen Module:

```
Get-Module
```

ModuleType	Version	Name	ExportedCommands
Manifest	3.1.0.0	Microsoft.PowerShell.Management	{Add-Computer, Add...
Manifest	3.0.0.0	Microsoft.PowerShell.Security	{ConvertFrom-Secur...
Manifest	3.1.0.0	Microsoft.PowerShell.Utility	{Add-Member, Add-T...
Manifest	3.0.0.0	Microsoft.WSMan.Management	{Connect-WSMan, Di...
Script	1.1	PSReadline	{Get-PSReadlineKey...

Liste der geladenen Module in der aktuellen PowerShell-Sitzung am Beispiel von Windows 10

Zur Anzeige aller verfügbaren Module, die geladen sind oder nachgeladen werden können, erweitern Sie den Befehl um den Parameter `-ListAvailable`:

```
Get-Module -ListAvailable
```

ModuleType	Version	Name	ExportedCommands
Binary	1.0.0.1	PackageManagement	{Find-Package, Get...
Script	3.3.5	Pester	{Describe, Context...
Script	1.0.0.1	PowerShellGet	{Install-Module, F...
Script	1.1	PSReadline	{Get-PSReadlineKey...

ModuleType	Version	Name	ExportedCommands
Manifest	1.0.0.0	AppBackgroundTask	{Disable-AppBackgr...
Manifest	2.0.0.0	AppLocker	{Get-AppLockerFile...
Manifest	2.0.0.0	Appx	{Add-AppxPackage, ...
Script	1.0.0.0	AssignedAccess	{Clear-AssignedAcc...
Manifest	1.0.0.0	BitLocker	{Unlock-BitLocker,...
Manifest	2.0.0.0	BitsTransfer	{Add-BitsFile, Com...
Manifest	1.0.0.0	BranchCache	{Add-BCDataCacheEx...
Manifest	1.0.0.0	CimCmdlets	{Get-CimAssociated...
Manifest	1.0.0.0	CIPolicy	ConvertFrom-CIPolicy

Ausschnitt der Liste verfügbarer Module (Windows 10)

Welche Module zur Verfügung stehen, hängt von verschiedenen Faktoren ab:

- ✓ Betriebssystem
- ✓ Installation von Betriebssystemkomponenten
- ✓ Installation von Rollen und Features (Windows Server)

Auf einem Windows Server 2016 (oder 2019) Domänencontroller, der auch als DNS-Server fungiert, finden Sie etwa 90 Module, u. a.:

- |  |  |
|--|--|
| <ul style="list-style-type: none"> <li>✓ <i>ActiveDirectory*</i></li> <li>✓ <i>ADDSDeployment*</i></li> <li>✓ <i>AppLocker</i></li> <li>✓ <i>BestPractices*</i></li> <li>✓ <i>BranchCache</i></li> <li>✓ <i>DnsClient</i></li> <li>✓ <i>DnsServer*</i></li> <li>✓ <i>GroupPolicy*</i></li> </ul> | <ul style="list-style-type: none"> <li>✓ <i>NetTCPIP</i></li> <li>✓ <i>PKI</i></li> <li>✓ <i>PrintManagement</i></li> <li>✓ <i>RemoteDesktop*</i></li> <li>✓ <i>ScheduledTasks</i></li> <li>✓ <i>ServerCore*</i></li> <li>✓ <i>ServerManager*</i></li> <li>✓ <i>VpnClient</i></li> </ul> |
|--|--|

Die mit Stern (\*) gekennzeichneten Module stehen nur auf Windows Server 2016 (oder 2019) zur Verfügung, nicht auf dem Client-Betriebssystem Windows 10.

Sie können die Verwaltung Ihrer Umgebung an den Servern direkt oder von anderen Servern mit installiertem Feature *Remoteserver-Verwaltungstools* vornehmen. Auch Clients können Sie einsetzen, nachdem Sie sich aus dem Internet – passend zu Ihrem Client-Betriebssystem – die Microsofts Remoteserver-Verwaltungstools (Remote Server Administration Tools, RSAT) geladen und installiert haben (Download für Windows 10: <https://www.microsoft.com/de-de/download/details.aspx?id=45520>). In dem Fall stehen Ihnen auch auf dem Client – die entsprechenden administrativen Berechtigungen vorausgesetzt – die serverspezifischen Module zur Verfügung.

Die Installation weiterer Rollen und Features sowie weiterer Komponenten stellt weitere Module zur Verfügung. Microsoft geht konsequent den Weg, dass installierte Funktionen (auch) über die PowerShell verwaltet werden können, und liefert in der Regel zusätzliche PowerShell-Module mit entsprechenden Befehlssätzen.

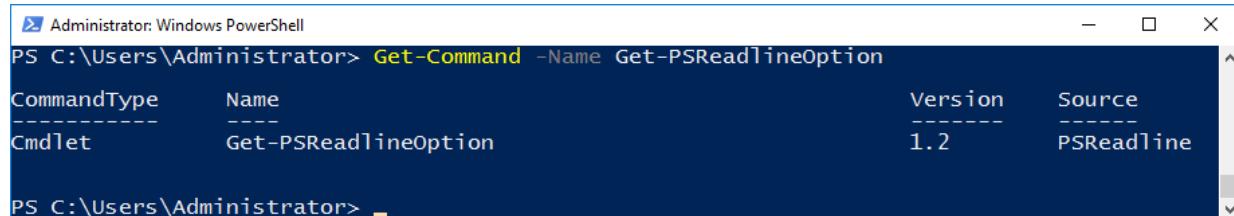
Produkte wie Exchange Server oder SharePoint Server besitzen ihre eigene Verwaltungs-Shell, die auf der PowerShell basiert und um Befehle speziell für diese Produkte erweitert ist. Auch hier werden Module (oder ältere Snap-Ins) zusätzlich geladen, um die speziellen Befehle zu ergänzen.

## Befehle Modulen zuordnen

Sie können mit den folgenden Methoden überprüfen, welchem Modul die einzelnen Cmdlets zuzuordnen sind:

- Wollen Sie die Modulzugehörigkeit eines speziellen Befehls (nicht nur Cmdlet) überprüfen, geben Sie Folgendes ein:

```
Get-Command -Name <Cmdlet>
```



CommandType	Name	Version	Source
Cmdlet	Get-PSReadlineOption	1.2	PSReadline

Modulzugehörigkeit des angegebenen Cmdlets

In der Spalte *Source* erhalten Sie die Information zur Modulzugehörigkeit des angegebenen Befehls, in der Spalte *CommandType* die Angabe zum Befehlstyp (Alias, Function, Cmdlet, Application).

- Wenn Sie sich eine allgemeine Übersicht über die Cmdlets und ihre Modulzugehörigkeit verschaffen wollen, sollten Sie **nicht** einfach `Get-Command -Name *` eingeben. Dies ist zum einen recht unübersichtlich durch abgeschnittene Spalten der automatisch formatierten Tabelle, zum anderen erhalten Sie im Ergebnis auch die anderen Befehlstypen. Hier hilft die folgende Zeile:

```
Get-Command - CommandType Cmdlet | Select-Object -Property Name, ModuleName
```

In dem Beispiel wird der Parameter `-CommandType` mit dem Wert *Cmdlet* verwendet, um die Suche auf Cmdlets zu beschränken. Durch die nachfolgende Filterung mithilfe des Cmdlets `Select-Object` erhalten Sie als Ausgabe eine gut lesbare Tabelle mit den Spalten *Name* und *ModuleName*.

Ausschnitt der Ausgabe mehrerer hunderter Cmdlets

In der Ausgabe sehen Sie, dass die nötigen Module für den Großteil der angezeigten Cmdlets nicht geladen sind. Ihnen stehen zwei alternative Vorgehensweisen zur Verfügung, um mit diesen Cmdlets zu arbeiten:

- ✓ Rufen Sie das betreffende Cmdlet direkt auf.

```
Administrator: Windows PowerShell
PS C:\Users\Administrator> Get-Module

ModuleType Version Name
---- - - -
Manifest 3.1.0.0 Microsoft.PowerShell.Management
Manifest 3.1.0.0 Microsoft.PowerShell.Utility
Script 1.2 PSReadline

ExportedCommands
{Add-Computer, Add-Content, ...
{Add-Member, Add-Type, Clear...
{Get-PSReadlineKeyHandler, G...

PS C:\Users\Administrator> Resolve-DnsName -Name Server2016

Name
-----
Server2016.herdt.ps
Server2016.herdt.ps

Type TTL Section IPAddress
----- - - -
AAAA 1200 Question fe80::adae:561:d28
:5af6
A 1200 Question 192.168.142.139

PS C:\Users\Administrator> Get-Module

ModuleType Version Name
---- - - -
Manifest 1.0.0.0 DnsClient
Manifest 3.1.0.0 Microsoft.PowerShell.Management
Manifest 3.1.0.0 Microsoft.PowerShell.Utility
Script 1.2 PSReadline

ExportedCommands
{Resolve-DnsName, Add-DnsCli...
{Add-Computer, Add-Content, ...
{Add-Member, Add-Type, Clear...
{Get-PSReadlineKeyHandler, G...
```

Im oberen Teil der Abbildung sehen Sie die geladenen Module (hier auf einem Windows Server 2016-Rechner) vor der Ausführung des Cmdlets `Resolve-DnsName`, das Bestandteil des Moduls `DnsClient` ist. Danach wird das Cmdlet ausgeführt und anschließend nochmals geprüft, welche Module geladen wurden. Module werden bei Bedarf automatisch nachgeladen, wenn Sie ein Cmdlet eines noch nicht geladenen Moduls aufrufen.

- ✓ Importieren Sie zuerst das Modul und arbeiten Sie anschließend mit den neuen Befehlen, die das Modul bereitstellt, wie Sie nachfolgend sehen.

## Import-Module: Module in der aktuellen Sitzung laden

Nachdem Sie sich mithilfe des Befehls `Get-Module -ListAvailable` informiert haben, welche Module zur Verfügung stehen und wie der genaue Name des zu ladenden Moduls lautet, laden Sie das gewünschte Modul mit dem folgenden Befehl:

```
Import-Module -Name <Modulname> [, <Modulname2>]
```

Mit dem Befehl können Sie ein Modul oder gleichzeitig mehrere, durch Kommata getrennte Module laden.

Wie üblich stehen Ihnen die geladenen Module bis zum Ende der aktuellen PowerShell-Sitzung zur Verfügung. Häufig wird deshalb dieser Befehl in Profilskripten eingesetzt, damit Ihre individuell angepasste Umgebung immer direkt nach dem Start einer PowerShell-Sitzung zur Verfügung steht.

## Remove-Module: Modul aus der aktuellen Sitzung entfernen

Wenn Sie ein Modul geladen haben, es aber wieder in der laufenden Sitzung entfernen wollen, verwenden Sie das Cmdlet `Remove-Module`. Es arbeitet nach denselben Regeln wie das Cmdlet `Import-Module`:

```
Remove-Module -Name <Modulname> [, <Modulname2>]
```

Mit diesem Befehl können Sie ein Modul oder gleichzeitig mehrere, durch Kommata getrennte Module aus der aktuellen PowerShell-Sitzung entfernen.

## Welche neuen Befehle bringt ein spezielles Modul?

Die vorrangigste Frage, die sich nach dem Import eines neuen Moduls stellt, ist, welche neuen Befehle durch das Modul zur Verfügung stehen. Hier hilft folgender Befehl:

```
Get-Command -Module <Modulname> [, <Modulname2>]
```

- ✓ An die Stelle des Platzhalters schreiben Sie den Namen des Moduls, dessen Befehle Sie kennenlernen wollen. Mehrere Module werden durch Kommata voneinander getrennt.
- ✓ Es spielt dabei keine Rolle, ob das angegebene Modul bereits geladen wurde. Die PowerShell stellt diese Informationen auch dann bereit, wenn das Modul noch nicht geladen wurde.

## Modul automatisch laden



In früheren Versionen der PowerShell musste ein Modul erst geladen werden, bevor die enthaltenen Befehle zur Verfügung standen. Das Verhalten ist seit PowerShell 3.0 komfortabler: Ein Modul wird automatisch geladen, wenn ein Befehl aufgerufen wird, der Bestandteil des Moduls ist. Das ist besonders angenehm, da Sie sich über die Modulzugehörigkeit der Cmdlets keine Gedanken mehr machen müssen. Allerdings empfiehlt es sich nach wie vor, häufiger verwendete Module im Voraus zu laden, z. B. über ein Profilskript. Die mit dem Laden verbundene Wartezeit wird so auf den Start der PowerShell verschoben und verlangsamt nicht die Abarbeitung von Eingaben oder Skripten im laufenden Betrieb.

Zum besseren Verständnis, wie Sie mit Modulen arbeiten können und welche Möglichkeiten Ihnen Module bringen, werden im Folgenden einige wichtige Beispielmodule vorgestellt.

## 9.3 Das Modul *ServerManager*

Das Modul *ServerManager* bietet zusätzliche Befehle zum Automatisieren und Bereitstellen von Rollen und Features auf Windows Server 2016 (oder 2019). Die Befehle sind als Alternative zur Aktion an der grafischen Oberfläche im Server-Manager zu verstehen.

Das Modul stellt folgende Befehle zur Verfügung:

Befehl	Erläuterung
Get-WindowsFeature	Frage ab, welche Rollen, Rollendienste und Features installiert sind
Install-WindowsFeature	Installiert eine oder mehrere Rollen, Rollendienste oder Features
Uninstall-WindowsFeature	Deinstalliert eine oder mehrere Rollen, Rollendienste oder Features
Add-WindowsFeature	Alias für das Cmdlet Install-WindowsFeature
Remove-WindowsFeature	Alias für das Cmdlet Uninstall-WindowsFeature
Enable-ServerManagerStandardUserRemoting	Gibt einem als Wert des Parameters –User spezifizierten Standard-Anwender die Berechtigung, im Server-Manager Ereignisse, Dienste, Leistungsdaten, Rollen und Features einzusehen
Disable-ServerManagerStandardUserRemoting	Entzieht die durch das Cmdlet Enable-ServerManagerStandardUserRemoting erteilte Berechtigung

Die drei grundlegenden Cmdlets Get-WindowsFeature, Install-WindowsFeature und Uninstall-WindowsFeature verwenden identische Parameter:

WindowsFeature-Cmdlets		
Parameter	Typ	Beschreibung
-Name	P (1)	Angabe des gewünschten Rollen- oder Featurenamens bzw. eines Namenteils (Wildcards sind dabei erlaubt)
-ComputerName	N	Angabe des Computernamens für die Abfrage eines Remotesystems
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: Get-Help Get-WindowsFeature -Full Get-Help Install-WindowsFeature -Full Get-Help Uninstall-WindowsFeature -Full

Die Cmdlets `Install-WindowsFeature` und `Uninstall-WindowsFeature` verwenden darüber hinaus noch weitere wichtige Parameter:

Install-WindowsFeature		
Parameter	Typ	Beschreibung
<code>-IncludeAllSubFeature</code>	S	Alle zugeordneten Rollendienste bzw. ungeordnete Features sollen zur Rolle oder zum Feature, wie im Parameter <code>-Name</code> angegeben, mit installiert werden. Dieser Parameter steht nur für das Cmdlet <code>Install-WindowsFeature</code> zur Verfügung. Bei der Deinstallation versteht sich die Entfernung untergeordneter Elemente von selbst.
<code>-IncludeManagementTools</code>	S	Verwaltungskonsolen sollen – sofern vorhanden – für die installierte Rolle, den Rollendienst oder das Feature installiert werden. Dies geschieht ansonsten <b>nicht</b> automatisch bei Verwendung des PowerShell-Cmdlets.
<code>-Restart</code>	S	Der Computer wird neu gestartet, falls ein Neustart durch die Installation erforderlich wird.

### Beispiele

Sie möchten einen Überblick über alle Rollen, Rollendienste und Features erhalten:

```
Get-WindowsFeature
```

Die ausgegebene Liste ist lang und unübersichtlich. Das Beispiel soll angepasst werden, damit nur noch installierte Komponenten aufgelistet werden:

```
Get-WindowsFeature | Where-Object {$_Installed -eq $true}
```

- ✓ Das Cmdlet `Get-WindowsFeature` sammelt die Informationen wie im vorigen Beispiel.
- ✓ Die Objekte werden über die Pipeline an das Cmdlet `Where-Object` weitergeleitet.
- ✓ Das Cmdlet filtert die Objekte und gibt nur die aus, deren Eigenschaft `Installed` den Wert `$true` aufweist. Alternativ können Sie die Eigenschaft `InstallState` auf den Wert `Installed` prüfen, das Ergebnis ist in beiden Fällen identisch.

Sie möchten den Microsoft Webserver IIS (*Internet Information Services*) inklusive aller Rollendienste auf einem Remoterechner mit dem Namen *Server3* installieren:

```
a) Get-WindowsFeature -Name Web-* | Install-WindowsFeature -ComputerName Server3
b) Install-WindowsFeature -Name Web-Server -IncludeAllSubFeature -ComputerName Server3
```

- ✓ Die Lösungen a) und b) zeigen unterschiedliche Herangehensweisen, führen aber zu dem gleichen Ergebnis.
- ✓ Lösung a) arbeitet mit der Pipeline. Zuerst werden die verfügbaren Rollen, Rollendienste und Features abgefragt, die mit der Zeichenfolge `Web-` beginnen. Damit wird die Auswahl auf die Rolle `Webserver (IIS)` und alle Rollendienste beschränkt, da nur der Webserver (IIS) und die dazugehörigen Rollendienste mit dieser Zeichenfolge beginnen. Schließlich wird die Auswahl auf einem Computer mit Namen *Server3* installiert.
- ✓ Lösung b) wählt zur Installation die Rolle `Webserver (IIS)` aus. Durch Parameter werden das Installationsziel und die Installation aller untergeordneten Rollendienste festgelegt.

## 9.4 Das Modul *DnsServer*

Die Verwaltung von DNS-Servern durch die PowerShell ist erst ab PowerShell Version 3.0 mithilfe des neuen Moduls *DnsServer* möglich. Bislang konnten Sie DNS-Server über die grafische Oberfläche oder zum Teil mit dem Befehlszeilentool *dnscmd.exe* verwalten.

Das PowerShell-Modul *DnsServer* stellt 130 Befehle zur Verwaltung von DNS-Servern bereit und integriert damit diesen Themenbereich vollständig in die PowerShell.

- Orientieren Sie sich mit dem folgenden Befehl über die Befehle, die dieses Modul zur Verfügung stellt:

```
Get-Command -Module DnsServer
```

Nachfolgend sehen Sie einige ausgewählte – im DNS-Serverumfeld häufiger verwendete – Befehle des Moduls und ihr Einsatzgebiet:

Befehl	Erläuterung
Add-DnsServerPrimaryZone	Erstellt eine primäre DNS-Zone (Active Directory-integriert oder Standard)
Add-DnsServerResourceRecordA Add-DnsServerResourceRecordAAAA Add-DnsServerResourceRecordCName Add-DnsServerResourceRecordMX	Die ausgewählten Befehle fügen die angegebenen Einträge einer Zone hinzu (A, AAAA, CName, MX).
Clear-DnsServerCache Show-DnsServerCache	Löscht den DNS-Servercache bzw. zeigt ihn an
ConvertTo-DnsServerSecondaryZone	Wandelt eine bestehende Zone in eine sekundäre Zone um
Export-DnsServerZone	Exportiert eine (Active Directory-integrierte) Zone in eine Datei
Get-DnsServer Get-DnsServerDsSetting Get-DnsServerScavenging	Fragt ausführliche Informationen zur allgemeinen Konfiguration eines DNS-Servers, zur Active Directory-Konfiguration und zu den Einstellungen des Aufräumprozesses ab
Get-DnsServerZone	Liefert Informationen zu den Zonen eines DNS-Servers
Invoke-DnsServerZoneSign	Signiert eine DNS-Zone für DNSSEC
Set-DnsServerResourceRecord	Ändert einen Eintrag einer Zone
Test-DnsServer	Prüft, ob ein angegebener Server ein DNS-Server ist

Die Befehle des Moduls sind umfassend. Darüber hinaus steht ein weiteres Modul *DnsClient* zur Verfügung, das zusätzliche Befehle für DNS-Clients bietet und das Thema DNS abrundet.

### Beispiele

Sie möchten Informationen zur Konfiguration Ihres lokalen DNS-Servers erhalten:

```
Get-DnsServer
```

- ✓ Der Befehl liefert ausführliche Informationen zur Konfiguration des DNS-Servers, der auf Ihrem Rechner läuft.
- ✓ Wollen Sie Angaben zu einem Remotesystem abfragen, ergänzen Sie den Parameter *-ComputerName*, den Sie von vielen anderen Cmdlets bereits kennen.

Sie möchten erfahren, welche IPv4-Host-Einträge sich in einer Ihrer DNS-Zonen befinden:

```
Get-DnsServerResourceRecord -ZoneName <Zone> | Where-Object {$_ .RecordType -eq "A" }
```

- ✓ Sie verwenden den Befehl `Get-DnsServerResourceRecord` mit dem Parameter `-ZoneName`, um die gewünschte Zone zu adressieren.
- ✓ Das Cmdlet `Where-Object` filtert die über die Pipeline übergebenen Objekte gemäß der Bedingung, nur den Eintragstyp "A" für IPv4-Hosts anzuzeigen (Eigenschaft `RecordType`).

Sie möchten detaillierte Informationen über die Zonen des Remote-DNS-Servers Server3 erhalten:

```
Get-DnsServerZone -ComputerName Server3 | Format-List
```

- ✓ Durch Angabe des Parameters `-ComputerName` können Sie Informationen über ein Remotesystem abfragen. Wenn Sie diesen Parameter weglassen, erhalten Sie Informationen über Ihren lokalen DNS-Server.
- ✓ `Format-List` dient dazu, alle verfügbaren Eigenschaften in einer Liste anzuzeigen. Ohne dieses Cmdlet erhalten Sie eine vorformatierte Standardtabelle mit wenigen Informationen.

Sie möchten die Active Directory-integrierte Zone `goettingen.ad` in eine Datei `export-goettingen.ad.dns` exportieren:

```
Export-DnsServerZone -Name goettingen.ad -FileName export-goettingen.ad.dns
```

- ✓ Im Parameter `-Name` geben Sie den Namen der Zone an, die exportiert werden soll.
- ✓ Im Parameter `-FileName` legen Sie den Namen der Exportdatei fest.
- ✓ Die Exportdatei wird standardmäßig im Verzeichnis `%windir%\System32\dns` gespeichert.

## 9.5 Das Modul *ServerCore*

Das Modul *ServerCore* liefert zwei zusätzliche Cmdlets für eine Situation, die Sie bislang nur durch einen Eingriff in die Registry lösen konnten. Das Modul wurde entwickelt, um die Bildschirmauflösung eines Rechners im Server Core-Modus auszulesen und zu ändern.

Das Modul liefert folgende Cmdlets:

Cmdlet	Erläuterung
<code>Get-DisplayResolution</code>	Zeigt die aktuelle Bildschirmauflösung des Rechners
<code>Set-DisplayResolution</code>	Ändert die aktuelle Bildschirmeinstellung des Rechners

Das Cmdlet `Get-DisplayResolution` besitzt keine eigenen Parameter. Als Rückgabe erhalten Sie die aktuelle Bildschirmauflösung in der Form `<Breite>x<Höhe>` in Pixeln, z. B. `1024x768`.

Das Cmdlet `Set-DisplayResolution` benötigt Parameter für die Änderungen, die mit seiner Hilfe vorgenommen werden sollen:

Set-DisplayResolution		
Parameter	Typ	Beschreibung
<code>-Width</code>	<code>P (1)</code>	Angabe der gewünschten Breite des Bildschirms in Pixeln
<code>-Height</code>	<code>P (2)</code>	Angabe der gewünschten Höhe des Bildschirms in Pixeln
<code>-Force</code>	<code>S</code>	Führt die Änderung ohne Bestätigungsnachfrage aus
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: <code>Get-Help Set-DisplayResolution -Full</code>

Beide Cmdlets sind zwar für Server Core ohne grafische Oberfläche entwickelt worden, funktionieren aber auch auf Rechnern mit kompletter grafischer Oberfläche.

Achten Sie bei der Änderung der Bildschirmauflösung darauf, dass die gewünschten Werte von Ihrer Hardware unterstützt werden (Bildformate 4:3, 16:9 etc.).



### Beispiel

Sie lesen die Bildschirmauflösung eines Server Core aus ( $800 \times 600$  Pixel) und wollen die Auflösung auf einen Wert von  $1024 \times 768$  Pixeln ändern:

- a) `Get-DisplayResolution`
- b) `Set-DisplayResolution -Width 1024 -Height 768`

- ✓ Bei Eingabe a) erhalten Sie die Rückgabe:  $800x600$ .
- ✓ Bei Eingabe b) erhalten Sie eine Meldung, die Ihre Vorgaben zusammenfasst und um Bestätigung bittet, ob Sie die Anzeigeeinstellungen speichern möchten. Wenn Sie den Befehl ohne Nachfrage ausführen möchten, ergänzen Sie den Befehl um den Parameter `-Force`.

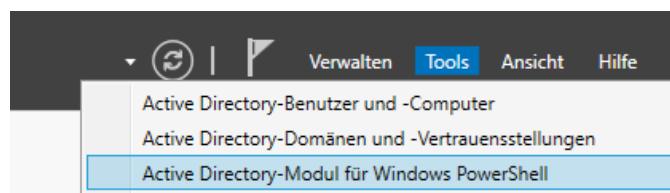
## 9.6 Das Modul *ActiveDirectory*

Die PowerShell bietet seit der Version 2.0 eine direkte Active Directory-Unterstützung durch das Modul *Active-Directory*. Das Modul stellt eine sehr umfassende und weitreichende Erweiterung der PowerShell dar. Es bietet 147 Cmdlets zur Verwaltung Ihrer Active Directory-Domäne.

### Das Active Directory-Modul für Windows PowerShell

Im Server-Manager von Windows Server 2016 (oder 2019) finden Sie eine spezielle PowerShell mit bereits geladenem Modul *ActiveDirectory*.

Es öffnet sich ein normales PowerShell-Konsolenfenster, das nur wenige Unterschiede zur gängigen PowerShell-Konsole bietet:



Server-Manager: Tools, Active Directory-Modul

- ✓ Die Hintergrundfarbe des Konsolenfensters ist schwarz und nicht blau.
- ✓ Das Modul *ActiveDirectory* wird bereits mit dem Start der Anwendung geladen.

Wenn Sie das Modul in der vertrauten Umgebung Ihrer bereits individuell konfigurierten PowerShell-Konsole oder PowerShell ISE explizit laden wollen, geben Sie folgenden Befehl ein:

```
Import-Module -Name ActiveDirectory
```

Alternativ rufen Sie einen Befehl des Moduls ohne vorheriges Laden des Moduls auf. Das Modul wird dann automatisch geladen. Bis zum Ende der aktuellen PowerShell-Sitzung steht Ihnen nun das Modul mit seinen zusätzlichen Befehlen zur Verfügung.

Das Active Directory-Modul für Windows PowerShell ist eine Verknüpfung, die folgenden Befehl ausführt:  
`%windir%\system32\WindowsPowerShell\v1.0\powershell.exe -noexit -command import-module ActiveDirectory`. Es wird also eine neue Instanz der PowerShell geöffnet, die das Modul *ActiveDirectory* lädt (-command import-module ActiveDirectory) und das Fenster nach Ausführung des Befehls geöffnet lässt (-noexit).

### Die Cmdlets des Moduls *ActiveDirectory*

Die PowerShell vereinfacht den Umgang mit Active Directory enorm, besonders wenn man mit der früheren Herangehensweise z. B. über Visual Basic Script und das LDAP-Protokoll vergleicht.

Aufgrund der Fülle der Cmdlets sehen Sie in der folgenden Tabelle nur die passenden Suchbegriffe, um die Cmdlets des Moduls – gefiltert nach einem Verb bzw. einer Aktion – aufzulisten:

Verb des Cmdlets	Suche nach den dazugehörigen Cmdlets	Anzahl
Add	Get-Command -Module ActiveDirectory -Verb Add	7
Get	Get-Command -Module ActiveDirectory -Verb Get	43
New	Get-Command -Module ActiveDirectory -Verb New	20
Remove	Get-Command -Module ActiveDirectory -Verb Remove	26
Set	Get-Command -Module ActiveDirectory -Verb Set	30
Sonstiges	Get-Command -Module ActiveDirectory -Verb Clear, Disable, Enable, Grant, Install, Move, Rename, Reset, Restore, Revoke, Search, Show, Sync, Test, Uninstall, Unlock	21

Für jedes Cmdlet steht die interne Hilfe der PowerShell zur Verfügung, die Ihnen ausführliche Informationen inklusive Beispielen liefert.

Jeder Substantivteil der Befehle dieses Moduls beginnt mit „AD“. Die allgemeine Struktur aller Befehle des Moduls lautet <Verb>-AD<Substantiv>. Auch viele andere Module statthen zugehörige Cmdlets mit einem einheitlichen Präfix im Substantivteil aus.

## 9.7 Das virtuelle Laufwerk AD:

Erweiterungen der PowerShell durch das Nachladen von Modulen können zusätzliche Provider liefern. Das Modul *ActiveDirectory* liefert einen Provider gleichen Namens, der ein virtuelles Laufwerk *AD*: zur Verfügung stellt. Wenn ein Laufwerk angesprochen wird, wird das Modul, das das Laufwerk zur Verfügung stellt, nicht automatisch geladen. Ein Laufwerk kann demnach erst angesprochen werden, wenn das dazugehörige Modul geladen ist.

Bei der Navigation in diesem Laufwerk offenbart sich eine Hürde: Sie müssen in diesem Laufwerk definierte Namen (*distinguished names, DN*) verwenden.

- Wechseln Sie mit dem folgenden Befehl auf das Laufwerk AD:

```
Set-Location -Path AD:
```

- Lassen Sie sich den Inhalt des Laufwerks auflisten:

```
Get-ChildItem
```

Name	ObjectClass	DistinguishedName
herdt	domainDNS	DC=herdt,DC=ps
Configuration	configuration	CN=Configuration,DC=herdt,DC=ps
Schema	dMD	CN=Schema,CN=Configuration,DC=herdt,DC=ps
DomainDnsZones	domainDNS	DC=DomainDnsZones,DC=herdt,DC=ps
ForestDnsZones	domainDNS	DC=ForestDnsZones,DC=herdt,DC=ps

*PowerShell-Laufwerk AD: des Providers Active Directory*

Der Domänenname der Testumgebung lautet *herdt.ps*. In der rechten Spalte der Ausgabe finden Sie den definierten Namen, den Sie für die Navigation im Laufwerk *AD*: verwenden müssen.

- Geben Sie bei der Navigation im Befehl den gewünschten Pfad in Anführungszeichen in Form eines definierten Namens an, z. B.:

```
Set-Location -Path "dc=herdt,dc=ps"
```

Eine konventionelle Pfadangabe führt auf diesem Laufwerk zu einer Fehlermeldung.

Die PowerShell-Konsole unterstützt bei der Eingabe auch dieser Pfade mit der Tabulator-Vervollständigung die PowerShell ISE durch die IntelliSense-Menüs, wie Sie in der nebenstehenden Abbildung erkennen können.

Alle weiteren Arbeiten im Active Directory nehmen Sie mithilfe der Cmdlets vor, die das Modul *ActiveDirectory* zur Verfügung stellt. Eine genaue Beschreibung der Arbeit mit diesen Cmdlets finden Sie im folgenden Kapitel.

PS C:\Users\Administrator> Import-Module -Name ActiveDirectory  
PS C:\Users\Administrator> Set-Location -Path AD:  
PS AD:> Set-Location -Path

- CN=Configuration,DC=herdt,DC=ps
- CN=Schema,CN=Configuration,DC=he...
- DC=DomainDnsZones,DC=herdt,DC=ps
- DC=ForestDnsZones,DC=herdt,DC=ps
- DC=herdt,DC=ps

*IntelliSense in PowerShell ISE: Hilfe bei definierten Namen*

Eine genaue Kenntnis definierter Namen ist für Befehlszeilentools unabdingbar. Eine Orientierung für den Umgang mit definierten Namen bietet der ADSI-Editor, den Sie im Menü *Tools* des Server-Managers finden. Rufen Sie den ADSI-Editor auf und stellen Sie eine Verbindung mit dem standardmäßigen Namenskontext her. Wie in der nachfolgenden Abbildung zu sehen ist, können Sie jede Organisationseinheit, jeden Container und jedes Objekt der Domäne finden und in der Spalte *Definierter Name* den dazugehörigen definierten Namen ablesen.



Name	Klasse	Definierter Name
CN=BuiltIn	builtinDomain	CN=BuiltIn,DC=herdt,DC=ps
CN=Computers	container	CN=Computers,DC=herdt,DC=ps
OU=Domain Controllers	organization	OU=Domain Controllers,DC=herdt,DC=ps
CN=ForeignSecurityPrincip...	container	CN=ForeignSecurityPrincipals,DC=herdt,DC=ps
CN=Keys	container	CN=Keys,DC=herdt,DC=ps
CN=LostAndFound	lostAndFound	CN=LostAndFound,DC=herdt,DC=ps
CN=Managed Service Acco...	container	CN=Managed Service Accounts,DC=herdt,DC=ps
CN=NTDS Quotas	msDS-Quota	CN=NTDS Quotas,DC=herdt,DC=ps
CN=Program Data	container	CN=Program Data,DC=herdt,DC=ps
CN=System	container	CN=System,DC=herdt,DC=ps
CN=TPM Devices	msTPM-Info	CN=TPM Devices,DC=herdt,DC=ps
CN=Users	container	CN=Users,DC=herdt,DC=ps
CN=Infrastructure	infrastructur...	CN=Infrastructure,DC=herdt,DC=ps

Der ADSI-Editor zeigt definierte Namen im Active Directory.

In einem definierten Namen wird z. B. nicht unterschieden, ob Sie vor oder nach einem Komma oder einem Gleichheitszeichen ein Leerzeichen verwenden oder nicht. So ist "OU = Domain Controllers, DC = herdt, DC = ps" gleichbedeutend mit "OU=Domain Controllers,DC=herdt,DC=ps". Wenn Sie in dieser Unterlage Beispiele mit und ohne Leerzeichen finden, hat dies ausschließlich layouttechnische Gründe.

## 9.8 Lokale Benutzer und Gruppen verwalten

Mit der PowerShell Version 5 wurde eine weitere Lücke in der Funktionalität der PowerShell geschlossen. Mit einem Modul namens *Microsoft.PowerShell.LocalAccounts* werden eine Reihe von Cmdlets zur Verwaltung lokaler Benutzer, Gruppen und Gruppenmitgliedschaften zur Verfügung gestellt.

Befehl	Erläuterung
Add-LocalGroupMember	Fügt Mitglieder zu einer lokalen Gruppe hinzu
Disable-LocalUser	Deaktiviert ein lokales Benutzerkonto
Enable-LocalUser	Aktiviert ein lokales Benutzerkonto
Get-LocalGroup	Liest lokale Gruppen aus
Get-LocalGroupMember	Liest die Mitglieder lokaler Gruppen aus
Get-LocalUser	Liest die lokalen Benutzerkonten aus
New-LocalGroup	Erstellt eine neue lokale Gruppe
New-LocalUser	Erstellt ein neues lokales Benutzerkonto
Remove-LocalGroup	Löscht eine lokale Gruppe
Remove-LocalGroupMember	Entfernt ein Mitglied aus einer lokalen Gruppe
Remove-LocalUser	Löscht ein lokales Benutzerkonto
Rename-LocalGroup	Benennt eine lokale Gruppe um
Rename-LocalUser	Benennt ein lokales Benutzerkonto um
Set-LocalGroup	Nimmt Änderungen an einer lokalen Gruppe vor
Set-LocalUser	Nimmt Änderungen an einem lokalen Benutzerkonto vor

Allen Cmdlet ist gemeinsam, dass sie über die Parameter **-Name** und **-SID** verfügen. Durch die Angabe des Namens oder der SID bezeichnen Sie das anzusprechende Objekt eindeutig.

Die Cmdlets **Add-LocalGroupMember** und **Remove-LocalGroupMember** erfordern selbstverständlich sowohl die Angabe der Gruppe (Parameter **-Group**) als auch des neuen bzw. alten Gruppenmitglieds (Parameter **-Member**), auf das die Aktion angewendet werden soll.

Die Cmdlets **Rename-LocalUser** und **Rename-LocalGroup** erwarten neben dem Namen des bisherigen Objekts (Parameter **-Name**) auch den neuen Namen (Parameter **-NewName**).

Die Cmdlets **New-LocalGroup** und **New-LocalUser** sowie **Set-LocalGroup** und **Set-LocalUser** bieten weitere Parameter, die aussagekräftig benannt wurden und sich daher selbst erklären.

An dieser Stelle werden anhand von Beispielen für das Cmdlet **New-LocalUser** die häufigsten Parameter vorgestellt:

### Beispiel 1

Sie wollen mithilfe der PowerShell einen neuen lokalen Benutzer erstellen:

```
New-LocalUser -Name "UserX" -Description "Neues Benutzerkonto für UserX"  
-NoPassword
```

- ✓ Das neue Benutzerkonto *UserX* wird angelegt und mit einer Beschreibung versehen. Es wurde kein Kennwort vergeben. Das Konto ist aktiv, der Benutzer wird bei der ersten Anmeldung aufgefordert, sein Kennwort zu ändern.

### Beispiel 2

Sie legen das Benutzerkonto *UserY* an, vergeben ein Kennwort und legen weitere Einstellungen fest:

```
$Kennwort = Read-Host -AsSecureString  
New-LocalUser "UserY" `  
-Password $Kennwort `  
-FullName "Benutzer Y" `  
-Description "Weiterer Benutzer" `  
-AccountNeverExpires `  
-UserMayNotChangePassword
```

- ✓ In der ersten Zeile wird der Anwender zur Eingabe eines Kennworts aufgefordert, das in der Variable `$Kennwort` gespeichert wird.
- ✓ Das Benutzerkonto wird angelegt. Informationen wie Kennwort (Parameter **-Password**), vollständiger Name (Parameter **-FullName**) und Beschreibung (Parameter **-Description**) werden mit dazugehörigen Werten zur genaueren Definition des Benutzerkontos verwendet.
- ✓ Letztlich werden die Parameter **-UserMayNotChangePassword** (Benutzer kann Kennwort nicht ändern) und **-AccountNeverExpires** (Konto läuft nie ab) ohne Wert verwendet.

## 9.9 Kurz zusammengefasst

### Erweiterungen für jedes Aufgabengebiet

Module sind als Befehlserweiterungen der PowerShell zu verstehen, die nachgeladen werden können, um themenbezogen weitere Befehle zur Verfügung zu stellen. Seit der PowerShell 3.0 ist es möglich, Befehle aus Modulen zu verwenden, die noch nicht geladen wurden. Das Laden des Moduls, das den aufgerufenen Befehl beherbergt, erfolgt in diesen Fällen automatisch im Hintergrund.

Die Zahl der Module ist in PowerShell-Version 4.0 sprunghaft angestiegen, bei der neuesten Version werden vornehmlich weitere inhaltliche Lücken geschlossen. Microsoft setzt das Vorhaben bereits fast gänzlich um, das komplette System bzw. Netzwerk mit der PowerShell verwalten zu können. Wichtige Bausteine zum Erreichen dieses Vorhabens sind die Module, die für bestimmte Aufgabengebiete Befehle zur Verfügung stellen. Das bedeutet auch, dass die Installation weiterer Komponenten weitere Module zur Erledigung dieser speziellen Aufgaben liefern kann.

Die Arbeit mit Modulen unterscheidet sich nicht von der Arbeit mit den vorgestellten Cmdlets früherer Kapitel. Hilfreich für den Anwender ist eher, dass die Befehle der einzelnen Module häufig über den Namen als Bestandteil eines speziellen Moduls identifiziert werden können (wie z. B. <Verb>-DnsServer<...> als Bestandteil des Moduls *DnsServer* oder <Verb>-AD<...> als Bestandteil des Moduls *ActiveDirectory*).



Wissenstest: *PowerShell-Grundlagen-2*

## 9.10 Übung

### Mit Modulen arbeiten

Übungsdatei: –

Ergebnisdatei: *9\_ergebnisse.txt*

1. Lassen Sie sich anzeigen, welche Module in Ihrer aktuellen PowerShell-Sitzung geladen sind.
2. Können Sie sich die Befehle des Moduls *DnsServer* anzeigen lassen, ohne dass das Modul geladen ist? Probieren Sie es aus.
3. Warum gibt es unterschiedliche Module auf einem Domänencontroller (a), auf einem Mitgliedserver (b) und einem Clientcomputer mit Windows 10 (c)?
4. Wie finden Sie heraus, zu welchem Modul ein Cmdlet gehört?
5. Welche Befehle stehen im Modul *DnsClient* zur Verfügung? Informieren Sie sich über die enthaltenen Befehle.
6. Entfernen Sie das Modul *DnsClient* wieder aus der aktuellen PowerShell-Sitzung.
7. Finden Sie die Bildschirmauflösung Ihres Rechners heraus und ändern Sie sie in einen anderen gültigen Wert.
8. a) Wie lautet der Befehl, um das Modul *ActiveDirectory* auf Ihrem Domänencontroller zu laden?  
b) Muss das Modul manuell geladen werden, bevor ein Befehl des Moduls ausgeführt werden kann?
9. Wie können Sie auf dem virtuellen Laufwerk *AD*: ein Verzeichnis wechseln, z. B. vom Root-Verzeichnis in das Verzeichnis *herdt.ps*?

# 10 Active Directory verwalten



Beispieldatei: 10\_kompletter code.txt (Ordner Kapitel 10)

## 10.1 Cmdlets für die Verwaltung von Active Directory-Basisobjekten

Mit den 147 Cmdlets des Moduls `ActiveDirectory` steht eine mächtige Befehlssammlung zur Verfügung, die eine umfassende Verwaltung des Active Directory ermöglicht. Alle in diesem Kapitel behandelten Cmdlets sind Bestandteil des Moduls `ActiveDirectory`. Aufgrund der Fülle der Anwendungsmöglichkeiten werden nicht alle Cmdlets vorgestellt, sondern exemplarisch zu speziellen Aufgabenbereichen der Administration.

Zur Verwaltung der Basisobjekte einer Domäne – Benutzer, Gruppen, Organisationseinheiten (OUs) und Computer – stehen mehrere Cmdlet-Gruppen zur Verfügung:

- ✓ Verwaltung von Benutzern: Cmdlets mit dem Substantiv `ADUser`,
- ✓ Verwaltung von Gruppen: Cmdlets mit dem Substantiv `ADGroup`,
- ✓ Verwaltung von Organisationseinheiten: Cmdlets mit dem Substantiv `ADOrganizationalUnit`,
- ✓ Verwaltung von Computern: Cmdlets mit dem Substantiv `ADComputer`.

Jede dieser Gruppen besteht aus vier Cmdlets mit den gleichen Verben:

- ✓ Get: Auflesen von Informationen eines Objekts,
- ✓ New: Erstellen eines neuen Objekts,
- ✓ Remove: Löschen eines Objekts,
- ✓ Set: Ändern von Eigenschaften eines Objekts.

Die einzelnen Cmdlets lassen sich mit den folgenden Befehlen anzeigen:

- ✓ `Get-Command -Noun ADUser`,
- ✓ `Get-Command -Noun ADGroup`,
- ✓ `Get-Command -Noun ADOrganizationalUnit`,
- ✓ `Get-Command -Noun ADComputer`.

Ausführliche Hilfe zu den Cmdlets erhalten Sie über das Cmdlet `Get-Help`.

## 10.2 Informationen auslesen: Benutzer, Gruppen, OUs und Computer

Zur Informationsbeschaffung verwenden Sie Cmdlets mit dem Verb `Get`. Alle vier Get-Cmdlets funktionieren nach ähnlichem Muster:

- ✓ Sie verwenden den Parameter `-Filter` und formulieren als Wert in geschweiften Klammern eine Bedingung, nach der gefiltert werden soll. Der Parameter arbeitet nach folgender Struktur:

```
-Filter {Eigenschaft -<Vergleichsoperator> Wert}
```

In geschweiften Klammern können auch mehrere Bedingungen miteinander verknüpft werden. In dem Fall sind sie durch Operatoren wie `-and`, `-or` o. Ä. zu verknüpfen. Nur wenn alle Objekte durchgelassen werden sollen, verzichten Sie auf geschweifte Klammern und verwenden den Stern als Platzhalter:

`-Filter *`.

- ✓ Sie verwenden anstelle des Parameters `-Filter` den Parameter `-Identity`, um ein spezielles Objekt anzusprechen. Dies erreichen Sie durch Eingabe eines definierten Namens in Anführungszeichen als Wert des Parameters. Zusätzlich funktionieren als verkürzte Angaben der `SamAccountName` für Benutzer-Objekte, der `Computername` für Computer-Objekte und der Gruppenname für Gruppen-Objekte.

### Beispiele

Sie möchten Informationen zu allen Benutzern, allen Gruppen, allen Organisationseinheiten und allen Computern Ihrer Active Directory-Umgebung abrufen:

```
Get-ADUser -Filter *
Get-ADGroup -Filter *
Get-ADOrganizationalUnit -Filter *
Get-ADComputer -Filter *
```

Sie möchten Informationen zu einer Organisationseinheit *Hamburg* abrufen, die sich in der Organisationseinheit *Benutzer* Ihrer Domäne befindet:

- `Get-ADOrganizationalUnit -Identity "OU=Hamburg,OU=Benutzer,DC=herdt,DC=ps"`
- `Get-ADOrganizationalUnit -Filter {Name -eq "Hamburg"}`

Sie möchten Informationen zu Benutzer *Hans Dampf* abrufen. Der Benutzer befindet sich in der Organisationseinheit *Hamburg* (siehe letztes Beispiel), sein SamAccountName lautet *hans.dampf*: Es existieren verschiedene Wege, die alle das identische Ergebnis liefern:

- `Get-ADUser -Filter {cn -eq "Hans Dampf"}`
- `Get-ADUser -Identity hans.dampf`
- `Get-ADUser -Identity " CN=Hans Dampf,OU=Hamburg,OU=Benutzer,DC=herdt,DC=ps"`

Sie möchten Informationen zu einem Server mit dem Namen *Core03* abrufen (mehrere Möglichkeiten):

- `Get-ADComputer -Identity Core03`
- `Get-ADComputer -Identity "CN=Core03,CN=Computers,DC=herdt,DC=ps"`
- `Get-ADComputer -Filter {Name -eq "Core03"}`

Sie möchten Informationen zu einer Gruppe *HH-Verwaltung* abrufen:

- `Get-ADGroup -Filter {Name -eq "HH-Verwaltung"}`
- `Get-ADGroup -Identity HH-Verwaltung`
- `Get-ADGroup -Identity "CN=HH-Verwaltung,OU=FirmaX,DC=herdt,DC=ps"`

Informationen lassen sich fast immer über verschiedene Methoden abrufen. Verwenden Sie die Methode, die für Sie am nachvollziehbarsten ist. Besonders große Flexibilität bietet der Parameter `-Filter`, da Sie dort mit jeder Eigenschaft eines Objekts arbeiten können. Eine Abfrage nach einem gesuchten Namen ist ebenso realisierbar wie die Filterung nach Hamburger Mitarbeitern, die in einer bestimmten Filiale arbeiten und deren Telefonnummer auf eine bestimmte Ziffer endet.

## 10.3 Benutzer, Gruppen, OUs oder Computer erstellen

Die PowerShell bietet New-Cmdlets, mit denen Sie die Standardobjekte in Ihrer Domäne erstellen können. Wenn Sie sich die Flut möglicher Parameter ansehen, wird schnell klar, dass diese Cmdlets in ausführlicher Form nur im Skripting verwendet werden. Bei interaktiver Verwendung werden Sie nur wenige Parameter einsetzen und damit Objekte mit einigen wenigen definierten Eigenschaften erstellen.

So können Sie sich einen Überblick über die Syntax der Cmdlets verschaffen:

Cmdlet	Eingabe zur Anzeige der Syntax
New-ADComputer	Get-Command -Name New-ADComputer -Syntax
New-ADGroup	Get-Command -Name New-ADGroup -Syntax
New-ADOrganizationalUnit	Get-Command -Name New-ADOrganizationalUnit -Syntax
New-ADUser	Get-Command -Name New-ADUser -Syntax

Die Cmdlets bieten eigene Parameter für alle Eigenschaften, die Sie in den grafisch orientierten Tools auf vielen verschiedenen Registerkarten eintragen können.

### Benutzerobjekte

Wenn Sie die Cmdlets interaktiv verwenden, stellt sich die Frage nach der zulässigen Minimalsyntax, die Sie von Fall zu Fall um einige Parameter erweitern können. Die Erstellung eines neuen Benutzers mit all seinen Eigenschaften direkt in der Konsole wird bei vielen Hundert Zeichen kaum vorgenommen.

Für neue Benutzer-Objekte reicht es bereits aus, den Namen anzugeben. Andere Eigenschaften sind (noch) ohne zugewiesene Werte. Der neue Benutzer wird im Standardspeicherort für neu erstellte Benutzerobjekte erstellt. Darüber hinaus ist der Benutzer deaktiviert, da ihm kein Kennwort zugewiesen wurde.

### Benutzerobjekte: Beispiele

Minimaleingabe in der Konsole – Erstellung eines Benutzers mit Angabe eines Namens:

- a) New-ADUser -Name max.mustermann
- b) New-ADUser -Name "Doug Adams"

Der Benutzer erhält den eingegebenen Namen sowohl als Wert der Eigenschaft Name als auch als Wert der Eigenschaft SamAccountName.

Durch eine etwas erweiterte Eingabe geben Sie zumindest ein Grundgerüst an Informationen vor:

```
New-ADUser -Name "Doug Adams" -SamAccountName "doug.adams" -Path  
"OU=Benutzer,DC=herdt,DC=ps"
```

Ein gutes Ergebnis, bei dem der neue Benutzer gleichzeitig aktiviert wird, erfordert etwas mehr Aufwand, wie z. B. (Zeilenumbruch vor jedem Parameter nur der Übersicht halber):

```
New-ADUser  
-Path "OU=Meine Benutzer,DC=herdt,DC=ps"  
-Name "Doug Adams"  
-Surname "Adams"  
-GivenName "Doug"  
-SamAccountName "doug.adams"  
-UserPrincipalName "doug.adams@herdt.ps"
```

```
-Description "IT-Beauftragter Zweigstelle II"
-Company "Herdt-Verlag"
-Department "IT"
-Office "Bodenheim"
-AccountPassword (ConvertTo-SecureString -AsPlainText "Pa$$w0rd23456"
-Force)
-ChangePasswordAtLogon:$True
-Enabled:$True
```

- ✓ Die einzelnen Parameter sind selbsterklärend. Sie sollen zeigen, wie Sie die Eigenschaften eines neuen Benutzers mit Werten füllen können.
- ✓ Entscheidend sind die letzten drei Zeilen des Beispiels:
  - ✓ In der drittletzten Zeile wird ein Kennwort definiert. Da das Kennwort im Klartext vorliegt, wird es in eine sichere Zeichenfolge konvertiert. Um eine Verwendung eines Klartextkennworts zu ermöglichen, muss zudem der Parameter `-Force` angegeben werden.
  - ✓ In der vorletzten Zeile wird definiert, dass es sich bei dem zugewiesenen Kennwort um ein temporäres Kennwort handelt, das der Benutzer bei der nächsten Anmeldung ändern muss.
  - ✓ In der letzten Zeile erfolgt die Festlegung, dass das Benutzerkonto aktiviert werden soll. Dies funktioniert nur, wenn ein Kennwort definiert wurde.

Eine solche direkte Eingabe ist zu lang und fehlerträchtig und damit kaum praktikabel. Ein Beispielskript ist für diese umfangreiche Aufgabe ein möglicher Lösungsansatz. Im letzten Kapitel des Buchs finden Sie eine Fallstudie, in der auch das automatisierte Erstellen neuer Benutzer in Zusammenarbeit mit einer CSV-Datei gezeigt wird.

## Gruppenobjekte

Um Gruppen mit der PowerShell anzulegen, müssen Sie dieselben Fragen beantworten wie bei der Arbeit mit grafischen Tools:

- ✓ Wie soll die Gruppe heißen (PowerShell-Parameter `-Name`)?
- ✓ In welchem Container soll die Gruppe angelegt werden (PowerShell-Parameter `-Path`)?
- ✓ Welcher Gruppentyp soll erstellt werden (PowerShell-Parameter `-GroupCategory`)?
- ✓ Welcher Gruppenbereich wird gewählt (PowerShell-Parameter `-GroupScope`)?

Bei der Erstellung einer Gruppe müssen mindestens der Gruppenname und der Gruppenbereich angegeben werden. In diesem Fall wird die Gruppe im Standardcontainer für Benutzer als Sicherheitsgruppe erstellt.

## Gruppenobjekte: Beispiele

Sie legen eine globale Gruppe *GG\_HH-Mitarbeiter* mit der minimalen Anzahl an Parametern an:

```
New-ADGroup -Name "GG_HH-Mitarbeiter" -GroupScope Global
```

Sie legen eine Gruppe *LG\_HH-Einkauf* in der Organisationseinheit *Hamburg* als domänenlokale Sicherheitsgruppe an:

```
New-ADGroup -Name "LG_HH-Einkauf" -GroupCategory Security -GroupScope DomainLocal -Path "OU=Hamburg,DC=herdt,DC=ps"
```

## Organisationseinheiten

Zum Erstellen neuer Organisationseinheiten müssen Sie mindestens den Namen der Organisationseinheit (Parameter `-Name`) angeben. Das Objekt wird direkt im Stamm der aktuellen Domäne angelegt und ist automatisch vor versehentlichem Löschen geschützt.

Um den Speicherort und den Schutz vor versehentlichem Löschen selbst zu beeinflussen, verwenden Sie die Parameter `-Path` und `-ProtectedFromAccidentalDeletion`. Wollen Sie zusätzlich Eintragungen zur Beschreibung der Organisationseinheit vornehmen, verwenden Sie die Parameter `-Description` (Beschreibung), `-StreetAddress` (Straße), `-City` (Ort), `-State` (Bundesland/Kanton), `-PostalCode` (PLZ) sowie `-Country` (Land/Region) und belegen Sie mit den gewünschten Werten.

### Organisationseinheiten: Beispiele

Sie erstellen eine neue Organisationseinheit *HH-Mitarbeiter* nur unter Angabe der minimal erforderlichen Daten. Das Objekt wird automatisch im Stamm der aktuellen Domäne erstellt und ist vor versehentlichem Löschen geschützt:

```
New-ADOrganizationalUnit -Name "HH-Mitarbeiter"
```

Sie erstellen eine Organisationseinheit *Meine Mitarbeiter* in der Organisationseinheit *Meine Firma* in Ihrer Domäne. Die Organisationseinheit soll nicht vor versehentlichem Löschen geschützt werden:

```
New-ADOrganizationalUnit -Name "Meine Mitarbeiter" -Path "OU=Meine Firma, DC=herdt,DC=ps" -ProtectedFromAccidentalDeletion:$False
```

Sie erstellen eine neue Organisationseinheit *Bamberg* in der Organisationseinheit *Meine Firma* und nehmen direkt eine Beschreibung der neuen Organisationseinheit vor, die im Active Directory gespeichert wird:

```
New-ADOrganizationalUnit -Name "Bamberg" -Path "OU=Meine Firma, DC=herdt,DC=ps" -City "Bamberg" -Country "DE" -PostalCode 96047 -State "Bayern" -Description "Firmenressourcen in der Bamberger Filiale"
```

Zu beachten ist, dass der Wert des Parameters `-Country` festgelegten Länderkürzeln entsprechen muss.

Den Erfolg der Befehle können Sie mit den vorgestellten Get-Cmdlets oder mit dem grafisch orientierten Verwaltungstool Active Directory-Benutzer und -Computer überprüfen.

## Computerobjekte

Um ein Computerobjekt mit der PowerShell anzulegen, muss zumindest der Parameter `-Name` mit dem Namen des neuen Computers als Wert angegeben werden. In dem Fall wird der neue Computer im Standardspeicherort für Computerobjekte angelegt. Um den Speicherort manuell vorzugeben, wird der Parameter `-Path` verwendet.

### Computerobjekte: Beispiele

Sie legen den Computer *Client75* mit Standardeinstellungen im Active Directory an:

```
New-ADComputer -Name "Client75"
```

Sie legen den Computer *Server23* in der Organisationseinheit *Hamburg* an:

```
New-ADComputer -Name "Server23" -Path "OU=Hamburg,DC=herdt,DC=ps"
```

## 10.4 Eigenschaften von Benutzern, Gruppen, OUs oder Computern ändern

Die Set-Cmdlets zum Ändern der Werte von Eigenschaften von Objekten ähneln stark den New-Cmdlets zum Erstellen neuer Objekte. Im Prinzip verwenden die Cmdlet-Paare dieselben Parameter. Der größte Unterschied besteht darin, dass die Objekte bei der Verwendung von Set-Cmdlets bereits bestehen, während sie durch die New-Cmdlets erst angelegt werden.

Wie bereits in vorhergehenden Kapiteln erwähnt, verwenden Sie zur Orientierung am besten den Attribut-Editor, den Sie auf einer Registerkarte in den Eigenschaften aller Objekte im Verwaltungstool *Active Directory-Benutzer und -Computer* finden. (Damit diese Registerkarte angezeigt wird, aktivieren Sie im Menü *Ansicht* den Eintrag *Erweiterte Features*.) Die von dem Cmdlets angebotenen Parameter entsprechen den auf der Registerkarte gezeigten Attributnamen.

Um Werte von Eigenschaften zu ändern, wählen Sie aus den zwei üblichen Vorgehensweisen: Im ersten Fall setzen Sie ein Set-Cmdlet ein und geben direkt das zu ändernde Objekt an. Im anderen Fall arbeiten Sie mit der Pipeline und suchen im ersten Schritt das gewünschte Objekt, das Sie im zweiten Schritt ändern.

Beide Vorgehensweisen sehen Sie in den folgenden Beispielen.

### Beispiele

Sie ändern den Standort eines Computers und wollen diese Information in den Eigenschaften des Computerobjekts im Active Directory verzeichnen:

```
Set-ADComputer -Identity "CN=Client1,CN=Computers,DC=herdt,DC=ps"
-Location "Stratford-upon-Avon"
```

Sie wollen für alle Gruppen, die sich in der Organisationseinheit *Hamburg* (oder darunter) befinden, den Benutzer *Hans Dampf* zum Verwalter der Gruppen bestimmen:

```
Get-ADGroup -Filter * -SearchBase "OU=Hamburg,DC=herdt,DC=ps" | Set-ADGroup
-ManagedBy "CN=Hans Dampf,CN=Users,DC=herdt,DC=ps"
```

Sie wollen die Beschreibung der Organisationseinheit *Hamburg* ändern:

```
Set-ADOrganizationalUnit -Identity "OU=Hamburg,DC=herdt,DC=ps" -Description
"In dieser Organisationseinheit sind alle Benutzer und Gruppen des
Standorts Hamburg."
```

Sie wollen erreichen, dass sich der Benutzer *Hans Dampf* nur von den Arbeitsstationen *Client1* und *Client2* im Netzwerk anmelden kann. Wenn Sie mehrere Arbeitsstationen eintragen, beachten Sie, dass in der Aufzählung keine Leerzeichen verwendet werden:

```
Set-ADUser -Identity "CN=Hans Dampf,CN=Users,DC=herdt,DC=ps"
-LogonWorkstations "Client1","Client2"
```

In einem weiterführenden Beispiel sehen Sie, welche Möglichkeiten sich durch die PowerShell eröffnen:

Wenn Sie die Werte mehrerer Eigenschaften eines Objekts ändern, entfernen, löschen und hinzufügen wollen, können Sie das ebenfalls mit einem einzigen Cmdlet erreichen. In dem Fall verwenden Sie Hashtabellen und die Parameter *-Remove*, *-Add*, *-Replace* und/oder *-Clear*. Nachfolgend sehen Sie ein Beispiel zu entsprechenden Änderungen an einem Benutzer im Active Directory:

```
Set-ADUser -Identity "CN=Hans Dampf,CN=Users,DC=herdt,DC=ps" -Add
@{otherTelephone=987654321;url="herdt.ps"} -Clear Description -Replace
@{Title="Chef";telephoneNumber=12345678}
```

## Sonderfall: Gruppenmitgliedschaften verwalten

Gruppen haben einen Sonderstatus, da andere Objekte Mitglieder einer Gruppe sein können und Gruppen wiederum Mitglieder anderer Gruppen. Für diesen Zweck bietet die PowerShell spezielle Cmdlets, die sich um die Verwaltung von Gruppenmitgliedschaften kümmern:

Cmdlet	Erläuterung
Add-ADGroupMember	Fügt ein Mitglied (oder mehrere) einer Active Directory-Gruppe hinzu.
Get-ADGroupMember	Zeigt Informationen zu Mitgliedschaften in einer angegebenen Gruppe an.
Remove-ADGroupMember	Entfernt ein Mitglied (oder mehrere) aus einer Active Directory-Gruppe.

### Gruppenmitglieder hinzufügen

Um Mitglieder zu einer Gruppe hinzuzufügen, verwenden Sie den folgenden Befehl:

```
Add-ADGroupMember -Identity <DN Gruppe> -Members <DN Mitglied1>[, <DN Mitglied2>]
```

- ✓ Damit der Befehl allgemein verwendbar ist, wird die Verwendung von definierten Namen empfohlen. Dann können Sie verschiedene Objekttypen adressieren, sofern diese als Mitglied der angegebenen Gruppe zulässig sind.
- ✓ In verkürzter Schreibweise können Sie den Teil des Befehls `-Identity <DN Gruppe>` durch den Namen der Gruppe ersetzen. Das gilt ebenfalls für Werte des Parameters `-Members`, sofern es sich um Gruppen oder Benutzer handelt. Bei diesen Objekttypen müssen Sie den Gruppennamen bzw. den SamAccountName des Benutzers angeben.

### Beispiele

Um z. B. das Computerkonto *Client1*, den Benutzer *Hans Dampf* und die globale Gruppe *HH-Mitarbeiter* der domänenlokalen Gruppe *LG\_Drucker3\_Drucken* hinzuzufügen, verwenden Sie den folgenden Befehl:

```
Add-ADGroupMember -Identity
"CN=LG_Drucker3_Drucken,OU=Hamburg,DC=herdt,DC=ps"
-Members "CN=Client1,CN=Computers,DC=herdt,DC=ps", "CN=Hans
Dampf,CN=Users,DC=herdt,DC=ps", "CN=HH-
Mitarbeiter,OU=Hamburg,DC=herdt,DC=ps"
```

In verkürzter Schreibweise ist der Befehl übersichtlicher, falls Sie sich sicher sind, bei welchen Objekttypen Sie auf die Angabe des definierten Namens verzichten können:

```
Add-ADGroupMember LG_Drucker3_Drucken -Members hans.dampf, HH-Mitarbeiter,
"CN=Client1,CN=Computers,DC=herdt,DC=ps"
```

### Gruppenmitglieder anzeigen

Zur Anzeige vorhandener Mitgliedschaften einer Gruppe verwenden Sie das Cmdlet `Get-ADGroupMember` unter Angabe des Parameters `-Identity`. Als Wert des Parameters geben Sie den definierten Namen der Gruppe an, deren Mitglieder Sie anzeigen wollen. Die bekannte verkürzte Schreibweise ist auch hier erlaubt.

Zusätzlich können Sie den Switch-Parameter `-Recursive` verwenden. Bei Angabe dieses Parameters werden keine Gruppen als Mitglieder angezeigt, sondern alle Mitgliedschaften von Einzelobjekten, auch wenn sie mittelbar sind.

Es ist z. B. *Hans* Mitglied von *Gruppe 1*, und *Gruppe 1* und *Harry* sind Mitglieder von *Gruppe 2*. Wenn Sie die Gruppenmitgliedschaften von *Gruppe 2* ermitteln, erhalten Sie ohne den Parameter *-Recursive* als Ergebnis *Hans* und *Gruppe 1*. Geben Sie den Parameter *-Recursive* an, lautet das Ergebnis: *Hans* und *Harry*.

### Beispiele

Sie wollen die Mitglieder der Gruppe *LG\_Drucker3\_Drucken* ermitteln:

- a) Get-ADGroupMember -Identity  
"CN=LG\_Drucker3\_Drucken,OU=Hamburg,DC=herdt,DC=ps"
- b) Get-ADGroupMember LG\_Drucker3\_Drucken

Sie wollen alle Benutzer (und Computer) ermitteln, die mittelbar oder unmittelbar Mitglied der Gruppe *LG\_Drucker3\_Drucken* sind:

```
Get-ADGroupMember -Identity  
"CN=LG_Drucker3_Drucken,OU=Hamburg,DC=herdt,DC=ps" -Recursive
```

Sie wollen alle Mitglieder der Gruppe *LG\_Drucker3\_Drucken* ermitteln, aber nur eine kurze Anzeige der Namen der Mitglieder anzeigen:

```
Get-ADGroupMember -Identity  
"CN=LG_Drucker3_Drucken,OU=Hamburg,DC=herdt,DC=ps" | Select-Object  
-Property Name
```

### Gruppenmitglieder entfernen

Zur Entfernung von Gruppenmitgliedern stellt die PowerShell das Cmdlet *Remove-ADGroupMember* zur Verfügung. Als Parameter sind *-Identity* und *-Members* zu verwenden, um die Gruppe und die zu entfernenden Mitglieder anzugeben. Die allgemeinen Parameter *-WhatIf* zur Simulation der Tätigkeit und *-Confirm* zum Ausblenden der Bestätigungsfrage werden in der Praxis an dieser Stelle häufiger verwendet.

Wie bei den anderen vorgestellten Cmdlets ist die Langschreibweise mit definierten Namen genauso zulässig wie die Kurzschreibweise mit Gruppenname und SamAccountName von Benutzern.

Wie bei jedem Lösch- und Änderungsvorgang wird auch hier empfohlen, die Pipeline einzusetzen und zuerst die gewünschten Objekte zu filtern. Erst im letzten Schritt sollten dann die Änderungen durchgeführt werden.

### Beispiele

Sie wollen dem Benutzer *Hans Dampf* (SamAccountName: *hans.dampf*) das Recht entziehen, auf Drucker3 drucken zu dürfen. Dies wollen Sie damit erreichen, indem Sie ihn aus der Gruppe *LG\_Drucker3\_Drucken* entfernen:

- a) Remove-ADGroupMember -Identity  
"CN=LG\_Drucker3\_Drucken,OU=Hamburg,DC=herdt,DC=ps" -Members "CN=Hans Dampf,CN=Users,DC=herdt,DC=ps"
- b) Remove-ADGroupMember LG\_Drucker3\_Drucken hans.dampf

Sie wollen a) diesen Löschvorgang nur simulieren und b) keine Nachfrage erhalten, ob die Aktion wirklich ausgeführt werden soll:

- a) Remove-ADGroupMember -Identity  
"CN=LG\_Drucker3\_Drucken,OU=Hamburg,DC=herdt,DC=ps" -Members  
"CN=Hans Dampf,CN=Users,DC=herdt,DC=ps" -WhatIf
- b) Remove-ADGroupMember -Identity  
"CN=LG\_Drucker3\_Drucken,OU=Hamburg,DC=herdt,DC=ps" -Members  
"CN=Hans Dampf,CN=Users,DC=herdt,DC=ps" -Confirm:\$False

## 10.5 Benutzer, Gruppen, OUs oder Computer löschen

Normalerweise erfolgt ein Löschvorgang in der PowerShell aus Sicherheitsgründen in zwei Schritten: Zuerst lassen Sie sich die gewünschten Objekte anzeigen, erst dann löschen Sie sie. Damit können Sie sicher sein, dass Sie wirklich nur die gewünschten Daten löschen.

In dem Fall verwenden Sie die Pipeline und übergeben die Objekte an die Cmdlets `Remove-ADUser`, `Remove-ADGroup`, `Remove-ADOrganizationalUnit` und `Remove-ADComputer`.

Werden diese Cmdlets eigenständig eingesetzt, folgt die Syntax der Cmdlets identischen Regeln. Alle Cmdlets arbeiten mit dem Parameter `-Identity`, der als Wert einen definierten Namen in Anführungszeichen erwartet. Zusätzlich funktionieren auch hier stattdessen verkürzte Angaben wie `SamAccountName` für Benutzer-Objekte, `Computername` für Computer-Objekte oder `Gruppenname` für Gruppen-Objekte.

### Beispiele

Der bereits verwendete Benutzer Hans Dampf – mit SamAccountName `hans.dampf` – soll gelöscht werden:

- a) `Get-ADUser -Filter {cn -eq "Hans Dampf"} | Remove-ADUser`
- b) `Remove-ADUser -Identity "CN=Hans Dampf,OU=Hamburg,OU=Benutzer,DC=herdt,DC=ps"`
- c) `Remove-ADUser -Identity hans.dampf`

Die Organisationseinheit *Hamburg* soll gelöscht werden. Die Organisationseinheit ist vor versehentlichem Löschen geschützt:

```
Get-ADOrganizationalUnit -Filter {Name -eq "Hamburg"} |
Set-ADOrganizationalUnit -ProtectedFromAccidentalDeletion $False |
Remove-ADOrganizationalUnit -Confirm:$False
```

- ✓ Die Aktion besteht aus drei Cmdlets, die der Übersichtlichkeit halber mit Zeilenumbrüchen versehen wurden. Geben Sie die Befehle in die PowerShell als eine Zeile ein.
- ✓ Das erste Cmdlet `Get-ADOrganizationalUnit` findet – sofern vorhanden – eine Organisationseinheit *Hamburg* und reicht sie über Pipeline an das nächste Cmdlet weiter.
- ✓ Das Cmdlet `Set-ADOrganizationalUnit` hebt für das übergebene Objekt den Schutz vor versehentlichem Löschen auf und reicht es erneut über die Pipeline weiter.
- ✓ Das letzte Cmdlet der Pipeline löscht schließlich die Organisationseinheit *Hamburg*. Durch den Parameter `-Confirm:$False` wird der Löschvorgang ohne weitere Nachfrage vorgenommen.

## 10.6 Allgemeine Objektverwaltung

### ADObject-Cmdlets zur allgemeinen Objektverwaltung

Neben den speziellen Cmdlets, die Objekte einer bestimmten Objektklasse verwalten, kennt die PowerShell einen Satz an Cmdlets, die einen allgemeineren Ansatz bei der Verwaltung von Objekten verfolgen. All diese Cmdlets unterscheiden nicht von vornherein nach Objektklassen und verwenden das Substantiv *ADObject*. Hierbei handelt es sich um folgende Cmdlets:

Cmdlet	Erläuterung
<code>Get-ADObject</code>	Zeigt Informationen zu einem angegebenen Objekt an
<code>Move-ADObject</code>	Verschiebt das angegebene Objekt an einen anderen Ort
<code>New-ADObject</code>	Erstellt ein neues Objekt

Cmdlet	Erläuterung
Remove-ADObject	Entfernt das angegebene Objekt
Rename-ADObject	Benennt das angegebene Objekt um
Restore-ADObject	Stellt das angegebene (gelöschte) Objekt wieder her
Set-ADObject	Ändert Werte von Eigenschaften des angegebenen Objekts
Sync-ADObject	Repliziert ein Objekt zwischen zwei Domänencontrollern

Die Anwendung und das Verhalten der Cmdlets sind ähnlich den Cmdlets, die speziell für einzelne Objektklassen zur Verfügung stehen.

Sie haben auf den letzten Seiten z. B. gesehen, wie Sie spezielle Benutzer, Gruppen, Organisationseinheiten oder Computer finden konnten. Das Cmdlet Get-ADObject funktioniert ähnlich, wie Sie in der nachfolgenden Abbildung sehen.

```
PS C:\Users\Administrator> Get-ADObject -Filter { Name -like "*test*" }
DistinguishedName          Name    ObjectClass        ObjectGUID
-----            ----   -----            -----
OU=TEST,DC=herdt,DC=ps      TEST    organizationalUnit 0bf3b4b7-8fb9-4b70-96f8...
CN=GG_TEST,OU=TEST,DC=herdt,DC=ps  GG_TEST  group        22a81e21-eb09-48f8-a363...
CN=Testuser,OU=TEST,DC=herdt,DC=ps  Testuser user        201e607f-a541-4076-8283...
```

Es wird ein Filter definiert, der Objekte findet, die die Zeichenkette *test* im Namen führen. Der auffällige Unterschied zu den spezialisierten Cmdlets liegt in der Ausgabe der Eigenschaft *ObjectClass*. Da das Cmdlet nicht zwischen Objektklassen unterscheidet, findet es im Beispiel Gruppen und Organisationseinheiten.

Durch den allgemeineren Ansatz sind ADObject-Cmdlets nicht für Detaileinstellungen gedacht wie das Ändern einer Telefonnummer, sondern für globalere Aufgaben zuständig.

Anhand einiger Beispiele sollen diese Aufgaben verdeutlicht werden:

- ✓ Soll ein Objekt an einen anderen Ort im Active Directory verschoben werden, spielt es keine Rolle, um welche Objektklasse es sich handelt. Sie verwenden für diese Aufgaben das Cmdlet Move-ADObject und spezifizieren das Objekt, das verschoben werden soll (-Identity) und geben den Zielort an, der bereits existieren muss (-TargetPath):

```
Move-ADObject -Identity "CN=Hans Dampf,OU=Hamburg,OU=Benutzer,DC=herdt,DC=ps" -TargetPath "OU=AlleUser,DC=herdt,DC=ps"
```

- ✓ Soll ein Objekt umbenannt werden, treffen dieselben Argumente zu. Sie verwenden das Cmdlet Rename-ADObject mit dem bekannten Parameter -Identity und dem Parameter -NewName zur Festlegung des neuen Namens eines beliebigen Objekts:

```
Rename-ADObject -Identity "CN=Hans Dampf,OU=AlleUser,DC=herdt,DC=ps" -NewName "Hans-Peter Dampf-Hesse"
```

- ✓ Nur über ADObject-Cmdlets können Sie in der PowerShell den Schutz vor versehentlichem Löschen steuern. Dies lesen Sie ausführlich im nächsten Abschnitt direkt im Anschluss.
- ✓ Wenn Sie mit gelöschten Objekten arbeiten, müssen Sie ADObject-Cmdlets verwenden, da diese den Parameter -IncludeDeletedObjects verwenden können.

## 10.7 Schutz vor versehentlichem Löschen

### Versehentlich, nicht „zufällig“

Standardmäßig sind Organisationseinheiten vor versehentlichem Löschen geschützt, andere Objekttypen aber nicht. Einige Seiten zuvor haben Sie den Parameter `-ProtectedFromAccidentalDeletion` im Zusammenspiel mit dem Cmdlet `New-ADOrganizationalUnit` kennengelernt.

Im grafischen Tool *Active Directory-Benutzer und -Computer* sehen Sie auf der Registerkarte *Objekt*, ob das aktuelle Objekt vor versehentlichem Löschen geschützt ist. Die Registerkarte *Objekt* sehen Sie nur dann, wenn Sie im Menü *Ansicht* den Eintrag *Erweiterte Features* aktiviert haben.

Objekt vor zufälligem Löschen schützen

In deutschsprachigen Windows-Editionen wird dieser Schutz nach wie vor an vielen Stellen als Schutz „vor zufälligem Löschen“ bezeichnet. Es handelt sich hierbei um eine unglückliche Übersetzung, da der Zufall in diesem Fall keine Rolle spielt. Sie haben sich beim Löschgong geirrt. Ein Objekt wird unbeabsichtigt oder versehentlich gelöscht, nicht aber zufällig.

Der Schutz vor versehentlichem Löschen verhindert, dass ein Objekt gelöscht wird. Will man das Objekt wirklich löschen, muss man zuvor das Häkchen *Objekt vor zufälligem Löschen schützen* auf der Registerkarte *Objekt* in den Eigenschaften des Objekts entfernen.

Bei dieser Aufgabe helfen grafische Tools und auch die PowerShell, die mit dem vorgestellten Befehl `Set-ADOrganizationalUnit -ProtectedFromAccidentalDeletion:$False` den Schutz vor versehentlichem Löschen direkt aufheben kann.

Wenn Sie den Schutzmechanismus nützlich finden und ihn auf andere Objekttypen und eine größere Anzahl an Objekten anwenden wollen, stehen Sie mit den grafischen Verwaltungstools vor einem Problem. Sie können das angesprochene Häkchen zum Schutz vor versehentlichem Löschen bei jedem einzelnen gewünschten Objekt setzen, allerdings nicht für mehrere Objekte gleichzeitig.

### Die PowerShell hilft mit dem Cmdlet **Set-ADObject**

Bei diesem Problem hilft die PowerShell. 1000 User schnell vor versehentlichem Löschen zu schützen, ist mit grafischen Tools äußerst unpraktisch. Sie müssten denselben Arbeitsschritt 1000-fach wiederholen.

Um herauszufinden, welche PowerShell-Cmdlets den Parameter `-ProtectedFromAccidentalDeletion` unterstützen, geben Sie den folgenden Befehl ein:

```
Get-Command -ParameterName ProtectedFromAccidentalDeletion
```

CommandType	Name	Version	Source
Cmdlet	New-ADAuthenticationPolicy	1.0.0.0	ActiveDi...
Cmdlet	New-ADAuthenticationPolicySilo	1.0.0.0	ActiveDi...
Cmdlet	New-ADCentralAccessPolicy	1.0.0.0	ActiveDi...
Cmdlet	New-ADCentralAccessRule	1.0.0.0	ActiveDi...
Cmdlet	New-ADClaimTransformPolicy	1.0.0.0	ActiveDi...
Cmdlet	New-ADClaimType	1.0.0.0	ActiveDi...
Cmdlet	New-ADFineGrainedPasswordPolicy	1.0.0.0	ActiveDi...
Cmdlet	New-ADObject	1.0.0.0	ActiveDi...
Cmdlet	New-ADOrganizationalUnit	1.0.0.0	ActiveDi...
Cmdlet	New-ADReplicationSite	1.0.0.0	ActiveDi...
Cmdlet	New-ADResourceProperty	1.0.0.0	ActiveDi...
Cmdlet	New-ADResourcePropertyList	1.0.0.0	ActiveDi...
Cmdlet	Set-ADAuthenticationPolicy	1.0.0.0	ActiveDi...
Cmdlet	Set-ADAuthenticationPolicySilo	1.0.0.0	ActiveDi...
Cmdlet	Set-ADCentralAccessPolicy	1.0.0.0	ActiveDi...
Cmdlet	Set-ADCentralAccessRule	1.0.0.0	ActiveDi...
Cmdlet	Set-ADClaimTransformPolicy	1.0.0.0	ActiveDi...
Cmdlet	Set-ADClaimType	1.0.0.0	ActiveDi...
Cmdlet	Set-ADFineGrainedPasswordPolicy	1.0.0.0	ActiveDi...
Cmdlet	Set-ADObject	1.0.0.0	ActiveDi...
Cmdlet	Set-ADOrganizationalUnit	1.0.0.0	ActiveDi...
Cmdlet	Set-ADReplicationSite	1.0.0.0	ActiveDi...
Cmdlet	Set-ADResourceProperty	1.0.0.0	ActiveDi...
Cmdlet	Set-ADResourcePropertyList	1.0.0.0	ActiveDi...

Liste der Cmdlets, die den Parameter `-ProtectedFromAccidentalDeletion` unterstützen

Als Ergebnis erhalten Sie eine Liste mit 24 Cmdlets, die diesen Parameter unterstützen, jeweils 12 New- und 12 Set-Cmdlets, die als Cmdlet-Paar dasselbe Substantiv verwenden. Das ist zu erwarten: Wenn Sie ein Objekt erstellen (*New*), können Sie es direkt vor versehentlichem Löschen schützen. Dasselbe können Sie auch für bereits vorhandene Objekte (*Set*) erreichen.

Überraschend ist eher, dass keine Cmdlets für die einzelnen Objektklassen wie Benutzer, Gruppen und Computer aufgelistet werden. Wenn Sie für diese Objektklassen den Schutz vor versehentlichem Löschen aktivieren wollen, können Sie das mithilfe von PowerShell-Cmdlets einfach erreichen:

Im ersten Schritt suchen Sie die Objekte, die Sie schützen wollen, z. B. alle Gruppen im Container *Users*. Dort sind etwa 20 vordefinierte, vom System erstellte Gruppen zu finden, die keinen Löschschutz besitzen:

```
Get-ADGroup -Filter * -SearchBase "CN=Users,DC=herdt,DC=ps"
```

Im zweiten Schritt setzen Sie die Pipeline ein und ergänzen den obigen Befehl um die Ergänzung des Löschschutzes für die ausgewählten Objekte:

```
Get-ADGroup -Filter * -SearchBase "CN=Users,DC=herdt,DC=ps" | Set-ADObject -ProtectedFromAccidentalDeletion:$True
```

Alle Gruppen im Container *Users* sind damit vor versehentlichem Löschen geschützt.

### Beispiel: Schutz vor versehentlichem Löschen automatisieren

**Plus** Beispieldatei: *schutz.ps1* (Ordner Kapitel 10)

Finden Sie zunächst die gewünschten Objekte, die Sie schützen wollen. Verwenden Sie das Cmdlet `Get-ADObject` anstelle der auf einzelne Objektklassen spezialisierten Cmdlets, erhalten Sie allgemeinere Ergebnisse, die nicht an eine Objektklasse gebunden sind.

Suchen Sie nach allen Active Directory-Objekten im Container *Users*, verwenden Sie den folgenden Befehl:

```
Get-ADObject -Filter * -SearchBase "CN=Users,DC=herdt,DC=ps"
```

- ✓ Der Parameter `-SearchBase` steuert, an welchem Ort das Cmdlet suchen soll. Es werden auch untergeordnete Elemente durchsucht, sofern vorhanden.
- ✓ Es werden alle gefundenen Objektklassen angezeigt: der Container selbst, vordefinierte Gruppen und Benutzer und die von Ihnen manuell hinzugefügten Objekte.
- ✓ Für eine Suche an einem anderen Ort wandeln Sie den Wert des Parameters entsprechend ab.
- ✓ Suchen Sie z. B. nur nach Gruppen und Benutzern, formulieren Sie eine Bedingung als Wert des Parameters `-Filter`:

```
-Filter { (objectClass -eq "User") -or (objectClass -eq "Group") }
```

Anschließend aktivieren Sie mithilfe der Pipeline und dem Cmdlet `Set-ADObject` den Schutz:

```
... | Set-ADObject -ProtectedFromAccidentalDeletion:$True
```

Wenn Sie die bisherigen Überlegungen automatisiert umsetzen wollen, können Sie eine eigene Funktion erstellen, wie das folgende Skript zeigt.

Es wird eine Funktion definiert, die die Eingabe eines definierten Namens erwartet, um dort gespeicherte Gruppen und Benutzer vor versehentlichem Löschen zu schützen bzw. den Schutz zu entfernen:

```
① Function Set-Protection ($Path, [switch]$On)
{
    ② $schalter = $False
        $text = "ENTFERNT."

    ③ If ($On)
    {
        $schalter = $True
        $text = "AKTIVIERT."
    }

    ④ Get-ADObject -Filter { (objectClass -eq "User") -or (objectClass -eq "Group") } -SearchBase $Path |
        Set-ADObject -ProtectedFromAccidentalDeletion: $schalter -PassThru |
        Select-Object -Property Name | Sort-Object -Property Name
    ⑤ Write-Host "`nDer Schutz vor versehentlichem Löschen wird für die
        oben angegebenen Objekte $text"
}
```

Beispieldatei „schutz.ps1“

- ① Die Funktion `Set-Protection` wird definiert. Sie erhält den Parameter `-Path` für die Eingabe des definierten Namens eines Containers bzw. einer Organisationseinheit Ihrer Domäne. Zudem wird der Switch-Parameter `-On` zur Verfügung gestellt, der steuert, ob der Schutz aktiviert oder deaktiviert werden soll.
- ② In der Funktion selbst werden in den ersten Zeilen Variablen definiert, die innerhalb der Funktion benötigt werden. Den Variablen `$schalter` und `$text` werden Ausgangswerte zugewiesen, die dann verwendet werden, wenn der Parameter `-On` beim Aufruf der Funktion nicht verwendet wird (siehe ③).
- ③ Es wird geprüft, ob der Anwender im Aufruf der Funktion den Parameter `-On` verwendet hat. Ist dies der Fall, werden die Werte der im Vorfeld definierten Variablen verändert.
- ④ Die Umsetzung der eigentlichen Aktionen der Funktion wurde als Pipeline mit etlichen Befehlen umgesetzt. Um sie besser erläutern zu können, sehen Sie diese Zeile in einer Aufteilung in Einzelschritte, die nachfolgend erläutert werden:

```

a) Get-ADObject
b) -Filter { (objectClass -eq "User") -or (objectClass -eq "Group") }
   -SearchBase $Path
c) | Set-ADObject -ProtectedFromAccidentalDeletion:$schalter -PassThru
d) | Select-Object -Property Name | Sort-Object -Property Name

```

- a) Ausgangspunkt ist das Cmdlet `Get-ADObject`, um mit dem Parameter `-Filter` möglichst flexible Bedingungen formulieren zu können, die nicht auf eine Objektklasse beschränkt sind.
- b) Der Parameter `-Filter` steuert über die angegebene Bedingung, welche Objekte gefunden werden. Hier sind zwei Bedingungen miteinander verknüpft (`-or`), es wird entweder nach Objekten der Klasse *User* oder der Klasse *Gruppe* gesucht. Mit dem Wert des Parameters `-SearchBase` legen Sie fest, welche Organisationseinheit oder welcher Container inklusive aller untergeordneten Objekte durchsucht werden soll. Der Ort wird über die Parametereingabe `-Path` beim Aufruf der Funktion festgelegt.
- c) Die gefilterten Objekte werden über die Pipeline an das folgende Cmdlet `Set-ADObject` übergeben. Das Cmdlet aktiviert oder deaktiviert mithilfe des Parameters `-ProtectedFromAccidentalDeletion` den Schutz des Objekts vor versehentlichem Löschen. Ob der Schutz aktiviert oder deaktiviert wird, hängt von der Angabe des Switch-Parameters `-On` beim Aufruf der Funktion ab. Wurde der Parameter angegeben, wird der Variablen `$schalter` bei ③ der Wert `$True` zugewiesen. Damit wird der Schutz aktiviert. Wurde der Parameter nicht verwendet, hat die Variable `$schalter` einen anderen Wert (`$False`), und der Schutz wird deaktiviert. Zusätzlich werden die Objekte durch den Parameter `-PassThru` an das nächste Cmdlet in der Pipeline weitergereicht.
- d) Die Eigenschaft *Name* der übergebenen Objekte wird ausgewählt. Nach dieser Eigenschaft wird aufsteigend sortiert. Letztlich erfolgt eine Ausgabe der Objekte. Oder kurz: Die Namen aller Objekte, deren Schutz vor versehentlichem Löschen aktiviert bzw. deaktiviert wurde, werden aufsteigend sortiert ausgegeben.
- ⑤ Zum Ende der Funktion erfolgt eine Ausgabe, dass für die ausgegebenen Objekte der Schutz vor versehentlichem Löschen aktiviert oder entfernt wurde. Die Ausgabe hängt vom Wert der Variablen `$text` ab, der wiederum von der Angabe des Parameters `-On` beim Aufruf der Funktion abhängt.

Die oben beschriebene Funktion soll Sie vorrangig auf weitere, eigene Ideen bringen, was mit der PowerShell möglich ist. Sie können z. B. die Funktion erweitern und mit Begin-, Process- und End-Anweisungsblöcken arbeiten. Den fest definierten Filter können Sie nach eigenen Vorstellungen umformulieren. Vielleicht gestalten Sie den Filter auch variabel und steuern ihn über weitere Parameter, die vom Benutzer eingegeben werden.

## 10.8 Das Active Directory-Verwaltungscenter

Das Active Directory-Verwaltungscenter ist ein grafisches Verwaltungstool, das Sie über das Menü *Tools* im Server-Manager oder durch Eingabe von `dsac` z. B. in der PowerShell starten können. Das Verwaltungscenter erledigt in erster Linie Active Directory-Routineaufgaben.

Interessant im Zusammenhang mit der PowerShell ist, dass das Active Directory-Verwaltungscenter alle ausgeführten Tätigkeiten in PowerShell-Befehle übersetzt und diese ausführt. Am unteren Rand der Anwendung finden Sie einen unauffälligen Bereich *WINDOWS POWERSHELL-VERLAUF HISTORY*, den Sie erweitern (ausklappen) können.



In diesem Bereich protokolliert die Anwendung alle vorgenommenen Aktionen der aktuellen Sitzung. In der nachfolgenden Abbildung sehen Sie den erweiterten Bereich *WINDOWS POWERSHELL-VERLAUF HISTORY* mit zwei PowerShell-Cmdlets (markiert durch den Pfeil). Die Cmdlets wurden automatisch durch Änderungen an der Organisationseinheit *test* eingetragen.

The screenshot shows the Active Directory-Verwaltungcenter interface. On the left, there's a navigation pane with 'Übersicht' and 'herdt (lokal)' selected. The main area displays a table of objects under 'herdt (lokal) (14)', including 'LostAndFound', 'Managed Service Accounts', 'NTDS Quotas', 'Program Data', 'System', 'TEST', 'TPM Devices', and 'Users'. A context menu is open over the 'TEST' object, listing options like 'Neu', 'Löschen', 'Verschieben...', 'Unter diesem Knoten suchen', 'Eigenschaften', 'herdt (lokal)', 'Domänencontroller ändern', 'Gesamtstrukturfunktionseb...', 'Domänenfunktionsebene h...', and 'Papierkorb aktivieren...'. Below the table is a section titled 'WINDOWS POWERSHELL-VERLAUF HISTORY' containing a history of cmdlets. A green arrow points to the search bar in this section. The history shows two entries:

Cmdlet	Zeitstempel
Set-ADObject -Identity:"OU=TEST,DC=herdt,DC=ps" -ProtectedFromAccidentalDeletion:\$false -Server:"Server2016.herdtps"	10.08.2016 09:38:39
Set-ADOrganizationalUnit -Description:"Dies ist die Beschreibung der OU TEST." -Identity:"OU=TEST,DC=herdt,DC=ps" -Server:"Server2016.herdtps"	10.08.2016 09:39:29

#### Automatisch generierte PowerShell-Cmdlets im Verwaltungcenter

Wie der Markierungsrahmen zeigt, steht in diesem Bereich auch eine Bearbeitungsleiste zur Verfügung. Sie können folgende Tätigkeiten in der Bearbeitungsleiste ausführen:

- ✓ **Suchen:** Durch Eingabe eines Suchbegriffs werden die gelisteten Cmdlets gefiltert.
- ✓ **Kopieren:** Sie können ein oder mehrere Cmdlets kopieren, um sie z. B. in der PowerShell weiterzuverwenden.
- ✓ **Aufgabe starten:** Sie geben den Namen einer Aufgabe ein, führen mehrere Aktionen aus, klicken dann auf **Aufgabe beenden**. Damit sind die Aktionen der Aufgabe unter dem Namen der Aufgabe gelistet. Die Ansicht der Befehle der Aufgabe lässt sich auf- und einklappen.
- ✓ **Alle löschen:** Die Liste der PowerShell-Cmdlets wird geleert.
- ✓ **Alle anzeigen:** Alle Cmdlets der History werden angezeigt, auch die ausgeführten Befehle, die nicht auf Aktionen des Benutzers zurückzuführen sind. So werden z. B. beim Start einige Befehle automatisch ausgeführt.
- ✓ **Hilfe:** Eine Webseite des TechNet von Microsoft wird geöffnet, die Ihnen in englischer Sprache Hilfe für Cmdlets zur Verwaltung des Active Directory anzeigt.

Das Active Directory-Verwaltungcenter verbindet alte und neue Welt. Während ein Administrator vornehmlich mit grafischen Tools gearbeitet hat und dies für viele Aufgaben immer noch tun kann, bildet den Kern eines modernen Windows-Betriebssystems mittlerweile die PowerShell. Die Anwendung erlaubt nach wie vor die Administration mithilfe grafischer Verwaltungstools, bietet aber zusätzlich einen Einblick in die – sonst versteckte – interne Umsetzung in PowerShell-Cmdlets.

Damit wird die Verwendung des Active Directory-Verwaltungstools noch attraktiver. Neben der Ausführung von Routineaufgaben können Sie nebenbei Ihre PowerShell-Kenntnisse erweitern oder fertigen Code kopieren und später in der PowerShell nach eigenen Vorstellungen modifizieren.

## 10.9 Kurz zusammengefasst

Das Modul *ActiveDirectory* bietet eine sehr umfangreiche Sammlung von Cmdlets, die weitreichende Administrationstätigkeiten in Ihrer Domäne erlauben. Der Anspruch von Microsoft, eine möglichst umfassende bzw. komplette Verwaltung mit der PowerShell zu erlauben, wurde auch für das Active Directory fast vollständig umgesetzt. Es existieren Cmdletfamilien für alle Objektgruppen und auch allgemeine Cmdlets zum Erstellen, Löschen, Ändern und Auslesen von Active Directory-Objekten.

Sie können weiterhin für die meisten administrativen Tätigkeiten im Active Directory-Umfeld die gewohnten grafischen Verwaltungstools verwenden. Die PowerShell ist bei vielen Routineaufgaben ohnehin nur als Alternative gedacht. Die PowerShell ist vor allem dann immer eine interessante Alternative, wenn es um die Automatisierung von Vorgängen oder die Ausführung von identischen Befehlen auf mehreren Rechnern oder für eine Vielzahl von Objekten geht. Der nachträgliche Schutz aller Benutzer, Gruppen und Computer vor versehentlichem Löschen mag hier als Beispiel dienen.

Die PowerShell bietet eine einheitliche Syntax, an die sich der Anwender nach kurzer Zeit gewöhnt (Verb-Substantiv). Durch die Namensgebung der Cmdlets wird häufig auf den ersten Blick ihr Einsatzzweck sichtbar. Dies gilt ebenso für die Benennung der Parameter (z. B. `-ProtectedFromAccidentalDeletion`).

## 10.10 Übung

### Mit den Cmdlets des Moduls *ActiveDirectory* arbeiten

Übungsdatei: `schutz.ps1`

Ergebnisdatei: `10_ergebnisse.txt`

1. Nennen Sie die jeweils vier hauptsächlich verwendeten Cmdlets für die Verwaltung der Objektgruppen Benutzer, Computer, Gruppen und Organisationseinheiten.
  2. a) Geben Sie alle Benutzer aus, die sich im Container *Users* Ihrer Domäne befinden.
  - b) Zeigen Sie alle Gruppen an, deren Name mit der Zeichenfolge *Domäne* beginnt.
  - c) Zeigen Sie Informationen zur Organisationseinheit *Domain Controllers* an.
3. a) Legen Sie einen neuen Computer *Client42* im Standardcontainer für Computerobjekte an und schützen Sie ihn vor versehentlichem Löschen.
  - b) Erstellen Sie eine globale Sicherheitsgruppe *HB-Mitarbeiter* in der Organisationseinheit Bremen (die Sie vorher erstellen).
  - c) Erstellen Sie einen Benutzer *Harald Herdt* in der Organisationseinheit *Bremen*.
  - d) Fügen Sie den neuen Benutzer als Mitglied der Gruppe *HB-Mitarbeiter* hinzu.
4. Fügen Sie beliebige Werte für die Eigenschaften *Beschreibung* und *Firma* für den Benutzer *Harald Herdt* hinzu.
5. Finden Sie heraus, welche Benutzer in der vordefinierten Gruppe *Abgelehnte RODC-Kennwort-replikationsgruppe* sind.
  - a) Lassen Sie eine Liste der Namen der Gruppenmitglieder ausgeben.
  - b) Lassen Sie eine Liste der Namen der Mitglieder ausgeben, die Einzelobjekte sind und mittelbar oder unmittelbar der Gruppe angehören.
6. Verschieben Sie die Gruppe *HB-Mitarbeiter* in den Container *Users*.
7. Schützen Sie alle in der Domäne vorhandenen Benutzer vor versehentlichem Löschen.
8. Verwenden Sie das Active Directory-Verwaltungszentrum, setzen Sie das Kennwort eines Benutzers zurück und ändern Sie die Beschreibung einer Gruppe. Schauen Sie sich danach die dazugehörigen PowerShell-Befehle im Bereich der *Windows PowerShell-Verlauf History* an.

# 11 Weitere Angaben im Active Directory



**Beispieldatei:** 11\_kompletter code.txt (Ordner Kapitel 11)

## 11.1 Arbeit mit fein abgestimmten Kennwortrichtlinien

### Mehrere Kennwort- und Kontosperrungsrichtlinien in einer Domäne

Fein abgestimmte Kennwortrichtlinien (*password settings objects, PSO*) bieten seit Windows Server 2008 die Möglichkeit, verschiedene Kennwort- und Kontosperrungsrichtlinien für verschiedene Benutzer und Gruppen in einer Domäne zu definieren. In Domänen, die mit früheren Microsoft Server-Betriebssystemen betrieben werden, kann nur eine Kennwortrichtlinie und Kontosperrungsrichtlinie angewendet werden.

In einer einzigen Domäne können Sie mit fein abgestimmten Kennwortrichtlinien z. B. strengere Einstellungen für Konten mit höheren Berechtigungen oder mit besonderer Sicherheitseinstufung (wie Administratoren, Forschungsabteilung etc.) anwenden und weniger strenge Einstellungen für normale Mitarbeiter (wie Verwaltung, Lager etc.).

Während Sie bei Windows Server 2008 (R2) noch einen kaum bedienbaren Assistenten im ADSI-Editor zur Erstellung fein abgestimmter Kennwortrichtlinien verwenden mussten, können Sie ab Windows Server 2012 auf das Active Directory-Verwaltungscenter und seine ausgefälten, komfortablen Formulare zurückgreifen.

Fein abgestimmte Kennwortrichtlinien werden in Ihrer Domäne im Container *Password Settings Container* gespeichert. Dieser Container ist im Container *System* zu finden. Ein vollständig definierter Name eines Richtlinienobjekts lautet z. B. CN=<Name PSO>, CN=Password Settings Container, CN=System, DC=herdt, DC=ps.

### Fein abgestimmte Kennwortrichtlinien mit der PowerShell verwalten

Um fein abgestimmte Kennwortrichtlinien verwalten zu können, stellt die PowerShell vier Cmdlets mit den bekannten Verben New, Set, Get und Remove zur Verfügung. Sie können mit ihnen Richtlinien erstellen, verändern, auslesen und löschen. Allen Cmdlets gemeinsam ist das verwendete Substantiv `ADFineGrained-PasswordPolicy`:

Cmdlet	Erläuterung
<code>Get-ADFineGrainedPasswordPolicy</code>	Zeigt Informationen zu einem Kennwort-Richtlinienobjekt an
<code>New-ADFineGrainedPasswordPolicy</code>	Erstellt ein neues Kennwort-Richtlinienobjekt
<code>Remove-ADFineGrainedPasswordPolicy</code>	Entfernt ein Kennwort-Richtlinienobjekt
<code>Set-ADFineGrainedPasswordPolicy</code>	Ändert die Werte der Eigenschaften eines vorhandenen Kennwort-Richtlinienobjekts

### Eine fein abgestimmte Kennwortrichtlinie erstellen

Um ein Richtlinienobjekt zu erstellen, verwenden Sie das Cmdlet `New-ADFineGrainedPasswordPolicy`. Da die Festlegung einer Richtlinie etlicher Einstellungen bedarf, müssen Sie eine Anzahl Parameter verwenden. Im Folgenden sehen Sie die Parameter, ihre möglichen Werte und eine kurze Erläuterung der Parameter:

Parameter	Erläuterung
-Name	Name des Kennwort-Richtlinienobjekts Wert: Zeichenkette
-Description	Beschreibung des Kennwort-Richtlinienobjekts Wert: Zeichenkette
-Precedence	Rangfolge für Kennworteinstellungen (je kleiner der Wert, desto höher ist die Priorität der Kennwortrichtlinie) Wert: Ganzzahl größer als 0
-MinPasswordLength	Minimale Kennwortlänge für Benutzerkonten Wert: 0 (keine Vorgabe) bis 255
-MinPasswordAge	Das minimale Kennwortalter für Benutzerkonten legt fest, nach welcher Zeitspanne ein neu gesetztes Kennwort frühestens geändert werden darf. Wert: im Format x:xx:xx:xx (Tage.Stunden:Minuten:Sekunden)
-MaxPasswordAge	Maximales Kennwortalter für Benutzerkonten Wert: (Never) für nie, kann nicht auf 0 gesetzt werden, im Format x:xx:xx:xx (Tage.Stunden:Minuten:Sekunden)
-PasswordHistoryCount	Der Wert legt fest, wie viele zuletzt verwendete Kennwörter gespeichert werden, bevor ein altes Kennwort nochmals verwendet werden darf. Wert: 0 (deaktiviert) bis 1024
-ComplexityEnabled	Kennwortkomplexitätsstatus für Benutzerkonten Wert \$True oder \$False, empfohlen wird \$True
-ReversibleEncryptionEnabled	Legt fest, ob das Kennwort umkehrbar verschlüsselt gespeichert werden soll Wert: \$True oder \$False, empfohlen wird \$False
-LockoutThreshold	Legt die Sperrschwelle (Anzahl der erfolglosen Anmeldeversuche) für das Sperren von Benutzerkonten fest Wert: 0 bis 65535 (0 wird nicht empfohlen, da der Wert die Schwelle deaktiviert) Die beiden folgenden Parameter -LockoutObservationWindow und -LockoutDuration werden nur berücksichtigt, wenn der Wert dieses Parameters ungleich 0 ist.
-LockoutObservationWindow	Mit dem Wert wird festgelegt, welche Zeitspanne erfolglose Anmeldeversuche gespeichert werden, bevor der Zähler erfolgreicher Anmeldungen zurückgesetzt wird. Wert: im Format x:xx:xx:xx (Tage.Stunden:Minuten:Sekunden)
-LockoutDuration	Sperrdauer für gesperrte Benutzerkonten

Um z. B. eine Richtlinie zu definieren, die höheren Sicherheitsanforderungen entspricht als die normalerweise in Domänen verwendete Kennwortrichtlinie, geben Sie den folgenden Befehl ein (das Zeichen ` (Backtick) erlaubt einen Zeilenumbruch in einer Befehlszeile):

```
New-ADFineGrainedPasswordPolicy ` 
-Name:"Sicher" ` 
-Description:"sichere Einstellungen für sicherheitsrelevante Bereiche" ` 
-Precedence:"100" ` 
-MaxPasswordAge:"28.00:00:00" ` 
-MinPasswordAge:"1:00:00:00" ` 
-MinPasswordLength:"12" ` 
-PasswordHistoryCount:"100" ` 
-ReversibleEncryptionEnabled:$False ` 
-ComplexityEnabled:$True ` 
-LockoutDuration:"1:00:00:00" ` 
-LockoutObservationWindow:"12:00:00" ` 
-LockoutThreshold:"3"
```

Erstellen Sie weitere Kennwort-Richtlinienobjekte, beachten Sie, dass der Parameter `-Name` einen eindeutigen Wert erhalten muss.

### Weitere Beispiele

Mit den folgenden Befehlen werden zwei weitere Kennwort-Richtlinienobjekte erstellt. Somit sind in der Domäne drei Richtlinienobjekte *Sicher*, *Mittel* und *Unsicher* vorhanden:

```
New-ADFineGrainedPasswordPolicy -Name:"Mittel" -Description:"mittlere 
Sicherheitseinstellungen" -Precedence:"200" -MaxPasswordAge:"42:00:00:00" 
-MinPasswordAge:"1:00:00:00" -MinPasswordLength:"7" 
-PasswordHistoryCount:"24" -ComplexityEnabled:$True 
-ReversibleEncryptionEnabled:$False -LockoutDuration:"00:30:00" 
-LockoutObservationWindow:"00:30:00" -LockoutThreshold:"5"
```

```
New-ADFineGrainedPasswordPolicy -Name:"Unsicher" -Description:"Lasch - für 
wenig sicherheitsrelavante Bereiche" -Precedence:"300" 
-MaxPasswordAge:"90:00:00:00" -MinPasswordAge:"00:00:00" 
-MinPasswordLength:"5" -PasswordHistoryCount:"0" -ComplexityEnabled:$False 
-ReversibleEncryptionEnabled:$False -LockoutDuration:"00:30:00" 
-LockoutObservationWindow:"00:30:00" -LockoutThreshold:"0"
```

### Eine fein abgestimmte Kennwortrichtlinie ändern

Änderungen an Kennwort-Richtlinienobjekten können mithilfe des Cmdlets `Set-ADFineGrainedPasswordPolicy` vorgenommen werden. Das Cmdlet kennt zur Auswahl der gewünschten Richtlinie nur den Parameter `-Identity`, so dass eine Änderung pro Befehl an genau einer Richtlinie vorgenommen werden kann. Wollen Sie identische Änderungen an mehreren Objekten vornehmen, muss das Cmdlet die Objekte über die Pipeline empfangen.

Sie können alle Parameter verwenden, die Sie bereits vom Cmdlet `New-ADFineGrainedPasswordPolicy` kennen. Wollen Sie das Richtlinienobjekt allerdings umbenennen, verwenden Sie das Cmdlet `Rename-ADObject`.

### Beispiele

Sie möchten die Werte der Parameter `-ComplexityEnabled`, `-MaxPasswordAge` und `-Description` des Kennwort-Richtlinienobjekts *Unsicher* ändern:

```
Set-ADFineGrainedPasswordPolicy -Identity Unsicher -ComplexityEnabled:$True
-MaxPasswordAge "60:00:00:00" -Description "Richtlinie für wenig
sicherheitsrelevante Bereiche"
```

Sie möchten zu Testzwecken für alle Kennwort-Richtlinienobjekte Ihrer Domäne erlauben, Kennwörter zu wählen, die den Komplexitätsvoraussetzungen nicht entsprechen:

```
Get-ADFineGrainedPasswordPolicy -Filter * | Set-ADFineGrainedPasswordPolicy
-ComplexityEnabled:$false
```

Sie möchten das Kennwort-Richtlinienobjekt *Mittel* in *Standard* umbenennen:

```
Rename-ADObject -Identity "CN=Mittel,CN=Password Settings
Container,CN=System,DC=herdt,DC=ps" -NewName "Standard"
```

## Informationen zu fein abgestimmten Kennwortrichtlinien abrufen

Um Informationen über fein abgestimmte Kennwortrichtlinien zu erhalten, verwenden Sie das Cmdlet `Get-ADFineGrainedPasswordPolicy`. Wenn Sie nach Kriterien filtern wollen, setzen Sie den Parameter `-Filter` ein. Suchen Sie eine bestimmte Richtlinie, verwenden Sie den Parameter `-Identity` und geben als Wert den definierten Namen einer Richtlinie ein. Den Umgang mit beiden Parametern haben Sie ausführlich im letzten Kapitel gelernt.

### Beispiele

Sie wollen sich informieren, welche fein abgestimmten Kennwortrichtlinien in Ihrer Domäne vorhanden sind (Sie erhalten eine Liste mit Namen der vorhandenen Richtlinien):

```
Get-ADFineGrainedPasswordPolicy -Filter * | Select-Object -Property Name
```

Sie benötigen eine tabellarische Aufstellung mit dem Namen, dem Prioritätswert, der minimalen Kennwortlänge und dem maximalen Kennwortalter aller Kennwort-Richtlinienobjekte Ihrer Domäne:

```
Get-ADFineGrainedPasswordPolicy -Filter * | Format-Table -Property Name,
Precedence, MinPasswordLength, MaxPasswordAge -AutoSize
```

Name	Precedence	MinPasswordLength	MaxPasswordAge
Sicher	100	12	28.00:00:00
Standard	200	7	42.00:00:00
Unsicher	300	5	60.00:00:00

Sie suchen alle fein abgestimmten Kennwortrichtlinien, bei der die minimale Kennwortlänge auf mehr als sieben Zeichen festgelegt wurde:

```
Get-ADFineGrainedPasswordPolicy -Filter {MinPasswordLength -gt 7}
```

Sie wollen Informationen zum Kennwort-Richtlinienobjekt *Sicher* abrufen:

- a) `Get-ADFineGrainedPasswordPolicy -Identity Sicher`
- b) `Get-ADFineGrainedPasswordPolicy -Identity "CN=Sicher,CN=Password Settings Container,CN=System,DC=herdt,DC=ps"`

```
Administrator: Windows PowerShell
PS C:\Users\Administrator> Get-ADFineGrainedPasswordPolicy -Identity Sicher

AppliesTo          : {CN=Testuser,OU=TEST,DC=herdt,DC=ps,
ComplexityEnabled : False
DistinguishedName : CN=Sicher,CN=Password Settings
LockoutDuration   : 1.00:00:00
LockoutObservationWindow : 12:00:00
LockoutThreshold  : 3
MaxPasswordAge    : 28.00:00:00
MinPasswordAge    : 1.00:00:00
MinPasswordLength : 12
Name               : Sicher
ObjectClass        : msDS>PasswordSettings
ObjectGUID         : 6f226573-7c8d-4478-8be0-d8aa607fff10
PasswordHistoryCount : 100
Precedence         : 100
ReversibleEncryptionEnabled : False
```

Sie wollen alle verfügbaren Informationen zu Ihrem Kennwort-Richtlinienobjekt *Mittel* abrufen:

```
Get-ADFineGrainedPasswordPolicy -Identity Mittel -Properties *
```

### Eine fein abgestimmte Kennwortrichtlinien löschen

Auch zum Löschen von Kennwort-Richtlinienobjekten stellt die PowerShell das passende Cmdlet zur Verfügung: Remove-ADFineGrainedPasswordPolicy. Wie von anderen Cmdlets zum Löschen von Objekten bekannt, wird auch hier der Parameter **-Identity** zur Bestimmung des zu löschenen Richtlinienobjekts verwendet. Wollen Sie mehrere Richtlinienobjekte löschen, filtern Sie mit dem Parameter **-Filter**.

Alternativ zum direkten Aufruf des Cmdlets steht als sicherer Weg die Pipeline zur Verfügung: Zuerst suchen Sie die gewünschte Richtlinie mit Get-ADFineGrainedPasswordPolicy. Im zweiten Schritt übergeben Sie das gewünschte Objekt über die Pipeline an das Cmdlet Remove-ADFineGrainedPasswordPolicy, das ohne weitere Parameter ausgeführt wird.

### Beispiele

Sie wollen das Kennwort-Richtlinienobjekt *Sicher* löschen (hier durch Angabe des Parameters **-WhatIf** nur simuliert):

- a) Remove-ADFineGrainedPasswordPolicy -Identity Sicher -WhatIf
- b) Remove-ADFineGrainedPasswordPolicy -Identity "CN=Sicher,CN=Password Settings"

Sie wollen alle Kennwort-Richtlinienobjekte in Ihrer Domäne löschen, wobei die Löschung für jedes Objekt bestätigt werden soll:

```
Get-ADFineGrainedPasswordPolicy -Filter * | Remove-
ADFineGrainedPasswordPolicy -Confirm
```

### Richtlinienanwendung steuern

Mit den bisher vorgestellten Befehlen verwalten Sie Kennwort-Richtlinienobjekte. Eine Verknüpfung mit einem Benutzer oder einer Gruppe wird mit diesen Befehlen aber nicht vorgenommen.

Zur Erledigung dieser Aufgaben stellt die PowerShell die folgenden Cmdlets bereit:

Cmdlet	Erläuterung
Add-ADFineGrainedPasswordPolicySubject	Fügt Benutzer oder Gruppen hinzu, auf die ein Kennwort-Richtlinienobjekt angewendet werden soll
Get-ADFineGrainedPasswordPolicySubject	Zeigt an, auf welche Benutzer und Gruppen ein Kennwort-Richtlinienobjekt angewendet werden soll
Remove-ADFineGrainedPasswordPolicySubject	Legt einen Benutzer oder eine Gruppe fest, auf die ein Kennwort-Richtlinienobjekt nicht mehr angewendet werden soll

Auch wenn die Bezeichnung mancher Cmdlets etwas sperrig ist, kann man aus den Bezeichnungen schnell den Einsatzzweck ablesen. Ein bestehendes Kennwort-Richtlinienobjekt kann mit einem Benutzer verknüpft werden (*Add*), die Verknüpfung kann gelöst werden (*Remove*). Zudem können Informationen über die Verknüpfung eines Richtlinienobjekts mit Benutzern und Gruppen angezeigt werden (*Get*).

Wollen Sie ein Kennwort-Richtlinienobjekt mit einem Benutzer oder einer Gruppe verknüpfen, verwenden Sie das Cmdlet `Add-ADFineGrainedPasswordPolicySubject` und bezeichnen mithilfe des Parameters `-Identity` eindeutig die Richtlinie. Zusätzlich geben Sie als Wert des Parameters `-Subjects` ein oder mehrere Benutzer- oder Gruppenobjekte an. Dieses Vorgehen gilt auch für das Lösen der Verknüpfung zwischen Benutzer- oder Gruppenobjekt und Kennwort-Richtlinienobjekt.

Zum Auslesen von Informationen reicht die Verwendung des Cmdlets `Get-ADFineGrainedPasswordPolicySubject` mit dem Parameter `-Identity` zur genauen Bezeichnung des gewünschten Kennwort-Richtlinienobjekts.

### Beispiele

Sie möchten das Kennwort-Richtlinienobjekt *Sicher* auf den Benutzer *Harald Herdt* anwenden (bei beiden ist eine Kurz- und Langschreibweise möglich):

- a) `Add-ADFineGrainedPasswordPolicySubject -Identity Sicher -Subjects "Harald Herdt"`
- b) `Add-ADFineGrainedPasswordPolicySubject -Identity "CN=Sicher,CN=Password Settings Container,CN=System,DC=herdt,DC=ps" -Subjects "CN=Harald Herdt, OU=Bremen, DC=herdt, DC=ps"`

Sie möchten das Kennwort-Richtlinienobjekt *Sicher* auf den Benutzer *Harald Herdt* und die Gruppen *HH-Mitarbeiter* und *HB-Mitarbeiter* anwenden (Kurzschreibweise):

```
Add-ADFineGrainedPasswordPolicySubject -Identity Sicher -Subjects "Harald Herdt", "HH-Mitarbeiter", "HB-Mitarbeiter"
```

Zur Überprüfung, ob die Verknüpfungen angelegt wurden, wollen Sie kontrollieren, welche Personen und Gruppen momentan mit dem Kennwort-Richtlinienobjekt *Sicher* verknüpft sind:

```
Get-ADFineGrainedPasswordPolicySubject -Identity Sicher
```

Sie wollen die Verknüpfung des Benutzers *Harald Herdt* vom Kennwort-Richtlinienobjekt *Sicher* lösen (ohne Bestätigungsfrage):

```
Remove-ADFineGrainedPasswordPolicySubject -Identity Sicher -Subjects "Harald Herdt" -Confirm:$False
```

## Welche Richtlinie wird effektiv auf einen bestimmten Benutzer angewendet?

Sie haben Kennwort-Richtlinienobjekte erstellt und mit Benutzern bzw. Gruppen verknüpft. Wie aber werden fein abgestimmte Kennwortrichtlinien auf Benutzer angewendet?

Der Wert des Parameters `-Precedence` eines Kennwort-Richtlinienobjekts ist der entscheidende Parameter, da er steuert, ob das Kennwort-Richtlinienobjekt überhaupt auf einen Benutzer angewendet wird. Da nur **ein** Kennwort-Richtlinienobjekt (PSO) auf einen Benutzer angewendet werden kann, gibt es folgende Prioritätsregeln, falls mehrere Objekte existieren:

- ✓ Ist ein PSO direkt mit dem Benutzerkonto verknüpft? Dann wird dieses PSO angewendet.
- ✓ Sind mehrere PSOs mit dem Benutzerkonto verknüpft? Dann wird das PSO mit dem niedrigsten Wert des Parameters `-Precedence` angewendet. Haben mehrere PSOs denselben Wert, wird das PSO mit der niedrigsten GUID (*globally unique identifier*; Objekt-ID) angewendet.
- ✓ Ist keine PSO direkt auf das Benutzerkonto angewendet, sondern auf eine Gruppe (und damit mittelbar auch auf das Benutzerkonto als Gruppenmitglied), gelten dieselben Regeln, als würde das PSO direkt auf das Benutzerkonto angewendet.
- ✓ Erst wenn kein PSO mittelbar oder unmittelbar mit einem Benutzerkonto verknüpft ist, werden die Einstellungen aus der Gruppenrichtlinie *Default Domain Policy* angewendet.

Es ist von Vorteil, diese Regeln für eine effektive Verwaltung der Kennwort-Richtlinienobjekte zu kennen. Die PowerShell stellt ein weiteres Cmdlet zur Verfügung, das Informationen für einen Benutzer anzeigt, welches Kennwort-Richtlinienobjekt auf ihn wirkt und welche Kennwortoptionen konfiguriert sind: `Get-ADUser-ResultantPasswordPolicy`. Es können ausschließlich Informationen für ein Benutzerobjekt angezeigt werden.

### Beispiel

Um herauszufinden, welches Kennwort-Richtlinienobjekt effektiv auf den Benutzer *Harald Herdt* angewendet wird, geben Sie den folgenden Befehl ein:

```
Get-ADUserResultantPasswordPolicy -Identity "Harald Herdt"
```

Das effektive, auf den Benutzer angewendete Richtlinienobjekt wird mit seinen Einstellungen angezeigt.

Sollten Sie in diesem Befehl einen Benutzer angegeben haben, auf den kein Kennwort-Richtlinienobjekt angewendet wird, bleibt die Ausgabe leer. In dem Fall werden auf diesen Benutzer die Standardeinstellungen der Gruppenrichtlinie *Default Domain Policy* angewendet.

## 11.2 Verwaltung von Active Directory-Konten

### Cmdlets zum Umgang mit Active Directory-Konten

Sie haben im letzten Kapitel eine große Anzahl Cmdlets zur Verwaltung der einzelnen Objektklassen wie Benutzer und Computer kennengelernt. Die Einzelobjekte dieser Klassen haben aus Sicht eines Administrators Gemeinsamkeiten hinsichtlich der Verwaltung:

- ✓ Sie können die Konten aktivieren bzw. deaktivieren.
- ✓ Sie können den Zeitpunkt festlegen und löschen, wann ein Konto ablaufen soll.
- ✓ Sie können Kennwörter für diese Konten vergeben.
- ✓ Sie können Werte der Benutzerkontensteuerung verwalten.

Zur Erledigung der genannten Aufgaben bietet die PowerShell folgende Cmdlets:

Cmdlet	Erläuterung
Disable-ADAccount	Deaktiviert das Konto eines Active Directory-Benutzers, -Computers oder -Dienstkontos
Enable-ADAccount	Aktiviert das Konto eines Active Directory-Benutzers, -Computers oder -Dienstkontos
Search-ADAccount	Sucht ein Active Directory-Benutzer-, -Computer- oder -Dienstkontos
Unlock-ADAccount	Entsperrt ein Active Directory-Benutzer-, -Computer- oder -Dienstkontos
Clear-ADAccountExpiration	Entfernt das Ablaufdatum für ein Active Directory-Benutzer- oder -Computerkonto
Set-ADAccountExpiration	Legt ein Ablaufdatum für ein Active Directory-Benutzer- oder -Computerkonto fest
Set-ADAccountPassword	Legt ein Kennwort für ein Active Directory-Benutzer-, -Computer- oder -Dienstkontos fest

### Ein Konto aktivieren bzw. deaktivieren

Das Aktivieren bzw. Deaktivieren eines Kontos erfolgt über die Cmdlets `Enable-ADAccount` bzw. `Disable-ADAccount` unter Verwendung des Parameters `-Identity`, mit dem das gewünschte Konto spezifiziert wird. Nach Deaktivierung eines Kontos ist es nicht mehr möglich, sich mit diesem Konto an der Domäne anzumelden. Auch hier gilt: Wollen Sie mehrere Objekte aktivieren bzw. deaktivieren, verwenden Sie die Pipeline.



Besitzt ein Konto kein Kennwort, lässt es sich nicht aktivieren. Ein Kennwort, das den Domänenanforderungen bezüglich Länge, Komplexität und Verlauf entspricht, ist Voraussetzung für die erfolgreiche Aktivierung eines Kontos.

### Beispiele

Sie wollen das Computerkonto `Client42` deaktivieren (für Computerkonten muss der definierte Name verwendet werden):

```
Disable-ADAccount -Identity "CN=Client42,CN=Computers,DC=herdt,DC=ps"
```

Sie wollen das bislang deaktivierte Konto des Benutzers `Harald Herdt` aktivieren (bei Benutzern ist auch die Kurzschreibweise zulässig):

- a) `Enable-ADAccount -Identity "Harald Herdt"`
- b) `Enable-ADAccount -Identity "CN=Harald Herdt,OU=Bremen,DC=herdt,DC=ps"`

Sie legen 50 neue Benutzerkonten `Student-01` bis `Student-50` für Praktikanten in Ihrer Domäne an. Da die Praktikanten erst in einigen Wochen bei Ihrem Unternehmen beginnen, wollen Sie die Konten bis dahin deaktivieren:

```
Get-ADUser -Filter {Name -like "Student-*"} | Disable-ADAccount
```

### Das Konto-Ablaufdatum verwalten

Konten im Active Directory können ein Ablaufdatum besitzen. Wenn ein Konto nie abläuft, kann es ohne zeitliche Einschränkungen verwendet werden. Ist ein Ablaufdatum definiert, kann das Konto nur bis zu diesem Zeitpunkt verwendet werden.

Zur Festlegung eines Konto-Ablaufdatums verwenden Sie das Cmdlet `Set-ADAccountExpiration`, zur Entfernung eines definierten Ablaufdatums das Cmdlet `Clear-ADAccountExpiration`.

Das Cmdlet `Set-ADAccountExpiration` verwendet den Parameter `-Identity`, um das gewünschte Konto (Benutzer, Computer oder Dienstkonto) auszuwählen. Dabei stehen zwei mögliche Vorgehensweisen zur Verfügung:

- ✓ Wollen Sie eine genaue Zeit angeben, wann das Konto abläuft, verwenden Sie den Parameter `-DateTime`. Als Werte des Parameters sind Datums- und/oder Uhrzeitwerte anzugeben.
- ✓ Zur Angabe einer Zeitspanne ab der Ausführung des Cmdlets verwenden Sie den Parameter `-TimeSpan`. Die gewünschte Zeitspanne geben Sie im Format `xx.xx:xx:xx` (Tage:Stunden:Minuten: Sekunden) an.

Wollen Sie ein definiertes Ablaufdatum eines Active Directory-Kontos löschen, verwenden Sie das Cmdlet `Clear-ADAccountExpires` mit dem Parameter `-Identity` zur Bezeichnung des gewünschten Objekts. Nach der Aktion läuft das entsprechende Konto nie ab.

### Beispiele

Sie wollen erreichen, dass das Konto *Praktikant-23* am 26. November 2021 abläuft.

- ```
a) Set-ADAccountExpires -Identity Praktikant-23 -DateTime "26.11.2021"
b) Set-ADAccountExpires -Identity
   "CN=Praktikant-23,OU=Praktikum,DC=herdt,DC=ps" -DateTime "26.11.2021"
```

Sie erhalten den Auftrag, das Ablaufdatum der Benutzerkonten der Mitglieder der Gruppe *HH-Mitarbeiter* auf 60 Tage nach heute festzulegen:

```
Get-ADGroupMember -Identity HH-Mitarbeiter | Where-Object {$_ .objectClass -eq "User"} | Set-ADAccountExpiration -timespan 60.00:00:00
```

Sie erfahren, dass *Praktikant-23* länger im Unternehmen bleibt als angenommen. Als Administrator sollen Sie das Ablaufdatum des Benutzerkontos entfernen:

```
Clear-ADAccountExpires -Identity Praktikant-23
```

### Suche bzw. Filtern von Konten nach definierten Kriterien

Das Cmdlet `Search-ADAccount` hilft Ihnen bei der Suche nach Benutzer-, Computer- und Dienstkonten, die von Ihnen definierten Kriterien entsprechen. Die Suchkriterien schließen den Konto- und Kennwortstatus ein.

Wenn Sie nach allgemeinen Eigenschaften von Active Directory-Konten wie abgelaufenen Konten oder abgelaufenem Kennwort, gesperrtem Konto o. Ä. suchen wollen, ist `Search-ADAccount` die richtige Wahl.

Die Suchthemen und damit die Haupt-Parameter des Cmdlets sind:

- ✓ `-AccountDisabled`  
Suche nach deaktivierten Konten
- ✓ `-AccountExpired`  
Suche nach abgelaufenen Konten
- ✓ `-AccountExpiring`  
Suche nach Konten, die zu einem bestimmten Termin oder innerhalb eines bestimmten Zeitraums ablaufen
- ✓ `-AccountInactive`  
Suche nach Konten, von denen innerhalb eines bestimmten Zeitraums oder seit einer bestimmten Dauer keine Anmeldung erfolgt ist

- ✓ -LockedOut  
Suche nach gesperrten Konten
- ✓ -PasswordExpired  
Suche nach Konten mit abgelaufenem Kennwort
- ✓ -PasswordNeverExpires  
Suche nach Konten, deren Kennwort nie abläuft

### Beispiele

Sie wollen nach dem Namen und der Objektklasse deaktivierter Konten in Ihrer Domäne suchen (wollen Sie die Suche auf einzelne Objektklassen beschränken, ergänzen Sie das Cmdlet um die Switch-Parameter `-UsersOnly` bzw. `-ComputersOnly`):

```
Search-ADAccount -AccountDisabled | Select-Object Name, objectClass
```

Sie möchten a) alle abgelaufenen Konten anzeigen und b) alle Konten, die in den nächsten 14 Tagen ablaufen, und c) alle Konten, die bis zum 27. Februar 2017 ablaufen:

- a) Search-ADAccount -AccountExpired
- b) Search-ADAccount -AccountExpiring -TimeSpan 14.00:00:00
- c) Search-ADAccount -AccountExpiring -DateTime "27.02.2022"

Sie möchten alle Konten anzeigen, deren Kennwort a) abgelaufen ist und b) nie abläuft:

- a) Search-ADAccount -PasswordExpired
- b) Search-ADAccount -PasswordNeverExpires

Sie möchten alle gesperrten Konten anzeigen:

```
Search-ADAccount -LockedOut
```

### Kennwörter für Active Directory-Konten verwalten

Mit dem Cmdlet `Set-ADAccountPassword` legen Sie ein Kennwort für ein Benutzer-, Computer- oder Dienstkonto fest. Von entscheidender Bedeutung beim Einsatz dieses Cmdlets ist der korrekte Einsatz der Parameter. Das Cmdlet bietet u. a. folgende Parameter:

| <code>Set-ADAccountPassword</code> |              |                                                                                                                                                                                                                                        |
|------------------------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Parameter                          | Typ          | Beschreibung                                                                                                                                                                                                                           |
| <b>-Identity</b>                   | <b>P (1)</b> | Angabe eines Benutzerobjekts, z. B. in Form eines definierten Namens                                                                                                                                                                   |
| -NewPassword                       | N            | Angabe des neuen Kennworts, das als verschlüsselte Zeichenfolge gespeichert wird                                                                                                                                                       |
| -OldPassword                       | N            | Angabe des bisher verwendeten Kennworts. Der Wert wird als verschlüsselte Zeichenfolge verarbeitet.                                                                                                                                    |
| -Reset                             | S            | Gibt an, dass das Kennwort zurückgesetzt werden soll. Wird dieser Parameter verwendet, müssen Sie den Parameter <code>-NewPassword</code> verwenden, das alte Kennwort mit dem Parameter <code>-OldPassword</code> aber nicht angeben. |
| ...                                |              | Vollständige Hilfe inklusive aller Parameter, Beispiele etc.:<br><code>Get-Help Set-ADAccountPassword -Full</code>                                                                                                                     |

Wichtig ist, dass die Eingabe der Kennwörter über die Parameter `-NewPassword` und `-OldPassword` in verschlüsselte Zeichenfolgen umgewandelt werden muss. Geschieht dies nicht, reagiert die PowerShell mit einer Fehlermeldung.

Die Umwandlung in eine sichere Zeichenfolge erreichen Sie z. B. durch das folgende Vorgehen:

- ✓ Sie verwenden das Cmdlet `ConvertTo-SecureString`. Als Wert des Parameters `-NewPassword` kann z. B. folgender Code verwendet werden:

```
-New-Password (ConvertTo-SecureString -AsPlainText "Kennw0rt!" -Force)
```

Das im Klartext angegebene Kennwort (Parameter `-AsPlainText`) wird in eine sichere Zeichenfolge umgewandelt und kann somit als Argument des Parameters `-NewPassword` verwendet werden. Damit der Parameter `-AsPlainText` verwendet werden kann, muss der Switch-Parameter `-Force` angegeben werden.

- ✓ Sie fordern den Anwender direkt auf, ein Kennwort einzugeben. Die Eingabe wird als verschlüsselter Wert des Parameters `-NewPassword` verwendet:

```
-New-Password (Read-Host -Prompt "Neues Kennwort eingeben" -AsSecureString)
```

Das Cmdlet `Read-Host` fordert den Benutzer zu einer Eingabe auf. Der Parameter `-Prompt` legt einen Text fest, der in der Eingabeaufforderung angezeigt wird. `-AsSecureString` sorgt dafür, dass die Benutzereingabe nur als Sternchen zu sehen ist. Darüber hinaus wird als Ergebnis eine sichere Zeichenfolge generiert (sog. *SecureString-Object; System.Security.SecureString*).

## Beispiele

Der Benutzer *Harald Herdt* hat sein Kennwort vergessen und bittet Sie, seinem Konto ein neues Kennwort zuzuweisen:

```
Set-ADAccountPassword -Identity "Harald Herdt" -NewPassword
(ConvertTo-SecureString -AsPlainText "Test123" -Force) -Reset
```

Sie möchten eine Kennwortänderung für *Harald Herdt* programmieren, die interaktiv nach dem gewünschten neuen Kennwort fragt:

```
Set-ADAccountPassword -Identity "Harald Herdt" -NewPassword (Read-Host
-Prompt "Neues Kennwort eingeben" -AsSecureString) -Reset
```

Sie möchten das Kennwort des Benutzerkontos *Harald Herdt* interaktiv ändern:

```
Set-ADAccountPassword -Identity "Harald Herdt"
```

```
Administrator: Windows PowerShell
PS C:\Users\Administrator> Set-ADAccountPassword -Identity "Harald Herdt"
Geben Sie das aktuelle Kennwort für "CN=Harald Herdt,OU=Mitarbeiter,DC=herdt,DC=ps" ein.
Kennwort: *****
Geben Sie das gewünschte Kennwort für "CN=Harald Herdt,OU=Mitarbeiter,DC=herdt,DC=ps" ein.
Kennwort: *****
Kennwort wiederholen: *****
```

## Ein Konto entsperren

Wird ein Konto in Ihrer Domäne gesperrt, kann ein Administrator mit einem PowerShell-Cmdlet (`Unlock-ADAccount`) das betroffene Konto entsperren. Entgegen der verbreiteten Meinung, ein Administrator könne auch Konten sperren, verbirgt sich dahinter ein Automatismus des Windows-Systems. Ein Konto wird automatisch gesperrt, wenn in der Kennwort-Richtlinie ein Schwellwert für die Kontosperrung festgelegt wurde. Gibt der Benutzer in dem Fall mehrfach (d. h. einmal zu viel) ein falsches Kennwort bei der Anmeldung ein, wird sein Konto gesperrt.

Wird ein Benutzerkonto gesperrt, erhält die Eigenschaft *LockedOut* des Benutzerobjekts den Wert \$True. Ist ein Objekt nicht gesperrt, lautet der Wert \$False. Zur manuellen Entsperrung des Kontos wird das Cmdlet **Unlock-ADAccount** mit dem Parameter **-Identity** zur Angabe des Kontos verwendet, das entsperrt werden soll. Je nach Einstellung wird das Konto automatisch nach einer definierten Zeitspanne wieder entsperrt.

### Beispiele

Das Benutzerkonto *Harald Herdt* ist durch zu häufige falsche Kennworteingaben bei der Anmeldung an der Domäne gesperrt worden. Sie haben die Aufgabe, das Konto wieder zu entsperren:

- a) `Unlock-ADAccount -Identity "Harald Herdt"`
- b) `Unlock-ADAccount -Identity "CN=Harald Herdt,OU=Bremen,DC=herdt,DC=ps"`

Sie erhalten die Aufgabe, alle gesperrten Konten in der Domäne zu finden und zu entsperren:

```
Search-ADAccount -LockedOut | Unlock-ADAccount
```

## 11.3 Betriebsmasterrollen mit der PowerShell verwalten

### Was sind Betriebsmaster?

In modernen Windows-Domänen bzw. -Gesamtstrukturen sind alle Domänencontroller gleichberechtigt. Alle Änderungen können auf allen Domänencontrollern durchgeführt werden. Allerdings übernehmen einzelne Domänencontroller für einige unternehmenskritische Funktionen in einer Domäne oder Gesamtstruktur die Funktion eines Betriebsmasters (*Operations Master*, verkürzt von *Flexible Single Master Operation; FSMO*).

Der jeweils dazugehörige Teil der Active Directory-Datenbank wird nur für den Betriebsmaster zum Schreiben freigegeben, auf allen Domänencontrollern gilt hier ein Schreibschutz.

Es existieren folgende fünf Betriebsmasterrollen (zwei in jeder Gesamtstruktur und drei weitere in jeder Domäne der Gesamtstruktur):

| Name                | Einmalig in ... | Bezeichnung in der PowerShell | Funktion                                                                                                                     |
|---------------------|-----------------|-------------------------------|------------------------------------------------------------------------------------------------------------------------------|
| Domänennamen-Master | Gesamtstruktur  | <i>DomainNamingMaster</i>     | Verwaltet die Domänennamen in der Gesamtstruktur                                                                             |
| Schemamaster        | Gesamtstruktur  | <i>SchemaMaster</i>           | Verwaltet Änderungen am Active Directory-Schema                                                                              |
| Infrastrukturmaster | Domäne          | <i>InfrastructureMaster</i>   | Verwaltet Objekte und Gruppenzugehörigkeiten                                                                                 |
| PDC-Emulator        | Domäne          | <i>PDCEmulator</i>            | Zuständig für Gruppenrichtlinien, Zeitsynchronisation, Kontosperrungen etc., emuliert zudem einen primären Domänencontroller |
| RID-Pool-Master     | Domäne          | <i>RIDMaster</i>              | Verwaltet SID-Pools zur eindeutigen Bezeichnung von Objekten                                                                 |

## Informationen zu den aktuellen Betriebsmastern abfragen



**Beispieldatei:** *get-fsmo.ps1* (Ordner Kapitel 11)

Welche Domänencontroller als Betriebsmaster fungieren, können Sie mit den bereits vorgestellten Cmdlets Get-ADForest und Get-ADDomain abfragen. Get-ADForest liefert Informationen über die beiden in der Gesamtstruktur einmaligen Betriebsmasterrollen *Domänennamen-Master* und *Schemamaster*. Get-ADDomain ermittelt die Betriebsmaster *Infrastrukturmastor*, *PDC-Emulator* und *RID-Pool-Master*.

Da beide Cmdlets eine Vielzahl an Informationen liefern, bietet sich eine Filterung der Informationen an.

Im Folgenden sehen Sie die Ermittlung der Betriebsmasterrollen durch ein kurzes Skript:

```
(1) Write-Host "Betriebsmasterrollen der aktuellen Gesamtstruktur:"  

(2) Get-ADForest | Format-List -Property DomainNamingMaster, SchemaMaster  

(1) Write-Host "Betriebsmasterrollen der aktuellen Domäne:"  

(2) Get-ADDomain | Format-List -Property InfrastructureMaster, PDCEmulator, RIDMaster
```

*Beispieldskript „get-fsmo.ps1“*

- ① Über das Cmdlet Write-Host erfolgen Ausgaben von Überschriften auf dem Bildschirm.
- ② Die Cmdlets Get-ADForest und Get-ADDomain ermitteln die Betriebsmasterrollen der Gesamtstruktur und der Domäne. Das Cmdlet Format-List sorgt dafür, dass die Daten als Liste ausgegeben werden. Zusätzlich wird die Anzeige durch Filterung auf die gewünschten Daten reduziert.

```
Auswählen Administrator: Windows PowerShell
PS C:\Users\Administrator\Documents\MyFunctions> .\get-fsmo.ps1
Betriebsmasterrollen der aktuellen Gesamtstruktur:

DomainNamingMaster : Server2016.herdt.ps
SchemaMaster       : Server2016.herdt.ps

Betriebsmasterrollen der aktuellen Domäne:

InfrastructureMaster : Server2016.herdt.ps
PDCEmulator          : Server2016.herdt.ps
RIDMaster             : Server2016.herdt.ps
```

## Betriebsmasterrollen auf andere Domänencontroller übertragen

Das Übertragen von Betriebsmasterrollen kann über verschiedene grafische Oberflächen, ntdsutil.exe auf der Kommandozeile und die PowerShell vorgenommen werden.

Die Verwendung der PowerShell bringt in diesem Zusammenhang einige Vorteile:

- ✓ Anders als bei der grafischen Oberfläche und ntdsutil.exe müssen Sie mit der PowerShell vor dem Übertragen keine Verbindung zum zukünftigen Rolleninhaber herstellen.
- ✓ In der PowerShell kann das Verschieben der Betriebsmasterrollen von einem beliebigen Rechner ausgeführt werden (auch Client mit RSAT).
- ✓ Die PowerShell verwendet einen einheitlichen Befehl zur Übertragung der Betriebsmasterrollen, egal ob der bisherige Rolleninhaber online oder ständig offline ist. Der Unterschied besteht nur in der Angabe eines zusätzlichen Parameters für den Fall, dass der bisherige Rolleninhaber ständig offline ist.

Für das Übertragen einer Betriebsmasterrolle auf einen anderen Domänencontroller stellt die PowerShell das Cmdlet Move-ADDdirectoryServerOperationMasterRole zur Verfügung. Das Cmdlet bietet u. a. folgende Parameter:

| <b>Move-ADDirectoryServerOperationMasterRole</b> |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------------------------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Parameter                                        | Typ          | Beschreibung                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>-Identity</b>                                 | <b>P (1)</b> | Angabe des Ziel-Domänencontrollers (Name oder definierter Name)                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>-OperationMasterRole</b>                      | <b>P (2)</b> | <p>Angabe der Rolle, die übertragen werden soll. Jede Rolle besitzt zusätzlich einen Zahlwert, der ebenfalls verwendet werden kann:</p> <ul style="list-style-type: none"> <li>✓ <i>PDCEmulator oder 0</i></li> <li>✓ <i>RIDMaster oder 1</i></li> <li>✓ <i>InfrastructureMaster oder 2</i></li> <li>✓ <i>SchemaMaster oder 3</i></li> <li>✓ <i>DomainNamingMaster oder 4</i></li> </ul> <p>Sollen mehrere Rollen gleichzeitig übertragen werden, sind Werte mit Komma zu trennen.</p> |
| <b>-Force</b>                                    | <b>S</b>     | Erzwingt die Übertragung der Betriebsmasterrolle; wird verwendet, wenn der bisherige Rolleninhaber ständig offline ist                                                                                                                                                                                                                                                                                                                                                                 |
| ...                                              |              | Vollständige Hilfe inklusive aller Parameter, Beispiele etc.:<br>Get-Help <b>Move-ADDirectoryServerOperationMasterRole</b><br>-Full                                                                                                                                                                                                                                                                                                                                                    |

### Betriebsmasterrollen übertragen, wenn der bisherige Rolleninhaber online ist

Um alle Betriebsmasterrollen zu übertragen, verwenden Sie den folgenden Befehl (Angabe der Rollen in beliebiger Reihenfolge):

Bei einzelnen Betriebsmasterrollen lautet der Befehl entsprechend:

#### Domänennamen-Master

- a) Move-ADDirectoryServerOperationMasterRole -Identity <Ziel> -OperationMasterRole DomainNamingMaster
- b) Move-ADDirectoryServerOperationMasterRole -Identity <Ziel> -OperationMasterRole 4

#### Schemamaster

- a) Move-ADDirectoryServerOperationMasterRole -Identity <Ziel> -OperationMasterRole SchemaMaster
- b) Move-ADDirectoryServerOperationMasterRole -Identity <Ziel> -OperationMasterRole 3

#### Infrastrukturmaster

- a) Move-ADDirectoryServerOperationMasterRole -Identity <Ziel> -OperationMasterRole InfrastructureMaster
- b) Move-ADDirectoryServerOperationMasterRole -Identity <Ziel> -OperationMasterRole 2

#### PDC-Emulator

- a) Move-ADDirectoryServerOperationMasterRole -Identity <Ziel> -OperationMasterRole PDCEmulator
- b) Move-ADDirectoryServerOperationMasterRole -Identity <Ziel> -OperationMasterRole 0

**RID-Pool-Master**

- ```
a) Move-ADDirectoryServerOperationMasterRole -Identity <Ziel>  
    -OperationMasterRole RIDMaster  
b) Move-ADDirectoryServerOperationMasterRole -Identity <Ziel>  
    -OperationMasterRole 1
```

**Betriebsmasterrollen übertragen, wenn der bisherige Rolleninhaber ständig offline ist**

Um eine Übertragung der Betriebsmasterrollen von dem bisherigen Rolleninhaber zu erzwingen, weil dieser ständig offline ist, verwenden Sie die oben vorgestellten Befehle. Ergänzen Sie jeweils den Parameter **-Force**. Eine Kommunikation mit dem Rolleninhaber ist nicht möglich, die Übertragung muss erzwungen werden.

Wenn ein Domänencontroller ausfällt, der eine oder mehrere Betriebsmasterrollen hält, muss nicht unbedingt sofort eine Übertragung der Betriebsmasterrollen auf einen anderen Domänencontroller vorgenommen werden. Falls Sie sich doch dafür entscheiden, sorgen Sie dafür, dass der alte Rolleninhaber nicht mehr in der Domäne online geht. In diesem Fall rät Microsoft zu einer Neuinstallation des Betriebssystems, einer Neuaufnahme in der Domäne und gegebenenfalls zum abschließenden Hochstufen zum Domänencontroller.



## 11.4 Kurz zusammengefasst

Die PowerShell bietet auch in den Randthemen des Active Directory ausgearbeitete und mächtige Cmdlets. Durch ihren Namen ist ihr Einsatzzweck gut zu bestimmen, was manchmal allerdings zu recht sperrigen Cmdlet-Bezeichnungen führt.

Fein abgestimmte Kennwortrichtlinien können komplett mit der PowerShell verwaltet werden. Ein eigener Befehlssatz verwaltet die Kennwort-Richtlinienobjekte selbst, ein weiterer Befehlssatz kümmert sich um die Zuordnung der Richtlinienobjekte zu Gruppen und/oder Benutzern. Besonders nützlich in diesem Zusammenhang ist das Cmdlet **Get-ADUserResultantPasswordPolicy**, mit dem Sie die effektive Richtlinie für einen bestimmten Benutzer ermitteln können.

Im Bereich der Active Directory-Konten bietet die PowerShell weitaus mehr als die Cmdlets, die Sie im letzten Kapitel kennengelernt haben. Eine Anzahl weiterer Cmdlets kümmert sich um die allgemeinen Einstellungen von Konten, unabhängig von der Objektklasse des Kontos. Auch wenn es anfangs verwirrend sein mag, zu welchem Zweck man welches Cmdlet einsetzt, ist die Aufgabenteilung bei intensiver Beschäftigung mit diesen Cmdlets augenfällig.

Schließlich können Sie die PowerShell auch für Aufgaben rund um die Betriebsmasterrollen einsetzen. Mit wenigen Befehlen erreichen Sie das gewünschte Ziel einfacher als mit allen anderen verfügbaren Tools. Mit kleinen Skripten ist das Auslesen oder Übertragen der Betriebsmasterrollen schnell erledigt.

## 11.5 Übung

**Weitere Aufgaben im Active Directory mit der PowerShell erledigen****Übungsdatei:** –**Ergebnisdatei:** **11\_ergebnisse.txt**

1. Erstellen Sie zur Vorbereitung der Übung – gern mithilfe der PowerShell:
  - ✓ die Benutzerin *Corinna Cremer*, die Mitglied der globalen Sicherheitsgruppe *GG\_Forschung* werden soll
  - ✓ den Benutzer *Willi Wagner*, der Mitglied der globalen Sicherheitsgruppe *GG\_Chefetage* werden soll

- a) Erstellen Sie eine neue fein abgestimmte Kennwortrichtlinie *PSO\_Forschung* mit folgenden Einstellungen:
    - ✓ Prioritätswert: 100
    - ✓ Maximales Kennwortalter: 21 Tage
    - ✓ Minimales Kennwortalter: 5 Tage
    - ✓ Minimale Kennwortlänge: 14 Zeichen
    - ✓ Komplexes Kennwort erforderlich
    - ✓ Kennwortsperrungsschwelle: 3 falsche Kennwörter
    - ✓ Zurücksetzungsdauer des Kennwortzählers: 3 Stunden
    - ✓ Kontosperrdauer: 12 Stunden
  - b) Erstellen Sie eine neue fein abgestimmte Kennwortrichtlinie *PSO\_Chefetage* mit folgenden Einstellungen:
    - ✓ Prioritätswert: 200
    - ✓ Maximales Kennwortalter: 50 Tage
    - ✓ Minimales Kennwortalter: 1 Tage
    - ✓ Minimale Kennwortlänge: 6 Zeichen
    - ✓ Komplexes Kennwort erforderlich
    - ✓ Kennwortsperrungsschwelle: nicht aktiviert
  - c) Verknüpfen Sie das Kennwortrichtlinienobjekt *PSO\_Forschung* mit der Benutzerin *Corinna Cremer* und der Gruppe *GG\_Chefetage*.
  - d) Verknüpfen Sie das Kennwortrichtlinienobjekt *PSO\_Chefetage* mit dem Benutzer *Willi Wagner*.
  - e) Fügen Sie dem Kennwortrichtlinienobjekt *PSO\_Forschung* eine Beschreibung *Richtlinie für die Forschungsabteilung* hinzu.
  - f) Erzeugen Sie eine Ausgabe, in der Sie die Werte der Parameter für Name, Prioritätswert, minimale Kennwortlänge und maximales Kennwortalter für genau die beiden Kennwortrichtlinienobjekte *PSO\_Forschung* und *PSO\_Chefetage* tabellarisch anzeigen lassen.
  - g) Finden Sie heraus, welches Kennwortrichtlinienobjekt effektiv auf den Benutzer *Willi Wagner* angewendet wird. Warum ist es diese Richtlinie, die Anwendung findet?
  - h) Verknüpfen Sie den Benutzer *Willi Wagner* auch mit dem Kennwortrichtlinienobjekt *PSO\_Forschung*. Welches Richtlinienobjekt wird nun auf ihn effektiv angewendet?
2. Suchen Sie alle Konten in Ihrer Domäne, die ...
    - a) gesperrt sind.
    - b) ein Ablaufdatum bis zum 31.12.2016 haben.
    - c) seit 14 Tagen keine Anmeldung vorgenommen haben.
  3.
    - a) Sie wollen für die Benutzerin *Corinna Cremer* ein neues Kennwort vergeben. Fragen Sie interaktiv nach dem neuen Kennwort. (Beachten Sie die Auswirkungen von Übung 1.)
    - b) Sorgen Sie dafür, dass das Benutzerkonto von *Willi Wagner* in sechs Wochen abläuft.
    - c) Deaktivieren Sie das Konto von *Corinna Cremer*. Zeigen Sie dann die Eigenschaft ihres Kontos an, die anzeigt, dass das Konto deaktiviert ist (*Enabled*). Aktivieren Sie schließlich wieder ihr Konto.
  4.
    - a) Lesen Sie die drei Betriebsmasterrollen Ihrer Domäne aus.
    - b) Zeigen Sie die beiden Betriebsmasterrollen Ihrer Gesamtstruktur an.
    - c) Übertragen Sie die Betriebsmasterrolle *PDC-Emulator* auf einen anderen Domänencontroller (*Server2*). Sollten Sie mit nur einem Domänencontroller arbeiten, setzen Sie als Zielrechner Ihren eigenen Rechner (*Server1*) ein. Eine Erzwingung ist nicht nötig.

# 12 Active Directory-Papierkorb



Beispieldatei: 12\_kompletter code.txt (Ordner Kapitel 12)

## 12.1 Das Prinzip des Active Directory-Papierkorbs

Der Active Directory-Papierkorb dient dazu, gelöschte Objekte im Active Directory vollständig wieder herstellen zu können. Das Feature wurde mit Windows Server 2008 R2 eingeführt. Davor erlaubte die sogenannte *Reanimierung von Tombstones* nur die Wiederherstellung der wichtigsten Objektinformationen wie der eindeutigen ID des Objekts (*objectSID*). Viele andere Attribute und z. B. Gruppenmitgliedschaften bleiben verloren und mussten mühsam anderweitig wiederhergestellt werden.

Damit der Active Directory-Papierkorb aktiviert werden kann, ist Folgendes zu bedenken:

- ✓ Die Gesamtstruktur muss mindestens im Funktionsmodus *Windows Server 2008 R2* betrieben werden. Daraus folgt, dass die Betriebssysteme aller Domänencontroller der Gesamtstruktur – also auch jeder enthaltenen Domäne – mindestens mit dem Betriebssystem Windows Server 2008 R2 oder höher (neuer) betrieben werden muss.
- ✓ Der Active Directory-Papierkorb wird für die komplette Gesamtstruktur aktiviert. Eine Aktivierung für ausgewählte Domänen Ihrer Gesamtstruktur ist nicht möglich. Die Aktivierung muss auf dem Server vorgenommen werden, der die Betriebsmasterrolle Domänennamen-Master (*Domain Naming Master*) hält.
- ✓ Ist der Active Directory-Papierkorb einmal aktiviert, lässt er sich nicht wieder deaktivieren.
- ✓ Damit die Aktivierung durchgeführt werden kann, sind von Benutzerseite ausreichende Berechtigungen vorausgesetzt. Führen Sie diese Arbeiten z. B. als Organisations-Administrator durch.

Auch wenn ab Windows Server 2012 eine grafische Oberfläche für die Aktivierung des Active Directory-Papierkorbs verwendet werden kann, erfolgt die Ausführung dieser Aktionen systemintern mithilfe von PowerShell-Befehlen.

Im Folgenden sehen Sie die Schritte rund um die Einrichtung und Arbeit mit dem Active Directory-Papierkorb – unter Zuhilfenahme von PowerShell-Befehlen.

## 12.2 Gesamtstrukturfunktionsebene ermitteln und ggf. ändern

Das Cmdlet `Get-ADForest` steht zur Verfügung, um detaillierte Informationen zu Ihrer Gesamtstruktur zu ermitteln.

```
Get-ADForest -Identity herdt.ps
```

Mit dem Cmdlet erhalten Sie u. a. Informationen über die Domänen Ihrer Gesamtstruktur, Betriebsmasterrollen, Stammdomäne und Funktionsebene. Der Wert des Parameters `-Identity` bestimmt die Gesamtstruktur, zu der die Informationen gesammelt werden. Eine verkürzte Schreibweise ohne Parameter liefert identische Ergebnisse, sofern Informationen zur aktuellen Gesamtstruktur gesammelt werden sollen.

Wollen Sie nur die Funktionsebene der Gesamtstruktur anzeigen, benötigen Sie nur den Wert der Eigenschaft `ForestMode`, den Sie mit einem leicht modifizierten Befehl erhalten:

```
(Get-ADForest -Identity herdt.ps).ForestMode
```

Zur Ermittlung des Wertes der Eigenschaft *ForestMode* ist es nötig, den Befehl in Klammern zu setzen, der das Objekt mit der gewünschten Eigenschaft liefert. Dadurch wird `Get-ADForest` zuerst ausgeführt und erst danach der Wert der Eigenschaft *ForestMode* des bereits gelieferten Objekts abgefragt. Wird die Klammerung weggelassen, erhalten Sie eine Fehlermeldung.

Mögliche Rückgabewerte sind:

- ✓ `Windows2000Forest` (Zahlwert: 0),
- ✓ `Windows2003InterimForest` (Zahlwert: 1),
- ✓ `Windows2003Forest` (Zahlwert: 2),
- ✓ `Windows2008Forest` (Zahlwert 3),
- ✓ `Windows2008R2Forest` (Zahlwert: 4),
- ✓ `Windows2012Forest` (Zahlwert: 5),
- ✓ `Windows2012R2Forest` (Zahlwert: 6),
- ✓ `Windows2016Forest` oder neuer (Zahlwert: 7).

Um die Voraussetzung zu erfüllen, für die Aktivierung des Active Directory-Papierkorbs die Gesamtstruktur mindestens in der Funktionsebene `Windows2008R2Forest` zu betreiben, verwenden Sie das Cmdlet `Set-ADForestMode`.

Um die Funktionsebene der Gesamtstruktur auf den geforderten Wert hochzustufen, geben Sie den folgenden Befehl ein:

```
Set-ADForestMode -ForestMode Windows2008R2Forest -Identity herdt.ps
-Confirm:$False
```

- ✓ Mit dem Parameter `-ForestMode` legen Sie die Funktionsebene gemäß den oben angegebenen Werten fest. Alternativ können Sie auch die Zahlwerte verwenden.
- ✓ Als Wert des Parameters `-Identity` geben Sie den Namen der Gesamtstruktur ein, deren Funktionsebene Sie hochstufen wollen.
- ✓ Der Parameter `-Confirm` mit dem angegebenen Wert `$False` sorgt dafür, dass Sie die übliche Bestätigungsfrage zur Ausführung der Aktion nicht erhalten.



„Funktionsebenen können nur hochgestuft werden, eine nachträgliche Herabstufung ist nicht möglich.“ Dieser Satz gilt für alle grafischen Verwaltungstools, nicht aber für die PowerShell. Mit dem PowerShell-Cmdlet `Set-ADForestMode` können Sie frei zwischen den Funktionsebenen `Windows2012R2Forest`, `Windows2012Forest` und `Windows2008R2Forest` wechseln, wenn Sie das Betriebssystem Windows Server 2008 R2 oder höher verwenden. Haben Sie den Active Directory-Papierkorb (noch) nicht aktiviert, können Sie Ihre Gesamtstruktur-funktionsebene sogar auf `Windows2008Forest` herabstufen.

Die Zahlwerte für die Funktionsebenen können die Programmierung von Skripten vereinfachen:

```
(1) If ((Get-ADForest -Identity herdt.ps).ForestMode -gt 3)
{
(2)     Write-Host "Die Funktionsebene ist bereits auf Windows 2008 R2 oder
höher. Es sind keine weiteren Aktionen nötig."
}
(3) Else
{
    Set-ADForestMode -ForestMode Windows2008R2Forest -Identity herdt.ps
    -Confirm:$False
}
```

- ① In einer If-Anweisung wird die Funktionsebene der Gesamtstruktur `herdt.ps` abgefragt. Im Beispiel wird für einen einfachen Vergleich mit dem Zahlwert der Funktionsebene gearbeitet. Übersetzt bedeutet die Bedingung: „Ist die Funktionsebene 2008R2 oder höher?“ Die Abfrage ist mit einem Vergleich von Zahlwerten einfacher zu programmieren, da gleich mehrere Funktionsebenen adressiert werden können (z. B. `-gt 3` steht für die Ebenen 4, 5, 6 und 7).
- ② Trifft die Bedingung zu, wird eine Meldung ausgegeben.
- ③ Falls die Funktionsebene nicht den Vorgaben entspricht, erfolgt mithilfe des Cmdlets `Set-ADForestMode` eine Hochstufung auf `Windows2008R2Forest`.

## 12.3 Aktivieren des Active Directory-Papierkorbs

Nachdem Sie die Gesamtstrukturfunktionsebene auf Windows Server 2008 R2 oder höher festgelegt haben, können Sie den Active Directory-Papierkorb aktivieren. Die PowerShell stellt für diese Aufgabe das Cmdlet `Enable-ADOptionalFeature` zur Verfügung.

Zur Arbeit mit optionalen Features stellt die PowerShell drei Cmdlets zur Verfügung. Alle beinhalten das Substantiv `ADOptionalFeature`:

Cmdlet	Erläuterung
<code>Get-ADOptionalFeature</code>	Zeigt verfügbare optionale Features im Active Directory an
<code>Enable-ADOptionalFeature</code>	Aktiviert ein optionales Feature im Active Directory
<code>Disable-ADOptionalFeature</code>	Deaktiviert ein optionales Feature im Active Directory

Zur Anzeige aller verfügbaren optionalen Features in der Gesamtstruktur verwenden Sie den folgenden Befehl:

```
Get-ADOptionalFeature -Filter *
```

Sie erhalten eine ähnliche Ausgabe wie in den folgenden Zeilen:

```
DistinguishedName : CN=Recycle Bin Feature, CN=Optional Features,
                    CN=Directory Service, CN=Windows NT, CN=Services,
                    CN=Configuration, DC=herdt, DC=ps
EnabledScopes    : {}
FeatureGUID      : 766ddcd8-acd0-445e-f3b9-a7f9b6744f2a
FeatureScope     : {ForestOrConfigurationSet}
IsDisableable    : False
Name              : Recycle Bin Feature
ObjectClass       : msDS-OptionalFeature
ObjectGUID        : fcb1716a-03a8-4c31-a952-6180421f7966
RequiredDomainMode :
RequiredForestMode : Windows2008R2Forest
```

Sie erhalten Informationen zum Active Directory-Papierkorb (*Recycle Bin Feature*) und zu einem neuen Sicherheitsfeature von Windows Server 2016 (*Privileged Access Management*). Microsoft hat zwar mit den `ADOptionalFeature`-Cmdlets eine Struktur für optionale Features aufgebaut, vor Windows Server 2016 gab es allerdings nur ein einziges optionales Feature: den Active Directory-Papierkorb. Nun sind es zwei.

Zur Aktivierung des Active Directory-Papierkorbs reicht eine weitere PowerShell-Befehlszeile aus:

```
Enable-ADOptionalFeature -Identity "Recycle Bin Feature" -Scope
ForestOrConfigurationSet -Target herdt.ps
```

- ✓ Als Wert des Parameters `-Identity` ist das zu installierende optionale Feature anzugeben. Alternativ zur obigen Zeichenfolge `Recycle Bin Feature` können Sie den definierten Namen des Features eingeben. Beide Eingaben führen zum identischen Ergebnis.
- ✓ Der Parameter `-Scope` legt den Bereich fest, in dem das Feature aktiviert werden soll. Hier ist der Wert `ForestOrConfigurationSet` zu verwenden.
- ✓ Als Wert des Parameters `-Target` wird die Domäne oder Gesamtstruktur festgelegt, in der das optionale Feature aktiviert wird. In diesem Fall wird die Gesamtstruktur angegeben.

Nach der Aktivierung des Active Directory-Papierkorbs können Sie erneut mit dem Befehl `Get-ADOptionalFeature -Filter *` den Installationsstand der optionalen Features überprüfen. Die Ausgabe ändert sich nur in einem Punkt im Vergleich zur Ausgabe vor der Installation:

```
EnabledScopes: {CN=Partitions, CN=Configuration,DC=herdt,DC=ps,
  CN=NTDS Settings, CN=SERVER2016, CN=Servers,
  CN=Default-First-Site-Name, CN=Sites, CN=Configuration,
  DC=herdt,DC=ps}
```

Damit ist der Active Directory-Papierkorb installiert und zur Verwendung bereit.

Ein weiteres Cmdlet – `Disable-ADOptionalFeature` – ist für die Deaktivierung optionaler Features vorgesehen. Allerdings haben Sie in der obigen Ausgabe für den Active Directory-Papierkorb die Information `IsDisableable: False` erhalten. Da auch das zweite, neue Feature `Privileged Access Management` nicht deaktiviert werden kann, steht das Cmdlet nur für zukünftige optionale Features zur Verfügung.

## 12.4 Der Active Directory-Papierkorb im Active Directory-Verwaltungscenter

Gelöschte Active Directory-Objekte werden in den Container *Deleted Objects* verschoben. Für die Stammdomäne `herdt.ps` der Gesamtstruktur z. B. lautet der definierte Name dieses Containers `CN=Deleted Objects, DC=herdt, DC=ps`.

Standardmäßig wird der Container nicht angezeigt. Bei aktiviertem Active Directory-Papierkorb wird der Container im Active Directory-Verwaltungscenter eingeblendet, wie Sie in der nachfolgenden Abbildung erkennen können.

The screenshot shows the Active Directory-Verwaltungscenter interface. The left navigation pane shows 'herdt (lokal)' is selected. The main pane displays a table of objects under 'herdt (lokal) (15)'. The 'Deleted Objects' container is highlighted with a red border. The table columns are 'Name', 'Typ', and 'Beschreibung'. The 'Deleted Objects' entry is described as 'Default container for deleted objects'.

Name	Typ	Beschreibung
Builtin	builtinDom...	
Computers	Container	Default container for upgraded com...
<b>Deleted Objects</b>	Container	<b>Default container for deleted objects</b>
Domain Controllers	Organisati...	Default container for domain contro...

Active Directory-Verwaltungscenter mit aktiviertem Active Directory-Papierkorb

Falls Sie mithilfe von grafischen Verwaltungstools den Active Directory-Papierkorb zur Wiederherstellung gelöschter Objekte einsetzen wollen, verwenden Sie das Active Directory-Verwaltungszentrum und öffnen den Container *Deleted Objects*. Gelöschte Objekte im Container werden angezeigt und können über die Aufgabenleiste oder über einen Rechtsklick wiederhergestellt werden.

Name	Wenn gelöscht	Letzte bekannt...	Typ
Testuser	8/10/2016 3:39...	OU=Mitarbeiter...	Benutzer

Anzeige eines gelöschten Objekts im Container „Deleted Objects“ (bei gedrückter rechter Maustaste)

Ihre Aktionen im Active Directory-Verwaltungszentrum werden automatisch in PowerShell-Befehle umgesetzt. Durch eine direkte Eingabe von PowerShell-Befehlen ist eine Wiederherstellung gelöschter Objekte gleichermaßen möglich, durch zusätzliche Möglichkeiten von Skripting und Automatisierung sogar flexibler.

## 12.5 Gelöschte Objekte mit der PowerShell finden

Um das gelöschte Objekt zu finden, das Sie wiederherstellen möchten, verwenden Sie das Cmdlet Get-ADObject:

```
Get-ADObject -Filter {<String>} -IncludeDeletedObjects
```

- ✓ Sie verwenden das Cmdlet mit einer Filterung nach dem Objekt, das Sie suchen. Die Bedingung, nach der gefiltert wird, wird in geschweiften Klammern formuliert. Beispiele für Filter sind u. a.:
  - ✓ Filter {displayName -eq "Petra Muster"}
  - ✓ Filter {SamAccountName -eq "petra.muster"}
  - ✓ Filter {Name -like "\*pe\*"}
  - ✓ Filter {cn -eq "Client4"}
- Die Mechanismen sind Ihnen bereits aus den vorigen Kapiteln vertraut. Filtern Sie danach, was vom gelöschten Objekt bekannt ist. Bei der Suche nach den richtigen Attributbezeichnungen hilft der Attribut-Editor, der als Registerkarte in den Objekteigenschaften zur Verfügung steht.
- ✓ Um gelöschte Objekte zu finden, muss zwingend der Switch-Parameter -IncludeDeletedObjects angegeben werden. Fehlt der Parameter, werden gelöschte Objekte ignoriert.

Möchten Sie eine Liste aller gelöschten Objekte anzeigen, verwenden Sie den folgenden Befehl:

```
Get-ADObject -Filter * -IncludeDeletedObjects -SearchBase  
"CN=Deleted Objects, DC=herdt, DC=ps"
```

- ✓ Sie filtern in diesem Beispiel nicht nach bestimmten Objekten (-Filter \*).
- ✓ Da Sie nach gelöschten Objekten suchen, verwenden Sie den Parameter -IncludeDeletedObjects.
- ✓ Um ausschließlich gelöschte Objekte anzuzeigen, beschränken Sie die Suche mithilfe des Parameters -SearchBase auf den Container *Deleted Objects*.

Bei der Suche nach gelöschten Objekten können Sie nicht Get-ADUser, Get-ADComputer oder Get-ADGroup, sondern nur das allgemeinere, übergeordnete Cmdlet Get-ADObject verwenden, das alle Objekttypen abfragt. Nur dieses Cmdlet bietet den Switch-Parameter -IncludeDeletedObjects, ohne den keine gelöschten Objekte angezeigt werden.

Wollen Sie zusätzlich herausfinden, in welchem Container bzw. in welcher Organisationseinheit sich das gelöschte Objekt befunden hat, bevor es gelöscht wurde, ergänzen Sie den Befehl um den Parameter `-Properties` mit dem Wert `LastKnownParent`.

```
Get-ADObject -Filter {SamAccountName -eq "petra.muster"}  
-IncludeDeletedObjects -Properties LastKnownParent
```

- ✓ Befand sich das gesuchte Objekt im Container *Users*, erhalten Sie folgende Ausgabe:  
`LastKnownParent: CN=Users,DC=herdt,DC=ps.`
- ✓ Befand sich das gesuchte Objekt z. B. in einer Organisationseinheit (hier: *Meine Benutzer*), die ebenfalls gelöscht wurde, erhalten Sie folgende Ausgabe:  
`LastKnownParent: OU=Meine Benutzer\0ADEL:174f51eb-2a03-41b9-8dc1-b4e7fe5eca01,CN=Deleted Objects,DC=herdt,DC=ps`

Diese Informationen sind wichtig, da es nicht möglich ist, ein Objekt wiederherzustellen, dessen übergeordnetes Element nicht mehr existiert. In diesem Fall muss zuerst die Organisationseinheit wiederhergestellt werden, bevor das Objekt wiederhergestellt werden kann. Alternativ kann die Wiederherstellung des Objekts an einen anderen Ort erfolgen.

## 12.6 Gelöschte Objekte mit der PowerShell wiederherstellen

Nachdem Sie mithilfe des Cmdlets `Get-ADObject` und der beschriebenen Parameter das Objekt gefunden haben, das Sie wiederherstellen möchten, fehlt nur noch ein letzter Schritt: Sie senden das gefundene Objekt über die Pipeline an das Cmdlet `Restore-ADObject` ohne Angabe eines Parameters.

Die allgemeine Syntax für diesen Vorgang lautet:

```
Get-ADObject -Filter {<String>} -IncludeDeletedObjects | Restore-ADObject
```

### Beispiel

Sie haben das Benutzerkonto von *Hans* aus der Organisationseinheit *Meine Benutzer* versehentlich gelöscht und möchten es wiederherstellen:

```
Get-ADObject -Filter {givenName -eq "Hans"} -IncludeDeletedObjects |  
Restore-ADObject
```

- ✓ Damit wird das Objekt, auf das der Filter zutrifft, am Originalort wiederhergestellt.
- ✓ Trifft der Filter auf mehrere Objekte zu, werden alle Objekte wiederhergestellt.
- ✓ Falls Sie sich nicht sicher sind, ob Sie damit auch Objekte wiederherstellen, die Sie nicht wiederherstellen wollen, lassen Sie zum Testen das Cmdlet `Restore-ADObject` weg oder ergänzen es um den Parameter `-WhatIf`.

## 12.7 Was tun, wenn auch das übergeordnete Objekt gelöscht wurde?

Wurde das übergeordnete Objekt (also die Organisationseinheit) ebenfalls gelöscht, in der sich das gelöschte Objekt befunden hat, bricht der Befehl mit einer entsprechenden Fehlermeldung ab.

Sie können in dem Fall das Objekt an einem anderen Ort wiederherstellen, z. B. nachfolgend in den Container *Users*:

```
Get-ADObject -Filter {givenName -eq "Hans"} -IncludeDeletedObjects |  
Restore-ADObject -TargetPath "cn=Users,dc=herdt,dc=ps"
```

Soll das gelöschte Objekt am ebenfalls gelöschten Originalort wiederhergestellt werden, müssen Sie zuerst das übergeordnete Objekt, also die Organisationseinheit, wiederherstellen.

Dies erreichen Sie durch die bekannten Abfragen, die Sie um die Filterung nach Organisationseinheiten ergänzen:

```
Get-ADObject -Filter * -SearchBase "cn=Deleted Objects,dc=herdt,dc=ps"  
-IncludeDeletedObjects | Where-Object {$_.objectClass -eq  
"OrganizationalUnit"}
```

- ✓ Mit diesem Befehl suchen Sie ausschließlich nach gelöschten Objekten im Container *Deleted Objects*.
- ✓ Das Cmdlet *Where-Object* filtert über die angegebene Bedingung nach Organisationseinheiten. Handelt es sich bei dem übergebenen Objekt um eine Organisationseinheit, ist dies über die Eigenschaft *objectClass* des Objekts an einem entsprechenden Wert zu erkennen.

Mit dieser Methode werden alle gelöschten Organisationseinheiten ausgegeben. Suchen Sie nur eine bestimmte Organisationseinheit, erweitern Sie die Bedingung um die Angabe des Namens der Organisationseinheit, z. B. *Meine Benutzer*:

```
<...> | Where-Object {$_.objectClass -eq "OrganizationalUnit" -and $_.Name  
-like "Meine Benutzer*"}
```

Es werden zwei Bedingungen geprüft. Neben der Prüfung, ob es sich bei dem Objekt um eine Organisationseinheit handelt, wird zusätzlich der Name des Objekts überprüft.

Bei der Prüfung des Namens beachten Sie, dass sich der Name durch die Löschung verändert hat. Mit einer Prüfung der Bedingung *\$\_.Name -eq "Meine Benutzer"* wird das gewünschte Objekt nicht gefunden. Der Name des Objekts lautet nun im Beispiel *Meine Benutzer\OADEL:174f51eb-2a03-41b9-8dc1-b4e7fe5eca01*.



Der Aufwand zur Auswahl der Objekte ist gerechtfertigt. Haben Sie schließlich eine Abfrage formuliert, die ausschließlich das gewünschte Ergebnis mit einem oder mehreren Objekten liefert, erweitern Sie den Befehl nochmals:

```
<...> | Restore-ADObject
```

Alle Objekte, die den Kriterien entsprechen, werden wiederhergestellt.

Sind ganze Strukturen gelöscht worden, müssen Sie Folgendes beachten:

Sie müssen die Wiederherstellung von Objekten unbedingt bei der höchsten Hierarchieebene beginnen, weil gelöschte Objekte nur in einem vorhandenen übergeordneten Objekt wiederhergestellt werden können. Sie müssen also z. B. zuerst eine Organisationseinheit wiederherstellen und erst danach die dort enthaltenen Objekte.



## Organisationseinheit gelöscht: Welche Objekte befanden sich darin?

Stellen Sie sich vor, Ihr Kollege hat versehentlich eine komplette Organisationseinheit *Hamburg* gelöscht. Die Organisationseinheit ist schnell wiederhergestellt:

```
Get-ADObject -Filter { (objectClass -eq "OrganizationalUnit") -and
  (Name -like "Bremen*") } -IncludeDeletedObjects | Restore-ADObject
```

Im nächsten Schritt sollen die Objekte wiederhergestellt werden, die sich darin befanden. Doch welche waren das? Hier hilft eine spezielle Bedingung (Parameter `-Filter`), die nach einem angegebenen übergeordneten Objekt gelöschter Objekte filtert:

```
Get-ADObject -SearchBase "CN=Deleted Objects,DC=herdt,DC=ps"
  -IncludeDeletedObjects -Filter { LastKnownParent -eq
    "OU=Hamburg,DC=herdt,DC=ps" }
```

Mit diesem Befehl finden Sie alle Objekte, die sich vor dem Löschen in der Organisationseinheit *Hamburg* befunden haben. Wollen Sie die Angabe explizit in der Ausgabe sehen, ergänzen Sie den Befehl um einen weiteren Parameter:

```
Get-ADObject -SearchBase "CN=Deleted Objects,DC=herdt,DC=ps"
  -IncludeDeletedObjects -Filter { LastKnownParent -eq "OU=Hamburg, DC=herdt,
  DC=ps" } -Properties LastKnownParent
```

Durch den Parameter `-Properties` wird die Anzeige der gelöschten Objekte um eine Zeile ergänzt, die das übergeordnete Objekt (*LastKnownParent*) enthält.



Sollten Sie an dieser Stelle eine leere Ausgabe erhalten, war entweder die gelöschte Organisationseinheit leer oder ist noch nicht wiederhergestellt worden.

Nun werden die gefundenen Objekte der Organisationseinheit wiederhergestellt, indem Sie die letzte Eingabezeile um das Cmdlet `Restore-ADObject` erweitern:

```
Get-ADObject -SearchBase "CN=Deleted Objects,DC=herdt,DC=ps"
  -IncludeDeletedObjects -Filter { LastKnownParent -eq "OU=Hamburg, DC=herdt,
  DC=ps" } -Properties LastKnownParent | Restore-ADObject
```

## 12.8 Kurz zusammengefasst

Lange hatten Administratoren auf umfassende Werkzeuge zur vollständigen Wiederherstellung gelöschter Objekte gewartet. Seit Windows Server 2012 stehen nunmehr gleich zwei Werkzeuge zur Erfüllung dieser Aufgabe bereit: das Active Directory-Verwaltungscenter mit einer grafischen Oberfläche und die PowerShell für die Verwendung des Active Directory-Papierkorbs mittels eingegebener Befehle.

Der Active Directory-Papierkorb setzt eine Gesamtstrukturfunktionsebene von mindestens *Windows Server 2008 R2* voraus, er gilt gesamtstrukturweit und lässt sich nicht wieder deaktivieren.

Einmal aktiviert, ist die Verwendung des Papierkorbs mit wenigen Befehlen möglich: In der Regel sucht man das gewünschte gelöschte Objekt und stellt es anschließend wieder her. Bei gelöschten Strukturen muss man bei der Wiederherstellung mit der höchsten Hierarchieebene beginnen, da gelöschte Objekte nur in vorhandenen übergeordneten Objekten wiederhergestellt werden können.

Mit ein wenig Übung zählt der Active Directory-Papierkorb zu den unverzichtbarsten Tools innerhalb des Active Directory. Knackpunkt im Livebetrieb ist manchmal noch die geforderte Gesamtstrukturfunktionsebene.

## 12.9 Übung

### Mit Modulen arbeiten

Übungsdatei: –

Ergebnisdatei: 12\_ergebnisse.txt

1. Prüfen Sie die Funktionsebene Ihrer Gesamtstruktur. Falls die Funktionsebene kleiner als Windows Server 2008 R2 sein sollte, stufen Sie sie auf diese Stufe herauf.
2. Ist es möglich, nach Heraufstufung der Gesamtstrukturfunktionsebene aus Übung 1 die Funktionsebene wieder herabzustufen? Falls ja:
  - a) Welche Werte sind möglich?
  - b) Welche Voraussetzungen müssen erfüllt werden?
  - c) Wie lautet der Befehl für diese Aktion?
3.
  - a) Aktivieren Sie den Active Directory-Papierkorb für Ihre Gesamtstruktur.
  - b) Überprüfen Sie, ob der Active Directory-Papierkorb aktiviert wurde.
  - c) Mit welchem Cmdlet können Sie ein optionales Feature deaktivieren? Würde das bei dem Active Directory-Papierkorb funktionieren?
4.
  - a) Welches Cmdlet setzen Sie ein, um gelöschte Objekte zu finden?
  - b) Welche Auswirkungen hat es, wenn Sie bei der Suche nach gelöschten Objekten den Parameter `-IncludeDeletedObjects` weglassen?
5. Erstellen Sie direkt in Ihrer Stammdomäne eine Organisationseinheit *Anwender*. In dieser Organisationseinheit erstellen Sie zwei Benutzer *Paul English* und *Mary English*. Löschen Sie die Organisationseinheit inklusive beider Benutzer.
  - a) Suchen Sie mit einem möglichst einfachen Befehl nach beiden Benutzern.
  - b) Suchen Sie nach der gelöschten Benutzerin *Mary English*.
  - c) Suchen Sie nach gelöschten Organisationseinheiten.
  - d) Suchen Sie gezielt nach der gelöschten Organisationseinheit *Anwender*.
  - e) Gibt es etwas zu bedenken, wenn Sie alle drei gelöschten Objekte wiederherstellen möchten?
  - f) Stellen Sie alle gelöschten Objekte wieder her.

# 13 Sonderaufgaben für die PowerShell



Beispieldatei: 13\_kompletter code.txt (Ordner Kapitel 13)

## 13.1 Geplante Aufgaben mit der PowerShell verwalten

Ein wichtiger Bestandteil jeder Windows-Installation ist die Aufgabenplanung (*task scheduler*), die Vorgänge zur automatischen Ausführung nach Zeitplan oder Erfüllen einer definierten Bedingung plant. So werden z. B. regelmäßige Backups oder Updates über Windows Update über die Aufgabenplanung gesteuert.

Sie finden in Windows die Aufgabenplanung als Bestandteil der Verwaltung in der Systemsteuerung. Alternativ können Sie die Aufgabenplanung – auch aus der PowerShell heraus – durch Eingabe von *taskschd.msc* starten.

Die Arbeit mit der PowerShell ist häufig effektiver als sich durch weitverzweigte Baumstrukturen in grafischen Oberflächen durchzuklicken. Für die Arbeit mit der Aufgabenplanung bietet die PowerShell ein eigenes Modul namens *ScheduledTasks* mit etwa 20 Cmdlets.

Um sich einen ersten Überblick über die bestehenden Aufgaben (*tasks*) zu verschaffen, verwenden Sie das Cmdlet *Get-ScheduledTask*, wie in den folgenden Beispielen zu sehen:

- a) `Get-ScheduledTask`
- b) `Get-ScheduledTask | Where-Object {$_ .state -eq "Running"}`
- c) `Get-ScheduledTask -TaskName *update*`
- d) `Get-ScheduledTask -TaskPath *update*`

- ✓ Beispiel a) liefert eine Liste der auf dem Rechner vorhandenen Aufgaben inklusive des aktuellen Status (*ready, running, disabled*).
- ✓ In Beispiel b) wird das Cmdlet zusätzlich gefiltert und nur laufende Aufgaben werden angezeigt.
- ✓ In den Beispielen c) und d) werden die Ergebnisse auf einen angegebenen Namen (-TaskName) bzw. Pfad (-TaskPath) begrenzt, wobei Platzhalterzeichen verwendet werden können.

Wenn Sie Detail-Informationen zu einer bestimmten Aufgabe abrufen wollen, verwenden Sie das Cmdlet *Get-ScheduledTaskInfo*. Wenn sich auf dem Rechner z. B. mit dem Sicherungszeitplan der Windows Server-Sicherung eine wiederkehrende Aufgabe befindet, können Sie diese mithilfe von *Get-ScheduledTask* finden. Folgende Befehle können Sie für diese Aufgabe verwenden:

- a) `Get-ScheduledTask -TaskName *backup*`
- b) `Get-ScheduledTask -TaskName Microsoft-Windows-WindowsBackup | Get-ScheduledTaskInfo`

- ✓ Zeile a) liefert eine Liste aller Aufgaben, die den Begriff *backup* beinhalten. Sie finden damit auch den Task *Microsoft-Windows-WindowsBackup*, der den Sicherungszeitplan aus der Windows Server-Sicherung repräsentiert.
- ✓ In Zeile b) wird der gefundene Task explizit angesprochen und über die Pipeline an das Cmdlet *Get-ScheduledTaskInfo* zum Auslesen von Detailinformationen weitergereicht.

```

Administrator: Windows PowerShell
PS C:\Users\Administrator> Get-ScheduledTask -TaskName *backup*
TaskPath                               TaskName          State
-----                               -----          -----
\Microsoft\Windows\Backup\           Microsoft-WindowsBackup Ready
\Microsoft\Windows\Registry\         RegIdleBackup   Ready
\Microsoft\Windows\SettingSync\      BackupTask      Ready

PS C:\Users\Administrator> Get-ScheduledTask -TaskName Microsoft-WindowsBackup | Get-ScheduledTaskInfo

LastRunTime      : 29.11.1999 23:00:00
LastTaskResult   : 267011
NextRunTime      : 29.10.2016 22:00:00 ①
NumberOfMissedRuns : 0 ②
TaskName         : Microsoft-WindowsBackup
TaskPath         : \Microsoft\Windows\Backup\
PSComputerName   :

```

Details zu einer Aufgabe der Windows Server-Sicherung

Besonders interessant sind die Zeilen *NextRunTime* ① (nächste Ausführung der Aufgabe) und *NumberOfMissedRuns* ② (Anzahl fehlgeschlagener bzw. verpasster Ausführungen). Diese Eigenschaften können Sie in der PowerShell auch für allgemeinere Abfragen oder Filterungen einsetzen, wie z. B. Fragen nach Aufgaben mit verpassten Ausführungen (a)) und Aufgaben mit angegebenem nächsten Ausführungszeitpunkt (b)):

- a) Get-ScheduledTask | Get-ScheduledTaskInfo | Where-Object {\$\_.\_.NumberOfMissedRuns -gt 0}
- b) Get-ScheduledTask | Get-ScheduledTaskInfo | Where-Object {\$\_.\_.NextRunTime -ne \$null}

Darüber hinaus unterstützen weitere Cmdlets bei wichtigen Standardtätigkeiten, die einfach mit der PowerShell gesteuert werden können:

Befehl	Erläuterung
Disable-ScheduledTask	Deaktiviert die angegebene Aufgabe
Enable-ScheduledTask	Aktiviert die angegebene Aufgabe
Start-ScheduledTask	Startet die angegebene Aufgabe
Stop-ScheduledTask	Hält die angegebene Aufgabe an
Unregister-ScheduledTask	Löscht die angegebene Aufgabe

## 13.2 IP-Konfiguration: PowerShell anstelle von `netsh.exe` verwenden

### Netsh.exe bald (teilweise) nicht mehr verfügbar?

Netsh.exe ist eines der mächtigsten Werkzeuge an der Eingabeaufforderung. Im Bereich der Serverkonfiguration kennen Sie netsh.exe möglicherweise zur Konfiguration von IP-Adressinformationen und DNS-Server z. B. unter Windows Server 2008 (R2) Server Core.

Wenn Sie unter Windows Server 2016 netsh.exe an der Befehlszeile ausführen und in den Kontext interface zur Verwaltung der Netzwerkkarte wechseln, erhalten Sie die folgende Meldung:

In zukünftigen Versionen von Windows wird die Netsh-Funktionalität möglicherweise nicht mehr für "TCP/IP" verfügbar sein. Es empfiehlt sich ein Wechsel zu Windows PowerShell, wenn momentan netsh dazu verwendet wird, "TCP/IP" zu konfigurieren und zu verwalten.

Windows Server 2019 warnt zielgerichteter mit dem Satz:

In zukünftigen Versionen von Windows wird die Netsh-Funktionalität für TCP/IP von Microsoft entfernt.

Windows 10 und Windows Server 2016 (oder 2019) bieten eine Vielzahl an Cmdlets zur Konfiguration von Netzwerkadapters, IP-Adress- und DNS-Einstellungen. Die Anzahl von etwa 310 Befehlen für diesen Bereich sprengt allerdings den Rahmen, sodass an dieser Stelle nur eine kleine Auswahl vorgestellt wird.

Um alle Befehle im Bereich Netzwerk zu sehen, geben Sie den folgenden Befehl ein:

```
Get-Command -Noun Net*
```

Um einen Überblick über die beteiligten PowerShell-Module zu erhalten, erweitern Sie den Befehl:

```
Get-Command -Noun Net* | Group-Object -Property ModuleName
```

Count	Name	Group
35	NetEventPacketCapture	{Add-NetEventNetworkAdapter, Add-NetEventPacketCapture...
34	NetworkTransition	{Add-NetIPHttpsCertBinding, Disable-NetDnsTransitionCo...
13	NetLbfo	{Add-NetLbfoTeamMember, Add-NetLbfoTeamNic, Get-NetLbf...
13	NetNat	{Add-NetNatExternalAddress, Add-NetNatStaticMapping, G...
7	NetSwitchTeam	{Add-NetSwitchTeamMember, Get-NetSwitchTeam, Get-NetSw...
84	NetSecurity	{Copy-NetFirewallRule, Copy-NetIPsecMainModeCryptoSet,...
68	NetAdapter	{Disable-NetAdapter, Disable-NetAdapterBinding, Disabl...
19	NetworkSwitchManager	{Disable-NetworkSwitchEthernetPort, Disable-NetworkSwi...
34	NetTCPIP	{Find-NetRoute, Get-NetCompartment, Get-NetIPAddress, ...}
2	NetConnection	{Get-NetConnectionProfile, Set-NetConnectionProfile}
4	NetQos	{Get-NetQosPolicy, New-NetQosPolicy, Remove-NetQosPolি...}

Hilfe zu den einzelnen Befehlen erhalten Sie durch Verwendung des Cmdlets `Get-Help <Befehl>`.

## Über Netzwerkadapter und -konfiguration informieren

Das Cmdlet `Get-NetAdapter` ist ein Bestandteil des Moduls `NetAdapter`. Es zeigt die Basiseigenschaften der Netzwerkadapter Ihres Systems. Wollen Sie Informationen zu einem speziellen Adapter erhalten, verwenden Sie den Parameter `-Name`. Wollen Sie die Anzeige virtueller Adapter unterdrücken, setzen Sie den Switch-Parameter `-Physical` ein.

Ausführlichere Informationen erhalten Sie durch a) Formatierung als Liste und vor allem durch b) Einblendung aller Eigenschaften:

```
a) Get-NetAdapter | Format-List
b) Get-NetAdapter | Select-Object -Property *
```

Bevor Sie mit der PowerShell eine IP-Konfiguration vornehmen, rufen Sie zuvor mit dem Cmdlet `Get-NetIPConfiguration` (PowerShell-Modul `NetTCP/IP`) die aktuellen Einstellungen ab:

```
Get-NetIPConfiguration -InterfaceAlias "Ethernet 2" -Detailed
```

Sie erhalten ausführliche Informationen (Parameter `-Detailed`) über den Computernamen, den angegebenen Netzwerkadapter (Parameter `-InterfaceAlias`) inklusive physikalischer Adresse sowie die IP-Adresskonfiguration.

### IP-Konfiguration vornehmen

Zuerst entscheiden Sie sich, ob Ihr Netzwerkadapter von einem DHCP-Server automatisch konfiguriert werden soll oder Sie eine statische Konfiguration vornehmen. Um dies festzulegen, verwenden Sie das Cmdlet `Set-NetIPInterface` des Moduls `NetTCPiP`. Das Cmdlet kann eine Reihe nützlicher Aufgaben im IPv4- und IPv6-Umfeld erledigen. Hier ist der Parameter `-Dhcp` interessant, der die Werte `Enabled` oder `Disabled` annehmen kann.

Um DHCP für den angegebenen Netzwerkadapter zu aktivieren (a) oder zu deaktivieren (b), geben Sie folgenden Befehl ein:

- a) `Set-NetIPInterface -InterfaceAlias "Ethernet0" -Dhcp Enabled`
- b) `Set-NetIPInterface -InterfaceAlias "Ethernet0" -Dhcp Disabled`

Wenn Sie DHCP aktivieren, übernimmt der DHCP-Server die IP-Konfiguration des Rechners. Haben Sie DHCP deaktiviert, konfigurieren Sie eine statische Adresse mit dem Cmdlet `New-NetIPAddress`.

Das Cmdlet `New-NetIPAddress` ist eines von vier Cmdlets des Moduls `NetTCPiP`, die sich direkt um die Verwaltung von IP-Adressen kümmern. Die weiteren Cmdlets lauten: `Get-NetIPAddress` (Informationen auslesen), `Set-NetIPAddress` (Änderungen vornehmen) und `Remove-NetIPAddress` (Adresse entfernen). Sie haben bereits ähnliche Cmdlet-Familien kennengelernt.

`New-NetIPAddress` bietet u. a. folgende Parameter zur Erstellung einer neuen IPv4- oder IPv6-Adresse:

New-NetIPAddress		
Parameter	Typ	Beschreibung
<code>-IPAddress</code>	P (1)	Angabe einer IP-Adresse
<code>-DefaultGateway</code>	N	Angabe der IP-Adresse des Standard-Gateways
<code>-InterfaceAlias</code>	N	Bezeichnung des Netzwerkadapters, auf den die Einstellungen angewendet werden sollen
<code>-PrefixLength</code>	N	Angabe der Subnetzmaske in CIDR-Notation, bei der die Anzahl der 1en in der binären Schreibweise der Subnetzmaske angegeben werden, z. B. ✓ 8 steht für 255.0.0.0 ✓ 16 steht für 255.255.0.0 ✓ 24 steht für 255.255.255.0
...		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: <code>Get-Help New-NetIPAddress -Full</code>

Um dem Netzwerkadapter Ethernet die IP-Adresse 192.168.100.10/24 mit dem Standard-Gateway 192.168.100.254 zuzuweisen, geben Sie folgenden Befehl ein:

```
New-NetIPAddress -IPAddress 192.168.100.10 -PrefixLength 24 -DefaultGateway
192.168.100.254 -InterfaceAlias "Ethernet0"
```

## DNS-Serveradressen am Client verwalten

Eine IP-Adresszuweisung ist in der Domäne nur ein Teil der nötigen IP-Konfiguration eines jeden Rechners. Es fehlt noch die wichtige Angabe einer DNS-Server-IP-Adresse. Um einen oder mehrere DNS-Server einzutragen, verwenden Sie ein Cmdlet des Moduls *DnsClient*: `Set-DnsClientServerAddress`.

Das Cmdlet benötigt mindestens die Parameter `-InterfaceAlias` zur Angabe des zu konfigurierenden Netzwerkadapters und `-ServerAddresses` zur Angabe einer (a) oder mehrerer (b) IP-Adressen von DNS-Servern:

- a) `Set-DnsClientServerAddress -InterfaceAlias "Ethernet0" -ServerAddresses 192.168.100.254`
- b) `Set-DnsClientServerAddress -InterfaceAlias "Ethernet0" -ServerAddresses 192.168.100.254,192.168.100.253`

Wollen Sie die DNS-Konfiguration eines Netzwerkadapters auf den Ausgangszustand zurücksetzen, verwenden Sie anstelle des Parameters `-ServerAddresses` den Parameter `-ResetServerAddresses`:

```
Set-DnsClientServerAddress -InterfaceAlias "Ethernet0"  
-ResetServerAddresses
```

Um Informationen zur DNS-Serverkonfiguration eines Netzwerkadapters anzuzeigen, verwenden Sie das Cmdlet `Get-DnsClientServerAddress`. Geben Sie keinen Parameter an (a), erhalten Sie eine Ausgabe aller DNS-Server-IP-Adressen aller verfügbaren Netzwerkadapter. Verwenden Sie den Parameter `-InterfaceAlias`, erhalten Sie nur Informationen über den als Wert angegebenen Netzwerkadapter (b). Um die Ausgabe auf IPv4- oder IPv6-Adressen zu beschränken, setzen Sie den Parameter `-AddressFamily` mit dem Wert `IPv4` oder `IPv6` ein (c):

- a) `Get-DnsClientServerAddress`
- b) `Get-DnsClientServerAddress -InterfaceAlias "Ethernet0"`
- c) `Get-DnsClientServerAddress -InterfaceAlias "Ethernet0" -AddressFamily IPv4`

Eine IP-Grundkonfiguration eines Rechners bzw. eines Netzwerkadapters ist auch mit der PowerShell schnell zu realisieren. Weitere Schritte könnten die Ausdehnung der Konfiguration auf Remotesysteme oder der automatisierte, skriptgesteuerte Ausbau einer Gesamtstruktur inklusive der Konfiguration der Domänencontroller sein. Den letztgenannten Punkt lesen Sie in der Fallstudie des abschließenden Kapitels.

### 13.3 Server Core: PowerShell als Standard-Shell einrichten

 Beispieldatei: `Core_Shell.ps1` (Ordner Kapitel 13)

Die PowerShell steht auch im System zur Verfügung, auf dem Windows Server 2016 Server Core installiert ist. Beim Start des Rechners erscheint aber wie gewohnt die Eingabeaufforderung (`cmd.exe`).

Um in die PowerShell zu wechseln, geben Sie an der Eingabeaufforderung einfach `powershell` ein. Im selben Fenster ändert sich der Prompt, und Sie können mit der PowerShell arbeiten. Wollen Sie – im selben Fenster – zur Eingabeaufforderung zurückkehren, geben Sie den Befehl `exit` ein.

Um ein separates Fenster für die PowerShell im gewohnten Layout zu öffnen, geben Sie in der Eingabeaufforderung folgenden Befehl ein:

```
start powershell.exe
```

Doch warum manuell umschalten? Sie kennen viele Vorteile der PowerShell im Gegensatz zur Eingabeaufforderung. Die folgende Funktion zeigt die Möglichkeit, die PowerShell für den Server Core als Standard-Shell zu definieren bzw. wieder zur bisherigen Shell zurückzukehren.

Ein Wert in der Registry ist dafür zuständig, mit welcher Shell ein Windows Server Core startet. Die Funktion ermöglicht Ihnen die Festlegung von Eingabeaufforderung oder PowerShell als Standard-Shell nach einem Systemneustart:

```

① function Set-PSShell ([switch]$Aus)
{
    ② $Pfad = "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon"
    ③ $Wert = 'PowerShell.exe -noExit -Command "Set-Location $HOME"'
    ④ If ($Aus)
    {
        ⑤ $Wert = 'explorer.exe'
    }
    ⑥ Set-ItemProperty -Path $Pfad -Name Shell -Value $Wert -Confirm:$False
    ⑦ Restart-Computer -Force
}

```

*Beispielskript „Core\_Shell.ps1“*

- ① Die neue Funktion erhält den Namen `Set-PSShell` und einen Switch-Parameter `-Aus`.
- ② Die Variable `$Pfad` enthält als Wert den Pfad zu dem Schlüssel in der Registry, der den Eintrag `Shell` enthält, der für das Startverhalten des Windows Servers verantwortlich ist.
- ③ In der Variablen `$Shell` wird der Wert gespeichert, der als Wert des Eintrags in der Registry verwendet werden soll. Bei dem Wert handelt es sich um einen Befehl, der eine neue PowerShell-Sitzung startet, einen Befehl zum Wechsel in das Benutzerverzeichnis ausführt (Parameter `-Command`) und anschließend die Anwendung geöffnet hält (Parameter `-NoExit`).
- ④ Es wird geprüft, ob die Variable `$Aus` existiert. Das ist der Fall, wenn beim Aufruf der Funktion der Switch-Parameter `-Aus` angegeben wurde. Wenn Sie permanent zur PowerShell wechseln möchten, lassen Sie diesen Parameter weg. Wollen Sie zurück zur Eingabeaufforderung als Standard-Shell, rufen Sie die Funktion mit dem Parameter `-Aus` auf.
- ⑤ Trifft die Bedingung der `If`-Anweisung zu, wird der Variablen `$Wert` der Wert `'explorer.exe'` zugewiesen. Das ist der Standardeintrag des angesprochenen Eintrags in der Registry, der zum Start der Eingabeaufforderung beim Server Core führt.
- ⑥ Mithilfe des Cmdlets `Set-ItemProperty` erfolgt ein Zugriff auf die Registry zur Änderung des Werts für den genannten Eintrag.
- ⑦ Der Computer wird zur Durchführung der Änderungen neu gestartet. (Eine Abmeldung des aktuell angemeldeten Benutzers würde ebenfalls ausreichen.)

Wenn Sie dieses Skript ausführen, wird allerdings die dort definierte Funktion nicht ausgeführt. Damit die im Skript definierte Funktion auch nach Ausführung des Skripts allgemein zur Verfügung steht – und damit später auch wirklich ausgeführt werden kann –, rufen Sie das Skript (im aktuellen Verzeichnis) „dot sourced“ auf:

```
. .\Core_Shell.ps1
```

### Exkurs: Skript „dot sourced“ ausführen

Damit Sie mit den Funktionen im Skript arbeiten können, müssen Sie das Skript ausführen. Damit die Funktionen nach Ausführung des Skripts – bis zur Beendigung der aktuellen PowerShell-Sitzung – allgemein verfügbar sind, müssen Sie das sogenannte „Dotsourcing“ anwenden:

```

① PS C:\Users\Administrator> .\Documents\MyFunctions\Core_Shell.ps1
② PS C:\Users\Administrator> Get-ChildItem function:\set*
③ PS C:\Users\Administrator> . .\Documents\MyFunctions\Core_Shell.ps1 ←
④ PS C:\Users\Administrator> Get-ChildItem function:\set*

 CommandType      Name          Version      Source
 -----          ----          -----      -----
 Function        Set-PSShell

 PS C:\Users\Administrator> -

```

Skript „dot sourced“ ausführen

Die Eingabe der ersten Zeile ① führt das Skript *Core\_Shell.ps1* wie gewohnt aus. Zur Laufzeit des Skripts stehen die im Skript definierten Funktionen zur Verfügung. Nach Beendigung des Skripts erfolgt ein Aufräumprozess, so dass die Definitionen nicht mehr zur Verfügung stehen. Dies zeigt die zweite Eingabezeile ②, die auf dem virtuellen Laufwerk *Function:* nach Funktionen sucht, die mit der Zeichenfolge *set* beginnen.

In der dritten Eingabezeile ③ wird das Skript mit den Funktionsdefinitionen erneut aufgerufen. Diesmal erfolgt der Aufruf „dot sourced“ mit vorangestelltem Punkt und Leerzeichen (.) (. ). Die vierte Eingabezeile ④ zeigt die Auswirkungen des veränderten Skriptaufrufs. Auch nach Beendigung der Skriptausführung stehen nun – bis zum Ende der aktuellen PowerShell-Sitzung – die im Skript definierten Funktionen zur Verfügung.



Merken Sie sich dieses Vorgehen für alle Teile eines beliebigen Skripts, das über dessen Laufzeit hinaus zur Verfügung stehen soll.

## 13.4 Windows Explorer aus der PowerShell heraus starten

Manchmal ist es hilfreich, die grafische Oberfläche des Windows Explorers während der Arbeit mit der PowerShell zur Hand zu haben, z. B. um sich im aktuellen Ordner zu orientieren oder spezielle Tätigkeiten auszuführen.

Wie bei anderen externen Programmen können Sie den Windows Explorer durch Eingabe des absoluten Pfads zum Programm (*C:\Windows\explorer.exe*) oder einfach durch die Eingabe von *explorer.exe* starten. Dies ist möglich, da der Windows-Pfad in der Systemvariable *Path* verzeichnet ist und das Programm damit ohne Angabe des Pfads gefunden wird.

Versuchen Sie folgenden Befehl:

```
start .
```

- ✓ *start* ist in der PowerShell der Alias für *Start-Process*. Der nachfolgend als Argument angegebene Punkt bezeichnet das aktuelle Verzeichnis. Der Windows Explorer öffnet sich in einem neuen Fenster.

Somit stehen auch die Parameter des Cmdlets *Start-Process* zur Verfügung, wie z. B. der Parameter *-WindowSize* mit den möglichen Werten *Hidden*, *Maximized*, *Minimized* und *Normal*. Durch Angabe des Parameters steuern Sie, ob das neue Fenster versteckt, in voller Größe, minimiert oder normal (Standardwert) angezeigt werden soll, z. B.

```
start C:\Windows\System32\WindowsPowerShell\v1.0 -WindowSize Maximized
```

## 13.5 Startzeitpunkt und Laufzeit eines Servers bestimmen

Im Beispiel soll eine Funktion geschrieben werden, die anzeigt, wann ein Rechner gestartet wurde und seit wann er ununterbrochen läuft. Die Funktion soll allgemeingültig sein und versionsunabhängig funktionieren. Diese Aufgabe lässt sich erledigen, indem Sie Daten z. B. aus WMI (Windows Management Instrumentation) anzapfen und in der PowerShell weiterverarbeiten.

**WMI** ist eine standardisierte Schnittstelle, mit der auf nahezu alle System- und Netzwerk-Einstellungen eines Windows-Computers zugegriffen werden können. In Windows-Systemen ist WMI integraler Bestandteil. WMI ist die Microsoft-Implementation und -Erweiterung von CIM (Common Information Model).

Da die PowerShell Informationen von WMI und – seit Version 3 – auch von CIM auslesen kann, ist das möglich.

Die Informationen liegen in WMI objektorientiert vor: Sie finden dort Klassen mit Methoden und Eigenschaften sowie strukturierte Objekte als Instanzen der Klassen.

Zur Orientierung, welche Klassen verfügbar sind, können Sie folgenden Befehl verwenden:

```
Get-WmiObject -List
```

Sie erhalten eine sehr lange Liste mit einer vierstelligen Anzahl von Klassen. Das erlaubt nicht wirklich einen guten Überblick. Um gezielter nach Klassen suchen zu können, können Sie z. B. nach einem Begriff filtern:

```
Get-WmiObject -List | Where-Object {$_.name -like "*operating*"}  
PS C:\Users\Administrator> Get-WmiObject -List | Where-Object {$_.name -like "*operating*"}  
NameSpace: ROOT\cimv2  
Name Methods Properties  
---- {} {Element, Setting}  
Win32_OperatingSystemAutochkSetting {} {Caption, CreationClassName, CSCreationClassName, ...}  
CIM_OperatingSystem {Reboot, Shutdown} {BootDevice, BuildNumber, BuildType, Caption...}  
Win32_OperatingSystem {Reboot, Shutdown...} {GroupComponent, PartComponent, PrimaryOS}  
Win32_SystemOperatingSystem {} {GroupComponent, PartComponent}  
CIM_OperatingSystemSoftwareFeature {} {Antecedent, Dependent}  
Win32_OperatingSystemQFE {}
```

*Suche nach WMI-Klassen mit „operating“ im Namen*

Dasselbe Ergebnis erhalten Sie auch, wenn Sie mit einem ähnlichen Befehl CIM abfragen:

```
Get-CimClass -ClassName "*operating*"
```

Wenn Sie die Klasse gefunden haben, aus der Sie Informationen abfragen wollen (z. B. *Win32\_OperatingSystem*), empfiehlt sich eine weitere Abfrage, um die Eigenschaften der Klasse zu betrachten:

```
Get-WmiObject -Class Win32_OperatingSystem | Select-Object -Property *
```

Sie erhalten eine Auflistung der in der Klasse verfügbaren Eigenschaften inklusive der dazugehörigen Werte.

Lesen Sie diese Informationen mit den CIM-Cmdlets aus, dann sehen Sie in der Ausgabe, dass Sie eine strukturiertere Rückmeldung erhalten:

```
Get-CimClass -ClassName CIM_OperatingSystem | Select-Object -Property *
```

```
PS C:\Users\Administrator> Get-CimClass -ClassName CIM_OperatingSystem | Select-Object -Property *
```

```
CimClassName      : CIM_OperatingSystem
CimSuperClassName : CIM_LogicalElement
CimSuperClass     : ROOT/CIMV2:CIM_LogicalElement
CimClassProperties : {Caption, Description, InstallDate, Name...}
CimClassQualifiers : {Locale, UUID, Abstract}
CimClassMethods   : {Reboot, Shutdown}
CimSystemProperties : Microsoft.Management.Infrastructure.CimSystemProperties
```

Um nun die Eigenschaften der Klasse komplett anzeigen zu lassen, verwenden Sie den folgenden Befehl. Wichtig dabei ist, dass Sie den Parameter `-ExpandProperty` (und nicht `-Property`) verwenden. Nur so wird das Array `CimClassProperties` komplett ausgelesen:

```
Get-CimClass -ClassName CIM_OperatingSystem |
Select-Object -ExpandProperty CimClassProperties
```

In der WMI finden Sie in der Klasse `Win32_OperatingSystem` eine Eigenschaft mit der Bezeichnung `LastBootUpTime`. Die Eigenschaft besitzt als Wert einen Zeitstempel des letzten Systemstarts, den Sie mit folgendem Befehl abrufen können:

```
(Get-WmiObject -Class Win32_OperatingSystem).LastBootUpTime
```

Die Ausgabe erscheint nicht als formatierter Datums- und Zeitwert, wie die folgende Abbildung zeigt.

```
PS C:\Users\Administrator> (Get-WmiObject -Class Win32_OperatingSystem).LastBootUpTime
20161028115346.497321+120
```

Aus WMI ausgelesener Wert für den letzten Systemstart

Möchten Sie wissen, ob die Klasse Methoden liefert, die diese Angabe in einen Datums- bzw. Zeitwert konvertieren kann, geben Sie in PowerShell folgende Zeile ein:

```
Get-WmiObject -Class Win32_OperatingSystem | Get-Member -MemberType *Method
```

Neben anderen Methoden wird auch die Methode `Convert.ToDateTime` angegeben, die auf den Wert der Eigenschaft `LastBootUpTime` angewendet werden kann.

Damit können die per WMI (oder CIM) ausgelesenen Werte in einen Datumswert umgewandelt werden. Die restlichen Tätigkeiten sind aus vorhergehenden Kapiteln bekannt: Berechnung einer Zeitspanne, Verknüpfung von Zeichenketten und Definition einer Funktion. Eine Lösungsmöglichkeit zeigt die folgende Funktion.

**Beispiel:** `Get-TimeUp.ps1`

**Plus** Beispieldatei: `Get-TimeUp.ps1` (Ordner Kapitel 13)

Auf Basis der Vorüberlegungen wird ein Skript `Get-TimeUp.ps1` entwickelt, das die Startzeit eines Rechners ausliest und berechnet, seit wann der Rechner läuft.

```
① function Get-TimeUp
{
② Param ( [string] $RechnerName = $env:COMPUTERNAME )
③ $daten = Get-WmiObject Win32_OperatingSystem -ComputerName $RechnerName
④ if ($daten.LastBootUpTime)
```

```

⑤ $startzeit = $daten.Convert.ToDateTime($daten.LastBootUpTime)
$zeitspanne = (Get-Date) - $startzeit
⑥ Write-Output ("Rechnername : $RechnerName")
Write-Output ("Letzter Start: " +
              ($startzeit).ToString("dd.MM.yyyy hh:mm:ss"))
Write-Output ("Läuft seit : " + $zeitspanne.Days + " Tage " +
$zeitspanne.Hours + " Stunden " + $zeitspanne.Minutes + " Minuten")
}
else
{
    Write-Warning "Keine Verbindung zu $RechnerName."
}
}

```

- ① Die Funktion *Get-TimeUp* wird definiert. Wurde die Funktionsdefinition einmal aufgerufen, steht die Funktion während der Laufzeit der aktuellen PowerShell-Instanz zur Verfügung. Wenn Sie eine stete Verfügbarkeit der Funktion wünschen, binden Sie sie in ein Profilskript ein, das zu Beginn jeder Sitzung abgearbeitet wird.
- ② Um die Funktion flexibler zu gestalten, wird ein Parameter \$RechnerName definiert und mit einem Standardwert, der Umgebungsvariablen \$env:COMPUTERNAME (dem lokalen Rechner) initialisiert. In der Funktion steht damit der Parameter -RechnerName zur Verfügung, der es Ihnen erlaubt, die Funktion auch für Remoterechner auszuführen. Rufen Sie die Funktion ohne Parameter auf, wird sie automatisch für den lokalen Rechner ausgeführt.
- ③ Die Abfrage der WMI-Klasse *Win32\_OperatingSystem* wird für den angegebenen Computer lokal oder remote ausgeführt, sofern es die Berechtigungen des Benutzers zulassen.
- ④ Es wird die Bedingung formuliert, dass Berechnungen nur dann stattfinden sollen, wenn entsprechende Daten vorliegen. Andernfalls wird der else-Zweig am Ende der Funktion ausgeführt.
- ⑤ Der Wert, wann der Rechner gestartet wurde, wird ermittelt (*LastBootUpTime*), in einen Datumswert (*Convert.ToDateTime*) konvertiert und dann als Wert der Variablen \$startzeit zugewiesen. In der Variablen \$zeitspanne wird dieser Wert vom aktuellen Datum und der aktuellen Uhrzeit abgezogen, um die Zeitspanne zu berechnen, die der Rechner bereits läuft.
- ⑥ Schließlich werden drei Zeilen als Ergebnis ausgegeben. In der ersten Zeile wird der Rechner genannt, für den die Berechnungen ausgeführt wurden. In der zweiten Zeile wird der ausgelesene Startzeitpunkt des Rechners in gewünschtem Format angezeigt. In der letzten Zeile erfolgt eine Ausgabe der Laufzeit des Rechners, indem auf die Eigenschaften *Days*, *Hours* und *Minutes* zugegriffen wird. Dies ist möglich, da es sich um einen Datums-/Zeitwert handelt.

Auch dieses Skript müssen Sie „dot sourced“ ausführen, damit die enthaltene Funktion auch nach Abarbeitung des Skripts im Rahmen Ihrer PowerShell-Sitzung zur Verfügung steht. Nachfolgend sehen Sie den Aufruf des Skripts, das sich im aktuellen Verzeichnis befindet:

```
. .\Get-TimeUp.ps1
```

Für den lokalen Rechner sieht die Ausgabe nach Ausführung der Funktion in etwa wie folgt aus:

```

PS C:\Users\Administrator> Get-TimeUp
Rechnername : 2016-FINAL
Letzter Start: 28.10.2016 11:53:46
Läuft seit   : 2 Tage 12 Stunden 6 Minuten

```

*Aufruf der Funktion für den lokalen Rechner*

Soll ein Remoterechner abgefragt werden, rufen Sie die Funktion mit dem Parameter -RechnerName auf, z. B.

```
Get-TimeUp -RechnerName ServerX
```

Seit Version 3 der PowerShell können Sie auch CIM-Cmdlets einsetzen. Es gibt für den gezeigten Sachverhalt der nicht lesbaren Zeitangabe eine einfache Lösung:

```
(Get-CimInstance -ClassName CIM_OperatingSystem).LastBootUpTime
```

Diese Lösung ist im Vergleich zur vorherigen Vorgehensweise **versionsabhängig**.

## 13.6 Kurz zusammengefasst

Sonderaufgaben für die PowerShell gibt es viele. Es kursieren viele Tipps und Tricks, mit denen Sie Ihre Netzwerkadministration erleichtern können. Die in diesem Kapitel angerissenen Themen gehören dazu.

Die Vielzahl an Modulen, die in der neuen PowerShell-Generation zur Verfügung stehen, eröffnet neue Möglichkeiten. So ersetzt eine IP-Adresskonfiguration mit der PowerShell in Zukunft die entsprechenden Tätigkeiten mit netsh.exe. Denken Sie bei solchen Ausgaben immer einen Schritt weiter: Es geht bei diesen Themen weniger um eine Alternative zur grafischen Oberfläche, sondern eher um die Entwicklung einer Programmierung für häufige Anwendung bzw. um die Automatisierung von wiederkehrenden Aufgaben.

## 13.7 Ausblick: PowerShell – next level

Das Buch hat Ihnen einen fundierten Einblick in die Grundlagen der PowerShell geboten. Darüber hinaus haben Sie im Haupteinsatzgebiet der Netzwerkverwaltung in Domänen Aspekte des häufigsten Einsatzes der PowerShell kennengelernt. Bei einem so umfassenden Thema kann dies nur ausschnittsweise geschehen.

Freuen Sie sich auf den nächsten Level der PowerShell. Bei der fortgeschrittenen Beschäftigung mit der PowerShell gibt es noch viel zu entdecken, wofür Sie die erworbenen Basiskenntnisse benötigen. Weiterführende Themen könnten u.a. wie folgt lauten:

- ✓ PowerShell remote einsetzen
- ✓ Fehlerbehandlung und Debugging Ihrer eigenen Programmierung
- ✓ Arbeit mit WMI (*Windows Management Instrumentation*) und CIM (*Common Information Model*)
- ✓ Eigene Module erstellen
- ✓ Mit Jobs und Workflows arbeiten
- ✓ Windows PowerShell Web Access – Zugriff auf die PowerShell über einen Browser, z.B. von einem internethfähigen Rechner außerhalb Ihres Firmennetzwerks
- ✓ DSC (*Desired State Configuration*) – Erstellen einer Steuerdatei, die eine gewünschte Konfiguration für Remoterechner beschreibt. Die Steuerdatei umfasst beispielsweise die Installation von Features, die Einstellungen der Registry, die Verwaltung von Benutzern und Gruppen und das Arbeiten im Dateisystem. Wird die Datei für die Remoterechner ausgeführt, erhalten Sie im Ergebnis dort die gewünschte Konfiguration.



**Wissenstest:** PowerShell-Verwaltung Active Directory

## 13.8 Übung

### Sonderaufgaben für die PowerShell

**Übungsdatei:** *Set-PSShell.ps1*

**Ergebnisdatei:** *13\_ergebnisse.txt, interface-test-lokal.ps1,  
ip-konfiguration.ps1, Get-Set-PSShell.ps1*

1. Erstellen Sie in Windows in der Aufgabenverwaltung eine neue Aufgabe und fragen Sie Details zu dieser Aufgabe mit der PowerShell ab.
2. Schreiben Sie ein Skript, das für Ihren lokalen Rechner die komplette statische IP-Adressierung – mit Adressen Ihrer Wahl – vornimmt. Zur Vereinfachung stellen Sie vorher ein, die IP-Adresse und die DNS-Serveradresse automatisch zu beziehen (Ergebnis: *ip-konfiguration.ps1*).
3.
  - a) Schreiben Sie eine Funktion `Get-PSShell`, die – analog zur Funktion `Set-PSShell` im Skriptbeispiel *Set-PSShell.ps1* in diesem Kapitel – den aktuellen Wert des Eintrags `Shell` am angegebenen Ort in der Registry anzeigt (Ergebnis: *Get-Set-PSShell.ps1*).
  - b) Erweitern Sie die Funktion `Set-PSShell` im Skriptbeispiel *Set-PSShell.ps1* in diesem Kapitel: Sorgen Sie dafür, dass die Funktion nur dann ausgeführt wird, wenn eine andere Shell als die aktuelle gewählt wird (Ergebnis: *Get-Set-PSShell.ps1*).

# 14 Praxiskapitel: Automatisierung in der Domäne

 Beispieldatei: 14\_kompletter code.txt (Ordner Kapitel 14)

## 14.1 Installation einer neuen Gesamtstruktur

### Ausgangssituation

Beim Aufbau einer neuen Gesamtstruktur und einer neuen Domäne sind immer wieder identische Arbeitsschritte auszuführen. In diesem Kapitel soll die mögliche Anwendung von PowerShell-Skripten gezeigt werden, um das Grundgerüst eines Netzwerks aufzubauen: Gesamtstruktur, Domäne, Domänencontroller, Organisationseinheiten und der Massenimport von Benutzerkonten.

Als Ausgangspunkt wird ein neu installierter und noch nicht konfigurierter Rechner oder eine virtuelle Maschine mit dem Betriebssystem Windows Server 2016 (oder 2019) vorausgesetzt.

Folgende Einstellungen finden Sie bei einem Rechner direkt nach der Installation vor:

- ✓ Der Rechnername wurde automatisch zugewiesen und lautet WIN-xxxxxxxxxx. Der Name ist 15 Zeichen lang und beginnt mit *WIN*-, gefolgt von 11 zufällig vergebenen Buchstaben oder Ziffern.
- ✓ Die IP-Konfiguration (IPv4 und IPv6) wird automatisch bezogen. Der Rechner hat keine statischen IP-Adresseinstellungen.
- ✓ In der PowerShell ist die Ausführungsrichtlinie für Skripte ist noch nicht konfiguriert. Jegliche Ausführung von PowerShell-Skripten wird unterbunden.

### Vorbereitende Tätigkeiten

 Beispieldatei: forest-1\_vorbereitung.ps1 (Ordner Kapitel 14)

Die Zuweisung eines Rechnernamens und die Vergabe einer statischen IP-Adresse erfolgen mithilfe eines PowerShell-Skripts. Zuerst müssen Sie die Ausführungsrichtlinie für Skripte in der PowerShell konfigurieren. Dazu geben Sie in der PowerShell folgenden Befehl ein, um die Ausführung lokaler Skripte zu erlauben:

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned
```

Die übrigen vorbereitenden Tätigkeiten können Sie mit einem PowerShell-Skript erledigen. Das folgende Skript zeigt beispielhaft die Umsetzung dieser vorbereitenden Aufgaben:

```
#individuelle Werte IPv4 und Rechnername festlegen
① $Ip4Address = "192.168.200.10"
② $Ip4Prefix = "24"
③ $Ip4Gateway = "192.168.200.1"
④ $RechnerNameNeu = "Test-DC1"
⑤ $Adapter = (Get-NetAdapter).InterfaceAlias
#IP-Adresse zuweisen
```

```

⑥ New-NetIPAddress -IPAddress $Ip4Address -PrefixLength $Ip4Prefix
    -InterfaceAlias $Adapter -DefaultGateway $Ip4Gateway
    #Rechner umbenennen
⑦ Rename-Computer -NewName $RechnerNameNeu -Force
    #Rechner neu starten
⑧ Restart-Computer

```

Beispieldskript „forest-1\_vorbereitung.ps1“

- ① Zu Beginn des Skripts erfolgt die individuelle Festlegung der Werte, die Sie für Ihre Umgebung benötigen. Dies sind vor allem IP-Adresse (①), Subnetzmaske (②) in CIDR-Notation und neuer Rechnername (④). Die Festlegung einer Standard-Gateway-Adresse (③) ist je nach Umgebung optional.
- ⑤ Da auch der Name des Netzwerkadapters in den folgenden Befehlen verwendet wird, erfolgt eine Variablenzuweisung des Adapternamens über die Eigenschaft *InterfaceAlias* nach Ausführung des Cmdlets *Get-NetAdapter*. Verwenden Sie andere Netzwerkadapter als den Standard-Adapter *Ethernet*, nehmen Sie an dieser Stelle eine andere Zuweisung vor.
- ⑥ Mit dem Cmdlet *New-NetIPAddress* erfolgt die Zuweisung der IP-Adresse, der Subnetzmaske und der Adresse des Standard-Gateways für den gewünschten Netzwerkadapter.
- ⑦ Durch das Cmdlet *Rename-Computer* erhält der Rechner den zuvor festgelegten neuen Rechnernamen.
- ⑧ Da die Umbenennung des Rechners einen Neustart des Computers erfordert, wird das Skript mit dem Cmdlet *Restart-Computer* abgeschlossen.

Die Skriptausführung ist nach wenigen Sekunden beendet, der Computer startet neu. Nach dem Neustart können Sie sich durch den folgenden Befehl davon überzeugen, dass die entsprechenden Einstellungen vorgenommen wurden:

```
Get-NetIPConfiguration -InterfaceAlias "Ethernet" -Detailed
```

Damit sind die Vorbereitungen abgeschlossen, Sie können sich der Hauptaufgabe widmen: der Bereitstellung einer neuen Gesamtstruktur und der Hochstufung des Rechners zum ersten Domänencontroller in der Stammdomäne der Gesamtstruktur.

### Hintergrund: Das PowerShell-Modul ADDSDeployment

Das Modul *ADDSDeployment*, das seit PowerShell Version 3.0 verfügbar ist, ermöglicht direkt über PowerShell-Cmdlets den Aufbau von Gesamtstrukturen und Domänen sowie die Installation von Domänencontrollern. Zudem werden Cmdlets bereitgestellt, die einen Test vor einem Installations- bzw. Deinstallationsvorgang durchführen.

Folgende Cmdlets werden bereitgestellt. Die Namen sind aussagekräftig und selbsterklärend (ADDS steht für *Active Directory Domain Services*):

Add (hinzufügen)	Add-ADDSReadOnlyDomainControllerAccount
Install (installieren)	Install-ADDSDomain Install-ADDSDomainController Install-ADDSForest
Test (testen)	Test-ADDSDomainControllerInstallation Test-ADDSDomainControllerUninstallation Test-ADDSDomainInstallation Test-ADDSForestInstallation Test-ADDSReadOnlyDomainControllerAccountCreation
Uninstall (deinstallieren)	Uninstall-ADDSDomainController

## Installation einer neuen Gesamtstruktur, ihrer Stammdomäne und des ersten Domänencontrollers der Stammdomäne



**Beispieldatei:** *forest-2\_installation.ps1* (Ordner Kapitel 14)

Welche Aufgaben sind zu erledigen, bevor dieser Rechner als Domänencontroller der Stammdomäne einer neuen Gesamtstruktur fungieren kann?

- ✓ Die Rolle *Active Directory-Domänendienste* muss installiert werden.
- ✓ Die Rolle *DNS-Server* muss installiert werden, falls kein anderer DNS-Server für die Domäne bereitsteht.
- ✓ Das Feature *Gruppenrichtlinienverwaltung* sollte installiert werden.
- ✓ Anschließend kann eine Gesamtstruktur installiert werden.
- ✓ Der lokale Server wird letztlich zum Domänencontroller der Stammdomäne der neuen Gesamtstruktur.

Das folgende Skript übernimmt die Erledigung aller genannten Aufgaben. Nach einer mehrminütigen Wartezeit und einem automatischen Neustart des Rechners haben Sie das genannte Ziel erreicht:

```
#individuelle Domänenangaben festlegen
① $DomainName = "herdt.ps"
$NetbiosName = "HERDT"

#Installation von AD-Domänendienste, DNS-Server und
Gruppenrichtlinienverwaltung
② Start-Job -Name neueFeatures -ScriptBlock
{
    ③ Add-WindowsFeature -Name "AD-Domain-Services" -IncludeAllSubFeature
        -IncludeManagementTools
    Add-WindowsFeature -Name "DNS" -IncludeAllSubFeature
        -IncludeManagementTools
    Add-WindowsFeature -Name "GPMC" -IncludeAllSubFeature
        -IncludeManagementTools
}
④ Wait-Job -Name neueFeatures

# Erstellen einer neuen Gesamtstruktur + Domänencontroller in der
Stammdomäne
⑤ Import-Module ADDSDeployment
⑥ Install-ADDSForest `

    -DomainName $DomainName `

    -DomainNetbiosName $NetbiosName `

    -ForestMode "Windows Server 2016" `

    -DomainMode "Windows Server 2016" `

    -InstallDns:$True `

    -CreateDnsDelegation:$False `

    -LogPath "C:\Windows\NTDS" `

    -DatabasePath "C:\Windows\NTDS" `

    -SysvolPath "C:\Windows\SYSVOL" `

    -SafeModeAdministratorPassword (ConvertTo-SecureString
        -AsPlainText "Kennw0rt!" -Force) `

    -NoRebootOnCompletion:$False `

    -Force:$True
```

*Beispielskript „forest-2\_installation.ps1“*

- ① Am Anfang von Skripten bietet es sich an, Variablendefinitionen bzw. individuelle Einstellungen vorzunehmen. Einerseits muss ein Anwender nicht im gesamten Skript nach diesen Möglichkeiten suchen, andererseits zeigen Sie direkt, an welchen Stellen Sie Anpassungen durch den Benutzer für sinnvoll erachten. Wenn Sie die Eingriffsmöglichkeiten erweitern möchten, bietet sich inhaltlich Punkt ⑥ an. Im Beispiel sind nur der Domänenname und der NetBIOS-Name vordefiniert worden.
- ② Durch das Cmdlet `Start-Job`, das zu den fortgeschrittenen Techniken der PowerShell gehört, werden die zu einem Skriptblock zusammengefassten Cmdlets (schneller) im Hintergrund ausgeführt. Vor allen Dingen wartet das Skript darauf, bis alle Teile des Jobs ausgeführt wurden, bevor folgende Befehle ausgeführt werden (④).
- ③ Die Rollen *Active Directory-Domänendienste* und *DNS-Server* sowie das Feature *Gruppenrichtlinienverwaltung* werden inklusive aller dazugehörigen Features und aller Verwaltungstools installiert. Erfahrungsgemäß dauert die Ausführung eine Weile. Sie werden in der PowerShell während der Ausführung der Befehle mit Ausgaben bzw. einer Fortschrittsanzeige in der PowerShell ISE über den aktuellen Stand informiert.
- ⑤ Das Modul *ADDSDeployment* wird importiert, das zehn Cmdlets zur Installation und zum vorherigen Test einer Active Directory-Domäne, -Gesamtstruktur und -Domänencontroller bereithält.
- ⑥ Letztlich wird die Active Directory-Gesamtstruktur mithilfe des Cmdlets `Install-ADDSForest` installiert. (Die Schreibweise mit Zeilenumbrüchen nach *Backtick*-Zeichen dient nur der Übersicht.) Alle weiteren Definitionen, wie die Gesamtstruktur zu installieren ist, werden über Parameter festgelegt. Dies sind im Beispiel:
  - ✓ der Name der Stammdomäne (Parameter `-DomainName`) inklusive NetBIOS-Name (Parameter `-DomainNetbiosName`),
  - ✓ Funktionsebene der Gesamtstruktur (Parameter `-ForestMode`) und der Stammdomäne (Parameter `-DomainMode`),
  - ✓ Installation eines DNS-Servers (Parameter `-InstallDns`) und Festlegung, ob eine Delegation eingerichtet wird (Parameter `-CreateDnsDelegation`),
  - ✓ Festlegung der Verzeichnisse, wo Protokolldaten (Parameter `-LogPath`), Active Directory-Datenbank (Parameter `-DatabasePath`) und SYSVOL (Parameter `-SysvolPath`) zu finden sind,
  - ✓ weitere Einstellungen wie das Kennwort für die Verzeichnisdienstwiederherstellung (Parameter `-SafeModeAdministratorPassword`), die Einstellung für einen Neustart nach Abschluss der Befehlausführung (Parameter `-NoRebootOnCompletion`) und die Unterdrückung eventueller Warnmeldungen (Parameter `-Force`).

Nach dem Neustart des Rechners sind alle Arbeiten abgeschlossen. Eine neue Gesamtstruktur inklusive erstem Domänencontroller der Stammdomäne steht zu Ihrer Verfügung. Sie können dies in den verschiedenen Verwaltungstools wie Server-Manger und DNS-Manager sowie mit folgenden Befehlen überprüfen:

- |   |
|---|
| a) <code>Get-WindowsFeature   Where-Object {\$_ .InstallState -eq "Installed"}</code> |
| b) <code>Get-NetIPConfiguration -Detailed</code>                                      |
| c) <code>Get-DNSServerSetting</code> oder <code>Get-DNSServer</code>                  |

- ✓ Mit Befehl a) überprüfen Sie, welche Rollen und Features auf dem lokalen Rechner installiert wurden.
- ✓ Befehl b) liefert Details zur IP-Adress-Konfiguration.
- ✓ Mithilfe von Befehl c) ermitteln Sie Details zur Konfiguration des DNS-Servers bzw. erhalten eine kurze Ausgabe zum DNS-Server.

## 14.2 Domänencontroller einer Domäne hinzufügen

### Ausgangssituation

Die Gesamtstruktur wurde installiert, der erste Domänencontroller verrichtet seinen Dienst in der Stammdomäne. Ziel ist es, der Stammdomäne einen zweiten Server als Domänencontroller hinzuzufügen.

Für die nachfolgenden Tätigkeiten wird erneut ein neu installierter und noch nicht konfigurierter Rechner oder eine virtuelle Maschine mit dem Betriebssystem Windows Server 2016 (oder 2019) vorausgesetzt. Der neue Rechner befindet sich im selben Netzwerk wie der bislang einzige Domänencontroller.

Auch jetzt konfigurieren Sie zunächst den frisch installierten Rechner, damit er die Voraussetzungen für einen Domänencontroller erfüllt.

### Vorbereitende Tätigkeiten



**Beispieldatei:** *domain-1\_vorbereitung.ps1* (Ordner Kapitel 14)

Welche Konfiguration benötigt ein Server, der in die Domäne aufgenommen werden soll, um später zum Domänencontroller herausgestuft zu werden?

- ✓ Der Server muss eine statische IP-Adressierung aufweisen, um als Domänencontroller zu fungieren.
- ✓ Um überhaupt mit einem Domänencontroller bzw. Anmeldeserver kommunizieren zu können, muss eine IP-Adresse eines für die Domäne autorisierten DNS-Servers angegeben werden.
- ✓ Der Rechner sollte einen anderen, nachvollziehbaren Namen erhalten.
- ✓ Da die Arbeiten mit PowerShell-Skripten ausgeführt werden, muss zusätzlich die PowerShell-Ausführungsrichtlinie für Skripte verändert werden.

Im Prinzip werden dieselben Konfigurationsschritte ausgeführt wie beim Aufbau der Gesamtstruktur. Die Ausnahme bildet die wichtige Angabe eines DNS-Servers, die bei der ersten Übung nicht vorgenommen werden musste bzw. konnte. Die Installation des DNS-Servers wurde schließlich erst bei der Ausführung des Skripts vorgenommen.

Nachfolgend sehen Sie das im Vergleich zur vorigen Übung nur leicht veränderte Skript:

```
#individuelle Werte IPV4 und Rechnername festlegen
① $Ip4Address = "192.168.200.20"          #Wert für künftigen 2. DC
$Ip4Prefix = "24"                          #wie in Bsp. 1
$Ip4Gateway = "192.168.200.1"            #wie in Bsp. 1
② $Ip4DnsPrimary = "192.168.200.10"      #Domänen-DNS, IP von DC1
③ $RechnerNameNeu = "Test-DC2"           #Wert für künftigen 2. DC

$Adapter = (Get-NetAdapter).InterfaceAlias

#IP-Adresse zuweisen
New-NetIPAddress -IPAddress $Ip4Address -PrefixLength $Ip4Prefix
-InterfaceAlias $Adapter -DefaultGateway $Ip4Gateway

#DNS-Serveereintrag vornehmen
④ Set-DnsClientServerAddress -InterfaceAlias $Adapter -ServerAddresses
$Ip4DnsPrimary

#Rechner umbenennen
Rename-Computer -NewName $RechnerNameNeu -Force

#Installation von AD-Domänendiensten
```

```
(5) Add-WindowsFeature -Name "AD-Domain-Services" -IncludeAllSubFeature  
-IncludeManagementTools

#Rechner neu starten
Restart-Computer
```

*Beispielskript „domain-1\_vorbereitung.ps1“*

- ① Die IP-Adresse, die Sie zur weiteren Verwendung festlegen, muss einen individuellen Wert erhalten, der sich von der IP-Adressierung des ersten Domänencontrollers unterscheidet. Das ist zwar in der Regel bekannt, wird in Skripten aber gern übersehen.
- ② An dieser Stelle wird eine Variable definiert, die als Wert die IPv4-Adresse des DNS-Servers der Domäne enthält. In der vorigen Übung wurde ein DNS-Server auf dem ersten Domänencontroller installiert. Die anzugebende IP-Adresse ist demnach die des ersten Domänencontrollers.
- ③ Analog zu ① wird hier der individuelle Name des zukünftigen zweiten Domänencontrollers definiert.
- ④ Mithilfe des Cmdlets Set-DnsClientServerAddress wird die IP-Adresse des DNS-Servers in die IP-Konfiguration des lokalen Rechners eingetragen.
- ⑤ Auf einem zukünftigen Domänencontroller muss vor der Hochstufung zum Domänencontroller die Rolle *Active Directory-Domänendienste* installiert werden. Die Installation kann durchaus bereits an dieser Stelle durchgeführt werden.

Nach einem Neustart steht der vorkonfigurierte Rechner für den weiteren Installationsprozess zur Verfügung.

### Installation eines weiteren Domänencontrollers in einer bestehenden Domäne

**Plus** *Beispieldatei: domain-2\_installation.ps1 (Ordner Kapitel 14)*

Es verbleibt noch ein Arbeitsschritt: die Installation des Domänencontrollers in der bestehenden Domäne. Es genügt, das folgende Skript auszuführen.

```
#individuelle Domänenangaben festlegen
① $DomainName = "herdt.ps"
$NetbiosName = "HERDT"

# Hinzufügen eines zweiten Domänencontrollers
② Import-Module ADDSDeployment
③ Install-ADDSDomainController -DomainName $DomainName ` 
    -SafeModeAdministratorPassword (ConvertTo-SecureString
        -AsPlainText "Kennw0rt!" -Force) ` 
    -Credential (Get-Credential -Credential "$NetbiosName\administrator") ` 
    -NoGlobalCatalog:$False ` 
    -InstallDns:$False ` 
    -CreateDnsDelegation:$False ` 
    -CriticalReplicationOnly:$False ` 
    -SiteName "Default-First-Site-Name" ` 
    -LogPath "C:\Windows\NTDS" ` 
    -DatabasePath "C:\Windows\NTDS" ` 
    -SysvolPath "C:\Windows\SYSVOL" ` 
    -NoRebootOnCompletion:$False ` 
    -Force:$True
```

*Beispielskript „domain-2\_installation.ps1“*

- ① Um das Skript allgemeiner zu halten, werden zu Beginn Variablen für den Namen und den NetBIOS-Namen der Domäne definiert. Alternativ können die Namen direkt in den Code geschrieben werden.
- ② Das PowerShell-Modul *ADDSDeployment* wird auch in diesem Skript importiert.
- ③ Das hauptsächliche Cmdlet des Skripts *Install-ADDSDomainController* wird ausgeführt.
- ④ Der Parameter *-Credential* wird verwendet, um das Kennwort des Domänen-Administrators zur Zustimmung der Installation interaktiv einzugeben.
- ⑤ Hier werden – wie auch bei Punkt ⑥ – Einstellungen des neuen Domänencontrollers festgelegt (Replikation und Standort).

Die restlichen Einstellungen kennen Sie bereits aus dem vorigen Beispiel. Die nachfolgende Abbildung zeigt Ihnen das Verhalten der PowerShell während der Ausführung des Skripts. Sie werden über die Art und den Fortgang der Tätigkeiten auf dem Laufenden gehalten.

```

Administrator: Windows PowerShell ISE
Datei Bearbeiten Ansicht Tools Debuggen Add-Ons Hilfe
domain-1_vorbereitung.ps1 domain-2_installation.ps1
4
5 # Hinzufügen eines zweiten Domänencontrollers
6 Import-Module ADDSDeployment
7 Install-ADDSDomainController -DomainName $DomainName ` 
8     -SafeModeAdministratorPassword (ConvertTo-SecureString -AsPlainText "Kennw0rt!" -Force) ` 
9     -Credential (Get-Credential -Credential "$NetbiosName\administrator")
10    -NoGlobalCatalog:$False
11    -InstallDns:$False
12    -CreateDnsDelegation:$False
13    -CriticalReplicationOnly:$False
14    -SiteName "Default-First-Site-Name"
15    -LogPath "C:\Windows\NTDS"
16    -DatabasePath "C:\Windows\NTDS"
17    -SysvolPath "C:\Windows\SYSVOL"
18    -NoRebootOnCompletion:$False
19    -Force:$True

```

#### Rückmeldungen der PowerShell während der Installation des zweiten Domänencontrollers

Nach Beendigung des Skripts erfolgt ein nochmaliger Neustart des Rechners. Beim nächsten Start melden Sie sich dann bereits mit dem Konto eines Domänenadministrators an einem weiteren Domänencontroller der Domäne an.

Zur Überprüfung, ob tatsächlich ein zweiter Domänencontroller zur Verfügung steht, geben Sie den folgenden Befehl ein:

```
Get-ADDomainController -Filter * | Select-Object -Property Name
```

Wollen Sie ausführliche Informationen über Ihre Domänencontroller erhalten, lassen Sie das zweite Cmdlet *Select-Object* weg.

Nach ähnlichem Muster können Sie Ihrer Netzwerkumgebung einen schreibgeschützten Domänencontroller (RODC) hinzufügen oder bei Bedarf Domänencontroller deinstallieren.

## 14.3 Aufbau einer Organisationseinheiten-Struktur

### Ausgangssituation

Viele Firmen arbeiten mit einer weitverzweigten Struktur an Organisationseinheiten (nachfolgend auch kurz: „OU“ für *Organizational Unit*) im Active Directory. Häufig besteht die Struktur aus einer Mischung an geografischen und thematischen Gesichtspunkten. Beispielsweise werden auf oberer Ebene die einzelnen Standorte als Bezeichnungen der OUs verwendet und darunter die einzelnen Abteilungen. Auf unterster Ebene erfolgt vielleicht noch die Unterteilung in OUs für Computer- und für Benutzerobjekte.

Eine Beispiel-Organisationseinheitenstruktur – am besten innerhalb einer „Basis“-Organisationseinheit – könnte folgendes Aussehen besitzen:

Ebene 1	Ebene 2	Ebene 3
<b>Standort A</b>	<ul style="list-style-type: none"> <li>✓ Untergeordnete OU 1</li> <li>✓ Untergeordnete OU 2</li> <li>✓ Untergeordnete OU 3</li> <li>✓ Untergeordnete OU n</li> </ul>	<i>Mögliche weitere Verschachtelung, z. B. jeweils</i> <ul style="list-style-type: none"> <li>✓ Rechner</li> <li>✓ Benutzer</li> </ul>
<b>Standort B</b>	<ul style="list-style-type: none"> <li>✓ Untergeordnete OU 1</li> <li>✓ Untergeordnete OU 2</li> <li>✓ Untergeordnete OU 3</li> <li>✓ Untergeordnete OU n</li> </ul>	<i>Mögliche weitere Verschachtelung, z. B. jeweils</i> <ul style="list-style-type: none"> <li>✓ Rechner</li> <li>✓ Benutzer</li> </ul>
<b>Standort C</b>	<ul style="list-style-type: none"> <li>✓ Untergeordnete OU 1</li> <li>✓ Untergeordnete OU 2</li> <li>✓ Untergeordnete OU 3</li> <li>✓ Untergeordnete OU n</li> </ul>	<i>Mögliche weitere Verschachtelung, z. B. jeweils</i> <ul style="list-style-type: none"> <li>✓ Rechner</li> <li>✓ Benutzer</li> </ul>

Wenn die Bezeichnungen der Organisationseinheiten der zweiten Ebene immer gleich sein sollen, wie z. B. die Namen von Abteilungen, benötigen Sie nur noch eine Zulieferung der Namen der Standorte.

Das folgende Beispiel soll umgesetzt werden:

- ✓ Es existiert eine CSV-Datei mit einer Liste von Standorten, die eigene Organisationseinheiten erhalten sollen. Sie besteht nur aus dem Namen des Standorts, einer kurzen Beschreibung und einem Standortkürzel, das im Unternehmen verwendet wird. Da sie mit Microsoft Excel und deutschen Einstellungen erstellt wurde, verfügt sie standardmäßig über das Semikolon als Trennzeichen der Daten.
- ✓ Jeder Standort soll die fünf Abteilungen *Verwaltung, Einkauf, Produktion, Verkauf* und *Leitung* als untergeordnete Organisationseinheiten erhalten.
- ✓ In den weiteren Beispielen wird davon ausgegangen, dass Sie die Basis-Organisationseinheit „Firma“ nennen. Sollten Sie eine andere Bezeichnung wählen, müssen Sie die Beispiele entsprechend anpassen.

	A	B	C
1	Name	Description	Sto
2	Hamburg	Standort Hamburg	HH
3	Bremen	Standort Bremen	HB
4	Bremen-2	Standort Bremen 2	HB2
5	Hannover	Standort Hannover	H
6	Berlin	Standort Berlin	B
7	Hildesheim	Firmenzentrale	HI

CSV-Datei „OU-Vorlage.csv“

### Umsetzung

**Plus** Beispieldatei: *OU-Vorlage.csv, OU-Import.ps1* (Ordner Kapitel 14)

Die CSV-Datei *OU-Vorlage.csv* liegt vor. Da die zweite Ebene der Organisationseinheiten fest definiert ist, werden die Bezeichnungen im folgenden Skript definiert:

```

① try {
    $Eingabe = Read-Host -Prompt "Name der neuen OU (angelegt im Stamm
        der Domain; Abbruch mit STRG+C)"
    New-ADOrganizationalUnit -Name $Eingabe
}
catch {
    Write-Host "Etwas ist schief gegangen (OU existiert bereits?) Der
        Vorgang wird abgebrochen..."
    break
}
② $DNBasisOU = (Get-ADOrganizationalUnit -Filter 'Name
    -eq $Eingabe').DistinguishedName
③ $ImportOU = Import-Csv .\OU-Vorlage.csv -UseCulture
④ $Struktur = @("Verwaltung", "Einkauf", "Produktion", "Verkauf", "Leitung")
⑤ ForEach ($NeueOU in $ImportOU)
{
    ⑥     New-ADOrganizationalUnit -Name $NeueOU.Name -Path $DNBasisOU
        -Description $NeueOU.Description
    ⑦     $SubPath = "OU=" + $NeueOU.Name + "," + $DNBasisOU
    ⑧     ForEach ($Eintrag in $Struktur)
    {
        ⑨         New-ADOrganizationalUnit -Name ($NeueOU.Sto + "-" + $Eintrag)
            -Path $SubPath -Description "Abteilung $Eintrag des Standorts"
    }
}

```

Beispielskript „OU-Import.ps1“

- ① Mit der try{}-catch{}-Konstruktion können in der PowerShell Fehler abgefangen werden. Es handelt sich dabei um einen örtlich begrenzten Fehlerhandler. Die im try{}-Block enthaltenen Befehle werden überwacht. Sollte bei der Ausführung ein Fehler auftreten, erscheinen nicht die üblichen Fehlermeldungen der PowerShell, sondern es werden die Befehle des catch{}-Blocks ausgeführt.  
Im Beispiel wird im try{}-Block als Eingabe des Anwenders der Name einer Basis-Organisationseinheit eingefordert. Mit dem Cmdlet New-ADOrganizationalUnit wird die OU dann angelegt. Liefert dies einen Fehler, weil z. B. die OU bereits besteht, wird der catch{}-Block ausgeführt. Der catch{}-Block besteht aus einer Meldung, die angezeigt werden soll, und dem Befehl break, der die Ausführung des restlichen Skripts abbricht. Die weiteren Skriptzeilen basieren auf der Angabe einer Basis-OU und sollten daher nicht ausgeführt werden.
- ② Die Variable \$DNBasisOU wird erstellt. Der definierte Name der Basis-OU, die unter ① vom Anwender eingegeben wurde, wird der Variablen als Wert zugewiesen.
- ③ Die bereitgestellte Standortliste im CSV-Format wird eingelesen und für die weitere Verarbeitung in der Variablen \$ImportOU gespeichert. Der Parameter –UseCulture sorgt dafür, dass das Semikolon als deutsches Trennzeichen akzeptiert wird und das Einlesen problemlos funktioniert.
- ④ Da in den einzelnen Standorten eine gleichförmige untergeordnete Organisationseinheiten-Struktur aufgebaut werden soll, werden die Namen der OUs als Werte der Array-Variablen \$Struktur zugewiesen.
- ⑤ Mithilfe einer ForEach-Schleife erfolgt ein Durchlauf durch alle sechs in der CSV-Datei enthaltenen Datensätze. Bei sechs angegebenen Standorten erfolgen demnach sechs Schleifendurchläufe.
- ⑥ Für jeden Datensatz (also Standort) wird eine neue OU unterhalb der interaktiv eingegebenen Basis-OU der Domäne angelegt. Name und Beschreibung werden direkt den Daten der CSV-Datei entnommen, als Pfadangabe der Wert der unter ② erstellten Variablen.

- ⑦ Die untergeordneten OUs werden unter den Standorten angelegt. Aus diesem Grund wird eine Variable \$SubPath definiert, die als Wert den definierten Namen der OU des jeweiligen Standorts enthält. Geht man von einer Basis-OU namens *Firma* aus, würde die Variable bei einem Standort *Stuttgart* den Wert OU=Stuttgart,OU=Firma,DC=herdt,DC=zwo annehmen. Sie entspricht damit dem Pfad im Active Directory, an dem die untergeordneten OUs erstellt werden sollen.
- ⑧ An einer Stelle, an der bereits die OU für den jeweiligen Standort angelegt wurde, wird eine weitere ForEach-Schleife zur Erstellung der untergeordneten OUs für jeden Standort verwendet. Sie greift direkt auf die unter ④ definierte Variable \$Struktur zu. In der Variablen befinden sich fünf Werte, es werden also für jeden Standort fünf untergeordnete OUs erstellt.
- ⑨ Die untergeordneten OUs werden erstellt. Der Name jeder neuen OU setzt sich aus dem Standortkürzel aus der CSV-Datei, einem Bindestrich und den Namen der aktuellen Abteilung zusammen. Die jeweils korrekte Pfadangabe wurde unter ⑦ definiert. Die Beschreibung besteht aus einem Standardtext, der den Namen der aktuellen Abteilung verwendet.

Mit diesem Beispiel erstellen Sie auf Tastendruck 36 Organisationseinheiten. Alle OUs sind vor versehentlichem Löschen geschützt. Falls Sie das ändern wollen, ergänzen Sie die Cmdlets New-ADOrganizationalUnit um -ProtectedFromAccidentalDeletion:\$False.

Name	Typ	Beschreibung
HB-Einkauf	Organisationseinheit	Abteilung Einkauf des Standorts
HB-Leitung	Organisationseinheit	Abteilung Leitung des Standorts
HB-Produktion	Organisationseinheit	Abteilung Produktion des Standorts
HB-Verkauf	Organisationseinheit	Abteilung Verkauf des Standorts
HB-Verwaltung	Organisationseinheit	Abteilung Verwaltung des Standorts

Neue Organisationseinheiten-Struktur

Falls Sie beim Testen die erstellten OUs wieder löschen möchten, um z. B. das Skript zu verändern und mehrfach auszuführen, entfernen Sie den Löschschutz und die OUs. Mit der PowerShell erreichen Sie dies durch die folgende Eingabe:

```
Get-ADOrganizationalUnit -Filter {Name -ne "Firma"} -SearchBase
"OU=Firma,DC=herdt,DC=zwo" | Set-ADOrganizationalUnit
-ProtectedFromAccidentalDeletion:$False
```

Wollen Sie die komplette Struktur mit einer Erweiterung der gezeigten Eingabe gleich entfernen, verwenden Sie z. B. die folgende Eingabe:

```
Get-ADOrganizationalUnit -Filter {Name -ne "Firma"} -SearchBase
"OU=Firma,DC=herdt,DC=zwo" | Set-ADOrganizationalUnit
-ProtectedFromAccidentalDeletion:$False -PassThru | Remove-ADObject
-Recursive -Confirm:$False
```

## 14.4 Automatisierter Import neuer Benutzer

### Ausgangssituation

Der Import einer größeren Anzahl von Benutzerkonten stellt eine große Arbeitserleichterung dar. Ob Sie eine Datenbanklösung bevorzugen und sich eine passende Exportroutine programmieren oder eine einfache Excel-Datei, in der Sie die Überschriften gemäß Ihren Vorstellungen festgelegt haben, ist dabei unerheblich.

Wichtig dabei ist, sich vor der Umsetzung ein Konzept zu erarbeiten, um nicht nachträglich viel Arbeit investieren zu müssen, weil die gelieferten Daten nicht Ihren Vorgaben entsprachen. Testen Sie vorher, was Sie in welcher Form wirklich benötigen.

Wenn Sie eine Excel-Vorlage einsetzen, die im Format CSV an Sie zur Weiterbearbeitung übergeben wird, achten Sie auf die Benennung der Überschriften. Die Spaltenköpfe sind für den Import über die PowerShell so zu benennen, dass sie den Eigenschaften eines Benutzerobjekts entsprechen. Die Bezeichnungen für Benutzer finden Sie im Verwaltungstool *Active Directory-Benutzer und -Computer* auf der Registerkarte *Attribut-Editor*.



Füllen Sie die Eigenschaften eines Benutzers komplett aus und orientieren Sie sich dann an der Registerkarte *Attribut-Editor* um herauszufinden, welchen Eigenschafts-Namen Sie in der PowerShell verwenden müssen.

Sie erhalten von der Personalabteilung die gemäß Ihren Vorgaben ausgefüllte Datei *Verwaltung-Neue-MA.csv* (siehe Beispieldateien des Kapitels) mit Daten zu 100 neuen Verwaltungs-Mitarbeitern der einzelnen Standorte. In diesem Beispiel verwenden Sie die Organisationseinheiten-Struktur, die Sie im letzten Abschnitt aufgebaut haben.

Ihre Aufgabe besteht darin, für die neuen Mitarbeiter Benutzerkonten im Active Directory anzulegen.

### Umsetzung



Beispieldatei: *Verwaltung-Neue-MA.csv* (Ordner Kapitel 14)

Sie prüfen zuerst, ob die CSV-Datei verwertbare Überschriften verwendet. Wie in der nachfolgenden Abbildung zu sehen, sind die Überschriften identisch mit den Namen der Eigenschaften von Benutzerobjekten im Active Directory.

```
Verwaltung-Neue-MA.csv
1 name;givenname;surname;samaccountname;path;streetaddress;postalcode;city;TempPass
2 Adrian Zimmermann;Adrian;Zimmermann;adrian.zimmermann;OU=HB-Verwaltung,OU=Bremen,OU
19;28195;Bremen;Kennw0rt!-001
3 Alexander Ziegler;Alexander;Ziegler;alexander.ziegler;OU=HH-Verwaltung,OU=Hamburg,C
42;20095;Hamburg;Kennw0rt!-002
4 Alina Wolff;Alina;Wolff;alina.wolff;OU=HT-Verwaltung,OU=Hildesheim,OU=Firma,DC=herd
```

CSV-Vorlagedatei mit Angaben zu neuen Mitarbeitern (Ausschnitt)

Beim Einlesen einer CSV-Datei werden die Werte den entsprechend positionierten Überschriften zugewiesen. So ist im Beispiel *Adrian Zimmermann* als Wert der Eigenschaft *name* verfügbar. *Name* ist ebenso eine Eigenschaft eines Benutzerobjekts im Active Directory.



Werden in der CSV-Datei andere Überschriften verwendet als benötigt, ändern Sie die Überschriften direkt in der CSV-Datei. Dann kann der Import automatisiert erfolgen. Alternativ können Sie in einem Skript die Daten unter einem Namen einlesen, eventuell verändern und dann unter der benötigten Bezeichnung für den Import in das Active Directory bereitstellen.

### Schritt 1: CSV-Datei importieren

Sie prüfen, ob der Import der CSV-Datei in die PowerShell problemlos funktioniert. Beim Import erhalten Sie eine Ausgabe in der Konsole, anhand derer Sie beurteilen können, ob der Import korrekt vollzogen wurde.

Handelt es sich um eine Standard-CSV-Datei, genügt bereits der folgende Befehl:

```
Import-Csv .\Verwaltung-Neue-MA.csv
```

Im Beispiel erhalten Sie allerdings die folgende Ausgabe. Der Import hat wegen der unerwarteten Daten-Trennzeichen nicht korrekt funktioniert. Die Vorlagedatei verwendet das Semikolon als Trennzeichen, nicht das von der PowerShell erwartete Komma.

```
C:\Users\Administrator\Documents\MyFunctions
Shell > Import-Csv .\Verwaltung-Neue-MA.csv
name;givenname;surname;samaccountname;path;streetaddress;postalcode;city;TempPass
-----
Adrian Zimmermann;Adrian;Zimmermann;adrian.zimmermann;OU=HB-Verwaltung
Alexander Ziegler;Alexander;Ziegler;alexander.ziegler;OU=HH-Verwaltung
```

Ergänzen Sie den Befehl um einen Parameter, der auf ein anderes Trennzeichen in der Vorlagedatei hinweist. Hierbei verwenden Sie entweder die direkte Angabe, welches Trennzeichen zu erwarten ist (a), oder geben an, dass es sich um das im aktuellen Kulturreis definierte Trennzeichen handelt (b):

- a) Import-Csv .\Verwaltung-Neue-MA.csv -Delimiter ";"
- b) Import-Csv .\Verwaltung-Neue-MA.csv -UseCulture

Sie erhalten die folgende Ausgabe. Die Zuordnung der Überschriften zu den einzelnen Werten hat funktioniert. Im Beispiel wird offenbar eine CSV-Datei mit einer Kodierung verwendet, die nicht erkannt wird. Dies sehen Sie an der Ausgabezeile *streetaddress*, in der das deutsche Sonderzeichen (ß) nicht korrekt angezeigt wird. Dieses Verhalten trifft auf alle deutschen Umlaute zu:

```
Shell > Import-Csv .\Verwaltung-Neue-MA.csv -Delimiter ";"

name          : Adrian Zimmermann
givenname     : Adrian
surname       : Zimmermann
samaccountname: adrian.zimmermann
path          : OU=HB-Verwaltung,OU=Bremen,OU=Firma,DC=herdt,DC=ps
streetaddress : Buchenlandstraße 19
postalcode    : 28195
city          : Bremen
TempPass      : Kennw0rt!-001
```

Die PowerShell kennt einen Ausweg, ohne die Datei mit verschiedenen Programmen und Kodierungen zu speichern, bis alle Sonderzeichen korrekt dargestellt werden. Das Cmdlet `Import-Csv` unterstützt den Parameter `-Encoding`, der u. a. die Werte ASCII, Default, Unicode, UTF7 und UTF8 unterstützt.

Auch wenn in der Literatur der Tipp gegeben wird, beim Einlesen deutscher Sonderzeichen `-Encoding UTF8` zu verwenden, funktioniert das im Beispiel nicht. Sie müssen im Beispiel `-Encoding Default` verwenden. Dies kann bei anderen Beispielen ebenfalls passieren: Notfalls probieren Sie die wenigen Einstellungen und kommen so auf jeden Fall zum Ziel:

```
Import-Csv .\Verwaltung-Neue-MA.csv -UseCulture -Encoding Default
```

```
Shell > Import-Csv .\Verwaltung-Neue-MA.csv -Delimiter ";" -Encoding Default

name          : Adrian Zimmermann
givenname     : Adrian
surname       : Zimmermann
samaccountname: adrian.zimmermann
path          : OU=HB-Verwaltung,OU=Bremen,OU=Firma,DC=herdt,DC=zwo
streetaddress : Buchenlandstraße 19
postalcode    : 28195
city          : Bremen
TempPass      : Kennw0rt!-001
```

### Schritt 2: Importierte Daten verarbeiten

Wenn die Daten so gut aufbereitet vorliegen und Sie keine Änderungen an Inhalt bzw. Zuordnung zu Eigenschaften der zukünftigen Active Directory-Objekte vornehmen wollen, sollten Sie an dieser Stelle die Pipeline einsetzen ("|").

Im einfachsten Fall verwenden Sie das Cmdlet `New-ADUser` ohne weitere Parameter. Die nötigen Angaben sind ja bereits in der CSV-Datei vorhanden:

```
Import-Csv .\Verwaltung-Neue-MA.csv -UseCulture -Encoding Default | New-ADUser
```

Der Befehl erstellt Benutzerkonten in den Verwaltungs-Organisationseinheiten der einzelnen Standorte (24 in Hildesheim, 20 in Bremen und jeweils 14 in Berlin, Bremen-2, Hamburg und Hannover).

Soweit funktioniert alles wie geplant. Weitere Eigenschaften können Sie entweder in die CSV-Vorlagendatei eingeben oder mithilfe des Cmdlets `Set-ADUser` nachträglich ändern. Allerdings haben Sie den Benutzern noch kein Kennwort zugewiesen. Alle Konten sind deaktiviert.

Hinweis: Da Sie die Befehle mehrfach ausführen, ist es zur Vermeidung von Fehlermeldungen hilfreich, wenn Sie die bereits erstellten Benutzer zwischendurch wieder löschen. Der Befehl löscht alle Benutzerkonten, die sich in der Basis-Organisationseinheit *Firma* befinden:

```
Get-ADUser -Filter * -SearchBase "OU=Firma,DC=herdt,DC=zwo" | Remove-ADUser
```

Ein Massenimport von Benutzerkonten macht nur dann wirklich Sinn, wenn Sie mit den Benutzerkonten sofort arbeiten können. Aus diesem Grund importieren Sie nun die Benutzer aus der CSV-Datei erneut und nehmen an den Konten weitere Einstellungen vor:

- ✓ Sie vergeben ein temporäres Kennwort für die neuen Benutzerkonten.
- ✓ Sie aktivieren die Benutzerkonten.
- ✓ Sie weisen die neuen Benutzer an, das Kennwort bei der nächsten Anmeldung zu ändern.

Der Befehl wird um die entsprechenden Parameter verlängert:

```
Import-Csv .\Verwaltung-Neue-MA.csv -UseCulture -Encoding Default | 
New-ADUser -AccountPassword (ConvertTo-SecureString -AsPlainText 
"Kennw0rt!" -Force) -Enabled:$True -ChangePasswordAtLogon:$True
```

Damit haben Sie das Ziel erreicht: Insgesamt 100 Benutzer sind in den unterschiedlichen Organisationseinheiten angelegt worden. Alle Konten wurden aktiviert und haben ein temporäres Kennwort erhalten, das der Benutzer bei der nächsten Anmeldung ändern muss.

## Schritt 2 (alternativ): Importierte Daten nachbearbeiten



**Beispieldatei:** Benutzerimport-Alternative.ps1 (Ordner Kapitel 14)

Falls Sie der Meinung sind, dass Daten der CSV-Datei nachbearbeitet werden müssen und nicht direkt übernommen werden können, schreiben Sie ein Skript zur Nachbearbeitung der Daten. Das Skript importiert die CSV-Datei, verändert die Daten und erstellt die Benutzerkonten im Active Directory.

Angenommen, Ihre Firma ist aus Sicherheitserwägungen nicht einverstanden damit, dass alle neuen Mitarbeiter ein identisches temporäres Kennwort erhalten. In der Beispieldatei *Verwaltung-Neue-MA.csv* wurde eine Spalte *TempPass* hinzugefügt. Sie enthält individuelle Kennwörter für die einzelnen Benutzer, die Sie für Übungszwecke aber noch leicht nachvollziehen können.

Die in der CSV-Datei angegebenen Kennwörter können nicht direkt gespeichert werden, da sie erst in eine sichere Zeichenfolge umgewandelt werden und dann als Wert der Eigenschaft *AccountPassword* eines Benutzerobjekts gespeichert werden.

Nachfolgend sehen Sie das Listing eines Skripts, das individuelle Kennwörter vergibt und eine individuelle Beschreibung des Kontos vornimmt:

```
Import-Csv -Path .\Verwaltung-Neue-MA.csv -UseCulture -Encoding Default
|
① ` ForEach-Object {
②     Write-Host "Lege Benutzer" $_.Name "an..."
③     If($_.City -eq "Hildesheim")
    {
        $StoText = "Zentrale --- "
    }
    Else
    {
        $StoText = "Filiale --- "
    }
④     New-ADUser -Name $_.Name `-
        -GivenName $_.GivenName `-
        -Surname $_.Surname `-
        -SamAccountName $_.sAMAccountName `-
        -Path $_.Path `-
        -StreetAddress $_.StreetAddress `-
        -PostalCode $_.PostalCode `-
        -City $_.City `-
        -AccountPassword (ConvertTo-SecureString -AsPlainText
            ($_.TempPass) -Force) `-
        -Description ($StoText + "Temporäres Kennwort: " + $_.TempPass) `-
        -Enabled:$True `-
        -ChangePasswordAtLogon:$True
    }
}
```

*Beispieldatei „Benutzerimport-Alternative.ps1“*

- ① Der Import läuft im Prinzip ab wie bei der direkten Eingabe. Da aber zu viele Daten eingegeben werden müssen, fiel die Wahl auf ein PowerShell-Skript. Die importierten Daten werden über die Pipeline an das Cmdlet `ForEach-Object` weitergegeben. Der Skriptblock des Cmdlets beinhaltet alle geplanten Vorgänge für die einzelnen Objekte.

- ② Beim Anlegen der Benutzer wird eine individuelle Meldung ausgegeben. Alternativ könnte z. B. ein Protokoll in eine Datei geschrieben werden.
- ③ Falls der neue Mitarbeiter den Wert Hildesheim für die Eigenschaft *City* aufweist, arbeitet er in der Zentrale, ansonsten in einer Filiale. Es wird eine Variable definiert, die den entsprechenden Wert erhält. Der Text wird später (bei Punkt ⑥) in die Beschreibung des Benutzerobjekts geschrieben.
- ④ Die Erstellung des Benutzerobjekts im Active Directory wird mit dem Cmdlet `New-ADUser` initiiert. Bei dieser Vorgehensweise müssen Sie die in der CSV-Datei enthaltenen Felder selbst angeben.
- ⑤ Das in der CVS-Datei angegebene Kennwort (`$_.TempPass`) wird in eine sichere Zeichenfolge umgewandelt und als Wert der Eigenschaft *AccountPassword* des Benutzerobjekts zugewiesen.
- ⑥ Eine Zeichenfolge wird in die Beschreibung des jeweiligen Benutzerobjekts geschrieben, das – als Service für Ihre Tests – u. a. das temporäre Kennwort enthält.

Je nach Vorgehensweise können Sie Daten direkt oder nach einer Nachbearbeitung in Ihr Active Directory einspielen.

Ein weiteres Beispiel wären die Definition von benötigten Gruppen und die Zuweisung von Gruppenmitgliedschaften.

Um das Beispiel in einem überschaubaren Rahmen zu halten, sind die Pfadangaben der neuen Benutzer in der CSV-Datei fest kodiert und damit nicht flexibel. Über Benutzereingaben statt fester Kodierung, Verkettung von Textelementen zu einem Pfad und anderen im vorliegenden Buch vorgestellten Aktionen lässt sich recht schnell eine flexible Lösung erstellen. Denken Sie in diesem Zusammenhang an die Möglichkeit, den definierten Namen einer Organisationseinheit auszulesen:

```
(Get-ADOrganizationalUnit -Filter 'Name -eq XYZ').DistinguishedName.
```

### Wann entscheiden Sie sich für eine PowerShell-Lösung für die gezeigten Tätigkeiten?

Im Prinzip muss diese Frage von Fall zu Fall entschieden werden. Stellen Sie sich eine Reihe von Fragen, um dann zu entscheiden:

- ✓ Wie oft kommen solche Arbeiten auf Sie zu? Setzen Sie häufiger Testumgebungen oder Netzwerksegmente auf? Oder arbeiten Sie in einer starren Umgebung, die sich kaum ändert?
- ✓ Wie hoch ist der Arbeitsaufwand für die Umsetzung mit der PowerShell? Wie hoch ist der Arbeitsaufwand für die Umsetzung mit anderen Mitteln?
- ✓ Können Sie mit brauchbaren Zulieferungen (z. B. CSV-Dateien) rechnen? Können Sie die Inhalte der zu liefernden CSV-Dateien beeinflussen?
- ✓ Wie aufwendig sind diese Tätigkeiten in Ihrer Umgebung? Was ist zusätzlich zu definieren?
- ✓ Waren in den gezeigten Beispielen geschickte Ansätze, die Sie mit herkömmlichen Mitteln nicht oder nicht so einfach umsetzen konnten?

Wenn zehn Benutzer in Ihrer starren Domäne sind, werden Sie die meisten administrativen Tätigkeiten auf herkömmlichem Weg ausführen. Aber immer dann, wenn eine Automatisierung sinnvoll wäre, sollten Sie an die PowerShell als ein mögliches Arbeitsmittel denken.

# A Anhang: Testumgebung einrichten

## A.1 Download der benötigten Software

### Virtualisierungssoftware Hyper-V

Für die Beispiele im Buch wird die in aktuellen Windows-Betriebssystemen integrierte Virtualisierungslösung *Hyper-V* verwendet. Voraussetzung hierfür ist, dass Hyper-V in Ihrem Hostsystem zur Verfügung steht. Bei Client-Software ist dies der Fall in den 64-bit-Editionen Professional oder Enterprise von Windows 10 (und natürlich als Rolle in den Windows Server-Versionen ab Windows Server 2008).

Die im Skript verwendeten Microsoft-Betriebssysteme können damit als virtuelle Maschinen betrieben werden. Damit sind Sie unabhängig von Ihrem Produktivsystem und können gefahrlos testen.

Fall Sie Hyper-V nicht zur Verfügung haben, können Sie auch auf andere Lösungen zurückgreifen. Zu empfehlen ist u. a. VirtualBox von Oracle. VirtualBox ist eine freie Software, die unter der GNU General Public License (GPL) steht. Die aktuelle Version kann unter der Adresse <https://www.virtualbox.org> im Menü *Downloads* kostenlos heruntergeladen werden. Der direkte Downloadlink lautet: <https://virtualbox.org/wiki/Downloads>. Auf der angegebenen Website finden Sie auch Dokumentationen und ein Forum – jeweils in englischer Sprache.



### Betriebssysteme Windows 10 und Windows Server 2019

Um alle Beispiele im Buch nachzuvollziehen, benötigen Sie die aktuellen Microsoft-Betriebssysteme Windows 10 und Windows Server 2016 (oder 2019). Microsoft stellt beide Betriebssysteme kostenfrei als Evaluationsversion zur Verfügung.

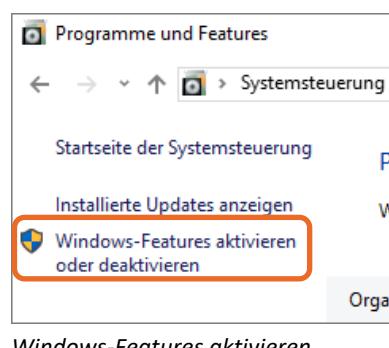
- ✓ Windows 10 Enterprise (90-Tage-Evaluierungsversion)  
Download unter <https://www.microsoft.com/de-de/evalcenter/evaluate-windows-10-enterprise>
- ✓ Windows Server 2019 (180-Tage-Evaluierungsversion)  
Download unter <https://www.microsoft.com/de-de/evalcenter/evaluate-windows-server-2019>



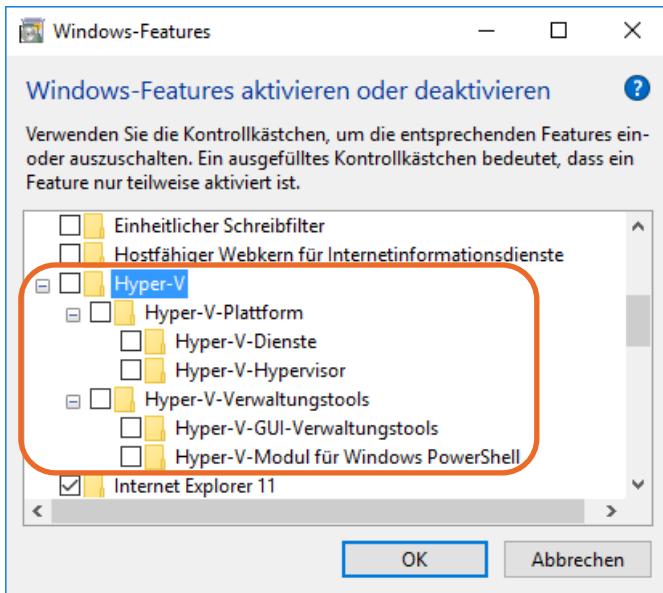
## A.2 Hyper-V unter Windows 10 (Enterprise) aktivieren

Da Hyper-V in den 64-bit-Editionen Professional oder Enterprise von Windows 10 bereits integriert ist, müssen Sie keine weitere Installation durchführen, sondern dieses Windows-Feature nur noch aktivieren.

- Öffnen Sie die Systemsteuerung und rufen den Eintrag *Programme und Features* auf.  
Sie sehen eine Liste der installierten Programme auf Ihrem Rechner.
- Klicken Sie in der Navigation im linken Fensterbereich auf *Windows-Features aktivieren oder deaktivieren*.



*Windows-Features aktivieren*



*Windows-Feature Hyper-V bereit zur Aktivierung*

- ▶ Aktivieren Sie den Eintrag **Hyper-V**.  
Damit werden automatisch alle Untereinträge aktiviert.
- ▶ Bestätigen Sie mit **OK**.  
Es dauert einen Moment, bis der Rechner Hyper-V aktiviert hat. Danach steht Ihnen in der Windows PowerShell das **Hyper-V-Modul** zur Verfügung, und Sie können als Hyper-V-GUI-Verwaltungstool den **Hyper-V-Manager** nutzen.

Der Hyper-V-Manager steht bereits als Kachel im Startmenü zur Verfügung und kann darüber aufgerufen werden. Mit Rechtsklick können Sie die Kachel anpassen, z. B. zur Taskleiste hinzufügen.

Alternativ können Sie im Startmenü nach „Hyper-V“ suchen und erhalten ebenfalls als Suchergebnis den Hyper-V-Manager.



*Startmenü: Kachel und Suchergebnis*

In Windows-Serverbetriebssystemen ab Windows Server 2008 steht Hyper-V zur Verfügung und kann über die üblichen Wege im Server-Manager als Rolle installiert werden.

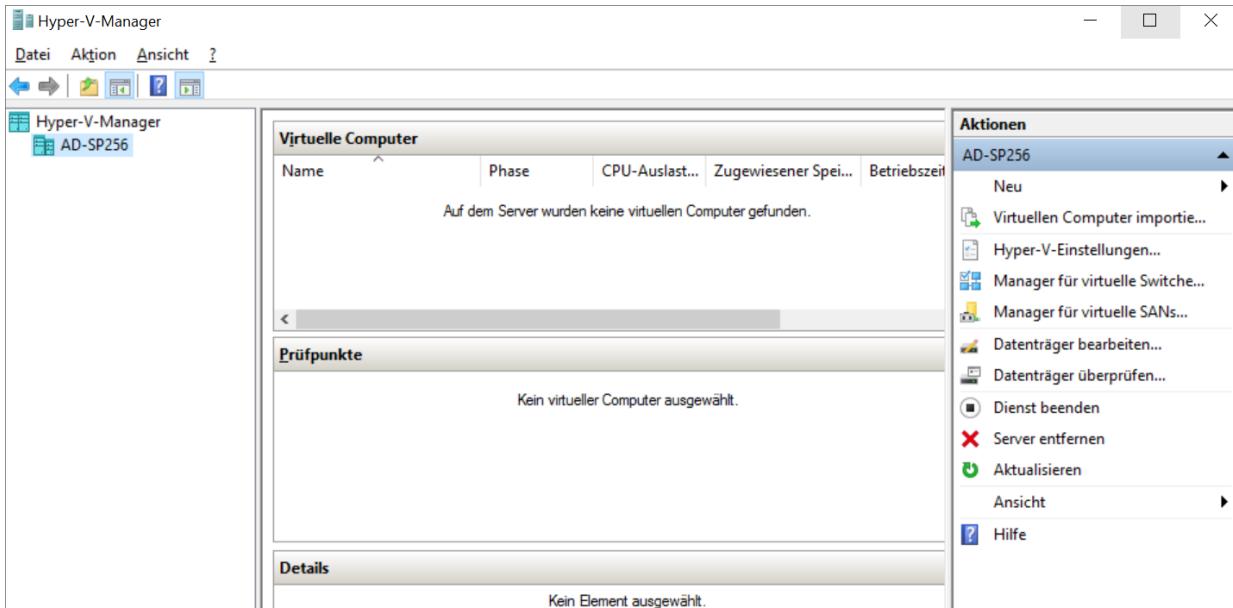


Sorgen Sie dafür, dass Sie vor den weiteren Schritten die Betriebssysteme von Microsoft heruntergeladen haben. Sie benötigen sie als Quelldateien für die Erstellung virtueller Maschinen.

## A.3 Hyper-V einrichten

Nach der Aktivierung von Hyper-V müssen Sie Hyper-V einstellen. Danach können Sie virtuelle Maschinen anlegen.

- Öffnen Sie den Hyper-V-Manager. Sie erhalten beim ersten Start eine recht leere Oberfläche:

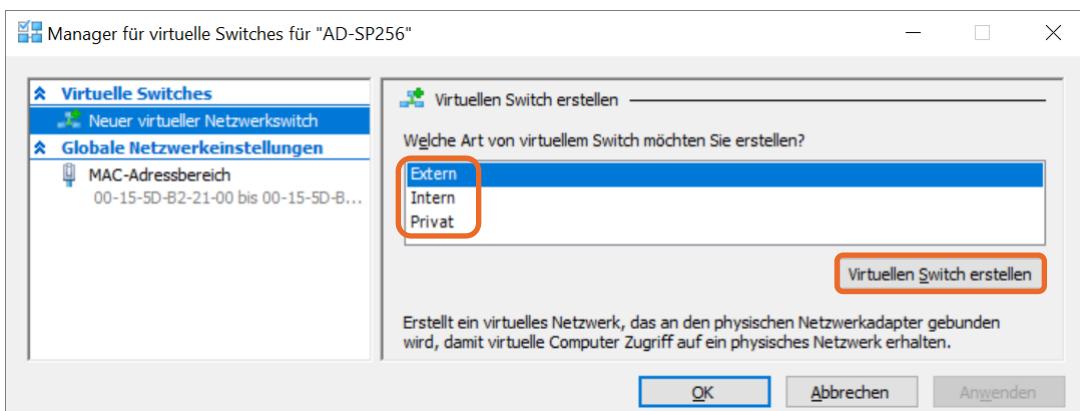
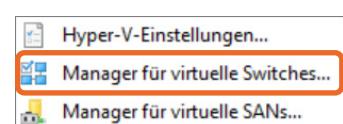


Im oberen Bereich *Virtuelle Computer* werden verfügbare virtuelle Maschinen angezeigt. Der rechte Bereich *Aktionen* unterstützt Sie bei der Konfiguration von Hyper-V.

### Virtuelle Switches konfigurieren

Bevor Sie virtuelle Maschinen anlegen, sollten Sie virtuelle Switches definieren. Als virtuelle Switches bezeichnet Microsoft die Netzwerkadapter, die in einer virtuellen Maschine zur Verfügung stehen können. Bei der Installation virtueller Maschinen können Sie dann gleich einen oder mehrere virtuelle Switches zuweisen und die gewünschte Netzwerkfunktionalität sicherstellen. Ohne eine erste, einmalige Konfiguration eines virtuellen Switches stehen Ihnen in den Gastbetriebssystemen keine Netzwerkadapter zur Verfügung.

- Klicken Sie im Bereich *Aktionen* des Hyper-V-Managers auf *Manager für virtuelle Switches*, um virtuelle Switches neu zu definieren oder zu konfigurieren.

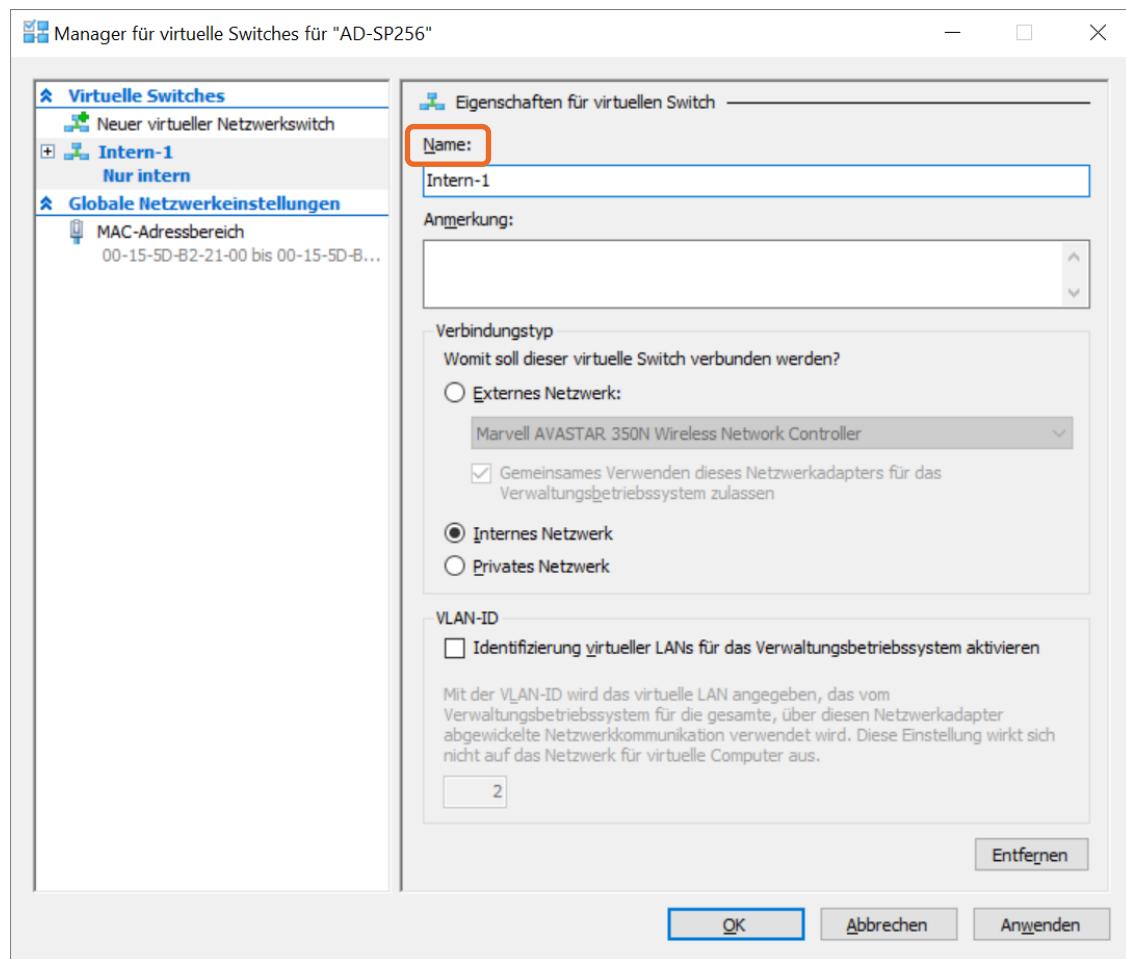


*Manager für virtuelle Switches: Arten von Switches und Erstellen eines Switches*

Es gibt drei verschiedene Arten von virtuellen Switches: *Extern*, *Intern* und *Privat*. Sie können mit einem externen Switch aus Ihrem Gastbetriebssystem auf das Internet zugreifen (sofern Ihr Host-Rechner über einen funktionierenden Internetzugang verfügt), nicht aber auf andere laufende virtuelle Maschinen. Mit internen und privaten Switches hingegen können Sie bei entsprechender IP-Konfiguration auf andere virtuelle Maschinen zugreifen, aber nicht direkt auf das Internet.

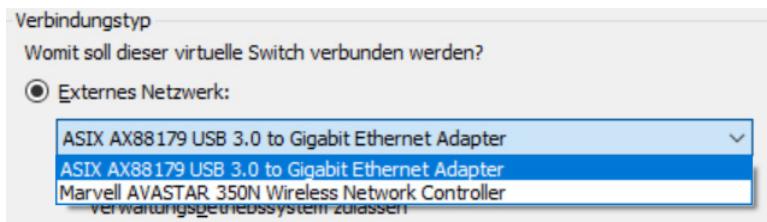
Für die ersten Kapitel, in denen Sie zum Teil Internetzugang benötigen, aber keinen Zugriff auf andere virtuelle Maschinen, benötigen Sie einen externen Switch. Wenn Sie für die Übungen der letzten Kapitel mehrere virtuelle Maschinen im Netzwerkverbund betreiben, aber auch zusätzlich eine Verbindung ins Internet herstellen wollen, definieren Sie für Ihre virtuellen Computer zwei Netzwerkadapter (in Hyper-V einen *externen* sowie einen *internen* oder *privaten*). Der externe Netzwerkadapter wird für den Zugang zum Internet konfiguriert, der interne bzw. private Netzwerkadapter wie in der obigen Abbildung zur netzwerkinternen Kommunikation. Eine Anleitung finden Sie weiter unten in diesem Kapitel.

Um einen internen Switch zu erstellen, reicht es aus, nach Betätigen der Schaltfläche *Virtuellen Switch erstellen*, einen Namen für den Switch einzugeben.



Internen virtuellen Switch erstellen

Wollen Sie einen externen Switch erstellen, geschieht dies analog zur Erstellung eines internen Switches. Der einzige Unterschied besteht darin, dass Sie bei der Verbindung mit dem externen Netzwerk einen Netzwerkadapter Ihres Host-Rechners auswählen müssen, mit dem der virtuelle Switch verbunden werden soll.



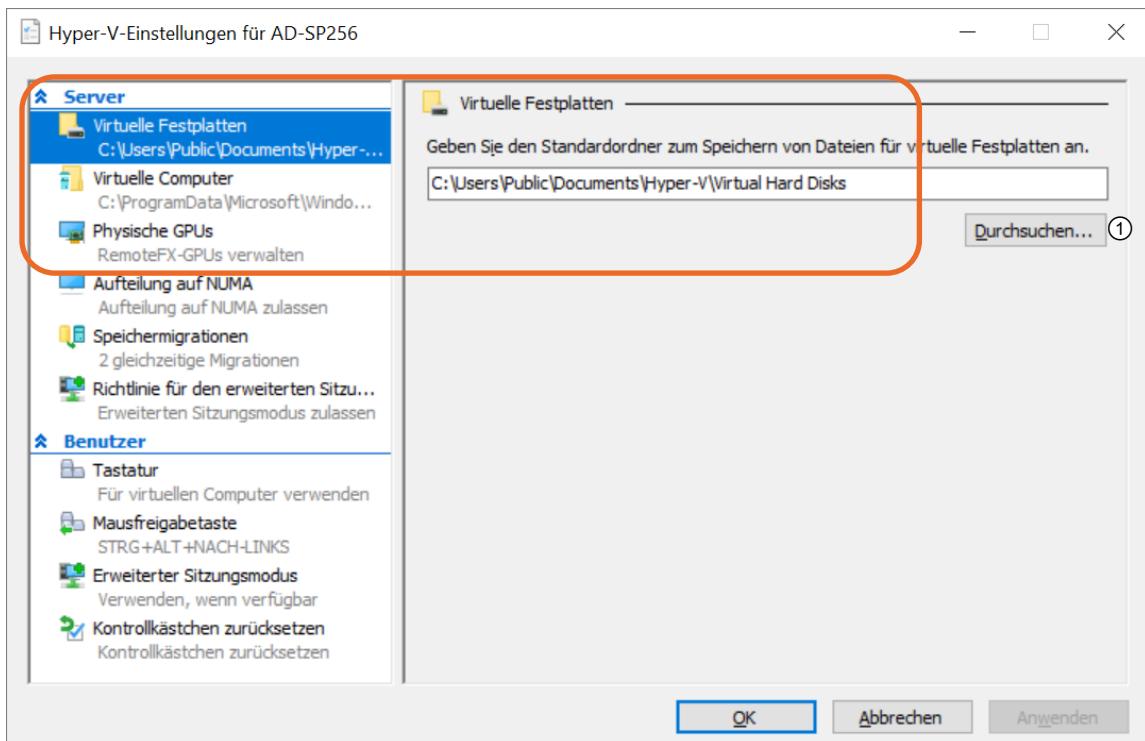
Auswahl des gewünschten physischen Netzwerkadapters zur Verbindung mit dem externen virtuellen Switch

Dies spielt nur dann eine wichtige Rolle, wenn sich in Ihrem Host-Rechner mehrere physische Netzwerkadapter befinden und Sie einen bestimmten wählen wollen/müssen. Verfügt Ihr Host-Rechner nur über einen einzigen Netzwerkadapter, wird er automatisch ausgewählt.

Nach der Erstellung und Konfiguration virtueller Switches stehen diese in Hyper-V für die virtuellen Maschinen zur Verfügung. Weitere vorbereitende Schritte sind nicht nötig. Wie Sie später im Kapitel sehen werden, können Sie bei der Erstellung virtueller Maschinen aus dem Pool definierter virtueller Switches auswählen, welche Switches in der virtuellen Maschine zur Verfügung stehen sollen.

### Speicherort für virtuelle Festplatten anpassen

Standardmäßig speichert Hyper-V die Daten für virtuelle Computer und Festplatten auf demselben Laufwerk wie das Host-System (Laufwerk C:), wie Sie in der folgenden Abbildung sehen können.



Hyper-V-Einstellungen: Speicherort für virtuelle Computer und virtuelle Festplatten

Dies birgt zwei mögliche Probleme: Zum einen sind bei einem Ausfall der physischen Festplatte des Host-Rechners auch die Daten der virtuellen Maschinen betroffen und ggf. verloren, zum anderen empfiehlt es sich aus Platzgründen, v. a. die großen Datenmengen virtueller Festplatten auf ein anderes Laufwerk zu speichern.

Wenn Sie die Möglichkeit haben, sollten Sie aus Platzgründen die viele GB großen virtuellen Festplatten auf ein anderes Laufwerk bzw. auf eine andere physische Festplatte speichern. Noch besser ist es, auch den Speicherort der virtuellen Computer entsprechend anzupassen:

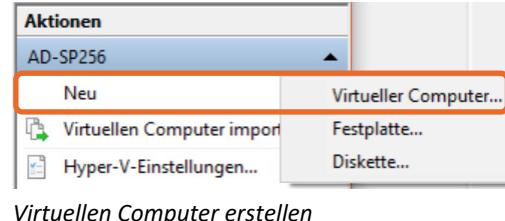
- Klicken Sie auf *Durchsuchen* ① und wählen Sie einen Speicherort auf einem anderen Laufwerk.

Nun sind Ihre virtuellen Maschinen leicht in ein Backup einzubeziehen und v. a. unabhängig von Ihrem Host-System. Bei Bedarf können Sie die virtuellen Computer auf einem anderen Hyper-V-Host importieren und weiter arbeiten.

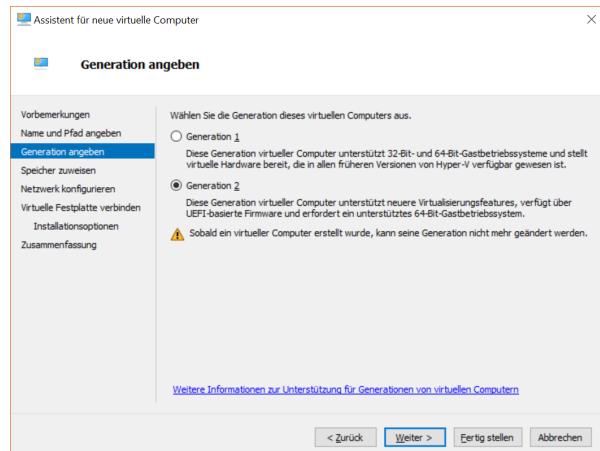
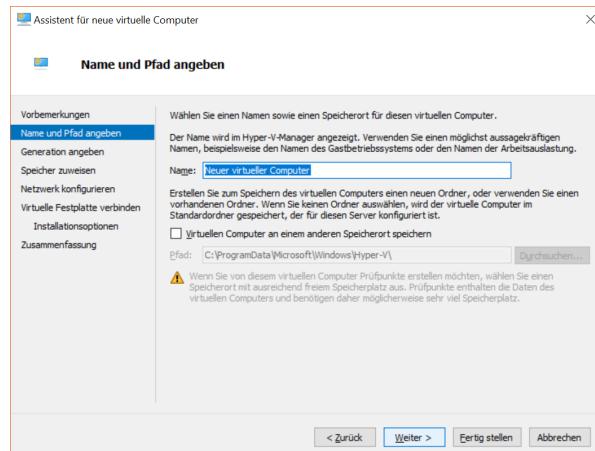
## A.4 Virtuellen Computer erstellen und betreiben

Um virtuelle Computer bzw. virtuelle Maschinen zu erstellen, klicken Sie im Hyper-V-Manager rechts im Bereich *Aktionen* auf *Neu* und wählen den Eintrag *Virtueller Computer*.

Es startet ein Assistent, der Sie durch die nötigen Schritte zum Erstellen eines virtuellen Computers führt. Sie geben den Namen und eventuell einen vom Standardwert abweichenden Speicherort für den virtuellen Computer sowie die sog. „Generation“ des virtuellen Computers an. Generation 2 bietet für neue Betriebssysteme deutlich besseren Benutzerkomfort und sollte daher ausgewählt werden.



*Virtuellen Computer erstellen*



*Assistent für neue virtuelle Computer (1): Angabe von Name und Generation des virtuellen Computers*

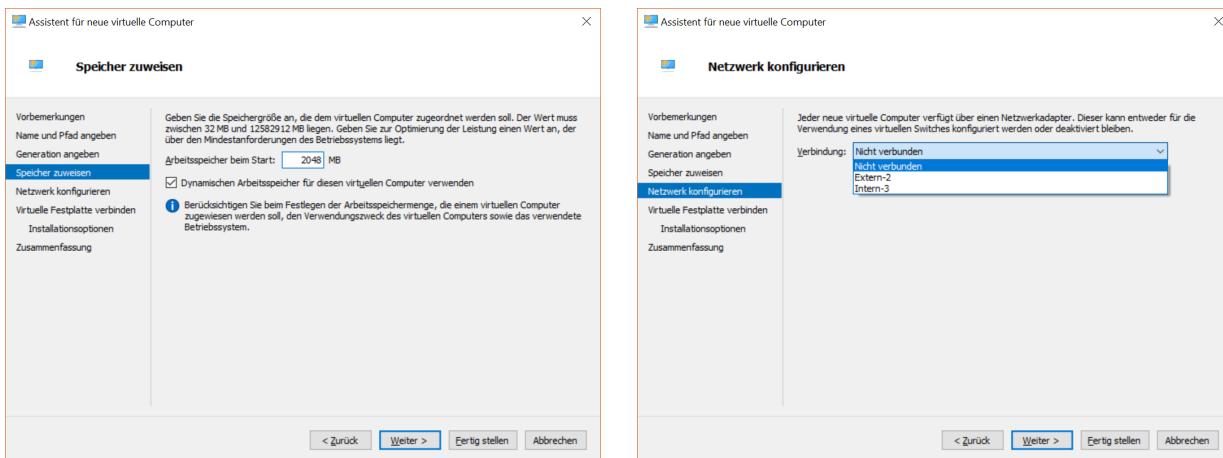
Danach wählen Sie die Größe des Arbeitsspeichers für den virtuellen Computer. Sie legen den Arbeitsspeicher für den Startvorgang des virtuellen Computers fest (z. B. 2048 MB, also 2 GB). Zusätzlich wählen Sie, ob für diesen virtuellen Computer dynamischer Arbeitsspeicher verwendet werden darf. Aktivieren Sie die Option, weist Hyper-V automatisch den benötigten Arbeitsspeicher zu. Verwenden Sie keinen dynamischen Arbeitsspeicher, wird der eingegebene Startwert als fester Arbeitsspeicher für die gesamte Laufzeit des virtuellen Computers verwendet.



Bedenken Sie, dass der für virtuelle Computer verwendete Arbeitsspeicher vom physischen Arbeitsspeicher Ihres Host-Rechners „abgezweigt“ wird. Je nach Anzahl gleichzeitig laufender virtueller Computer und der (begrenzten) Größe des physischen Arbeitsspeichers im Host-Rechner sollten Sie eher sparsam mit der Zuweisung von Arbeitsspeicher für virtuelle Computer umgehen.

Im nächsten Schritt legen Sie fest, welche Netzwerkadapter Sie für den virtuellen Computer aktivieren wollen. Die vorhandene Auswahl zeigt die definierten virtuellen Switches, die Sie weiter oben im Kapitel angelegt haben.

- Um neue virtuelle Computer für die ersten Übungen mit Internetzugang auszustatten (sofern auf dem Host-Rechner vorhanden), wählen Sie den externen Switch.

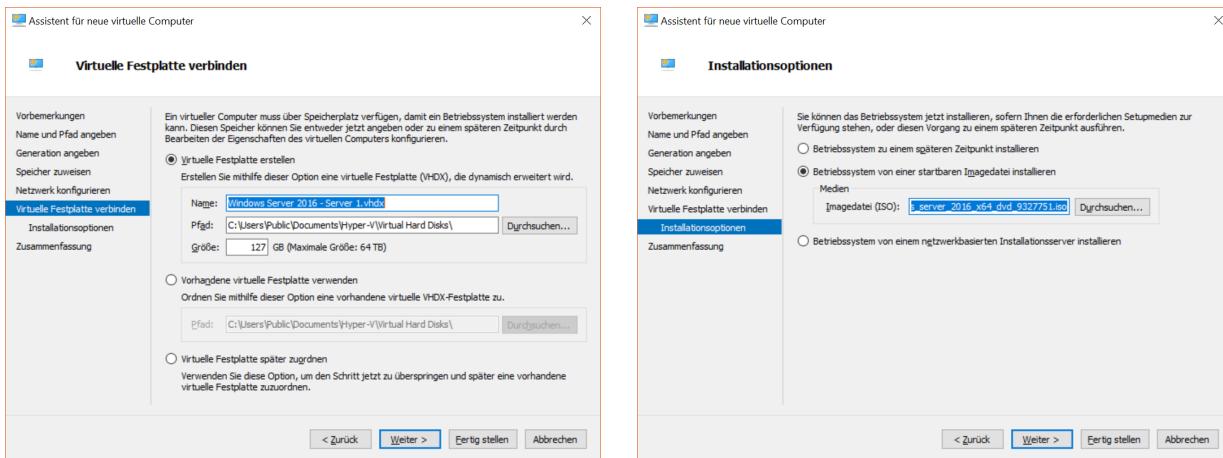


### Assistent für neue virtuelle Computer (2): Konfigurieren von Arbeitsspeicher und Netzwerk

Bei der Konfiguration der virtuellen Festplatte wird der Standardname (Name des virtuellen Computers mit der Dateiendung .vhdx) vorgeschlagen, den Sie auswählen. An dieser Stelle können Sie den Dateipfad und die Größe der Festplatte festlegen. Die Größe der benötigten Festplatte hängt davon ab, welche Software und Daten auf dem virtuellen Computer verwendet werden sollen. Für die benötigte Testumgebung reicht die Standardgröße 127 GB völlig aus.

Virtuelle Computer verhalten sich wie eigenständige Rechner. Beim ersten Start erwarten sie ein Medium, von dem sie ein Betriebssystem installieren (oder zumindest starten) können. Ist eine Installation erfolgt, haben Sie ab diesem Zeitpunkt einen kompletten „Rechner im Rechner“. Sie werden aus diesem Grund bei den Installationsoptionen gefragt, ob und von welchem Medium das Gastbetriebssystem installiert werden soll. Für diese Angabe benötigen Sie die zu Beginn des Kapitels beschriebenen ISO-Dateien der Evaluationsversionen der Betriebssysteme.

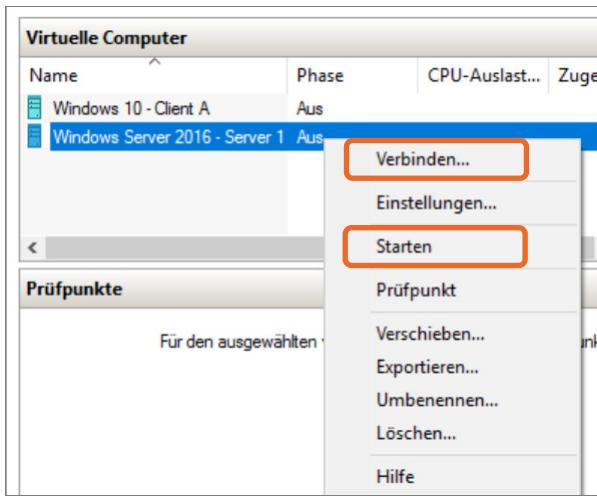
- Aktivieren Sie die Option *Betriebssystem von einer startbaren Imagedatei installieren* und wählen die von Microsoft heruntergeladene ISO-Datei für Windows 10 bzw. Windows Server 2019 aus.



### Assistent für neue virtuelle Computer (3): Einstellungen zur virtuellen Festplatte und der Betriebssystem-Quelle

Auf einer weiteren Seite des Assistenten erhalten Sie eine Zusammenfassung der von Ihnen vorgenommenen Einstellungen zu Name, Generation, Arbeitsspeicher, Netzwerk, Festplatte und Betriebssystem des neuen virtuellen Computers. Durch einen Klick auf die Schaltfläche *Fertig stellen* beenden Sie den Assistenten, und der virtuelle Computer wird erstellt.

Zusätzlich wird beim ersten Start des virtuellen Computers die Installation des Gastbetriebssystems vorgenommen. Damit haben Sie eine oder mehrere virtuelle Computer definiert und in ihnen jeweils ein Betriebssystem installiert.



*Hyper-V-Manager: Virtuelle Computer wurden definiert (Menü erscheint bei Rechtsklick)*

- ▶ Um einen virtuellen Computer zu starten, klicken Sie mit der rechten Maustaste auf den gewünschten virtuellen Computer und wählen Sie im Kontextmenü **Starten**.

Der virtuelle Computer wird zwar gestartet, es öffnet sich allerdings kein Fenster, in dem Sie mit dem Computer arbeiten können. Hyper-V ist als eigentliche Serverkomponente auf den Betrieb virtueller Maschinen ausgelegt, bei der Sie nicht stets selbst ein Fenster des Gastes sehen wollen.

- ▶ Um eine Konsole mit dem virtuellen Computer zu öffnen, wählen Sie in dem Kontextmenü den Eintrag **Verbinden**.
- ▶ Starten Sie Ihre virtuellen Computer und lassen Sie die Installation der Betriebssysteme ablaufen. Belassen Sie es bei einer Standardinstallation. Eine Eingabe eines Produktschlüssels für das Betriebssystem ist bei einer Evaluationsversion nicht erforderlich. Sie können im angegebenen Zeitraum ohne Einschränkungen testen.
- ▶ Vergeben Sie nach erfolgter Installation Kennwörter für Ihre Benutzer und ändern Sie nach eigenen Vorstellungen den Namen des Rechners.

## A.5 Testanordnung für die Übungen

### Teil I: Kapitel 1 bis 8

Im ersten Teil des Buches (Kapitel 1 bis 8) arbeiten Sie mit einem alleinstehenden Rechner. Ob Sie sich für einen virtuellen Computer mit Windows 10 oder Windows Server 2016 (oder 2019) entscheiden, spielt dabei keine Rolle. In diesen Kapiteln geht es um Grundlagen zur PowerShell. Beide Betriebssysteme verfügen über die Windows PowerShell Version 5.1.

Diese virtuellen Computer sollten Sie mit einem externen Switch ausstatten – wie oben beschrieben. Damit verhält sich der virtuelle Computer so, als wäre er Ihr Hostrechner. Mehrere virtuelle Computer sehen sich zwar untereinander nicht, haben dafür aber Internetzugriff, sofern Ihr Hostrechner darüber verfügt.



Ändern Sie für die Arbeit im ersten Teil des Buches **nicht** die Einstellungen des Netzwerkadapters der virtuellen Maschine und vergeben Sie **keine** IP-Adresse. Oder sorgen Sie alternativ für eine andere Lösung für den Internetzugang Ihrer virtuellen Maschinen, z. B. durch die Konfiguration eines zweiten Netzwerkadapters mit entsprechender Konfiguration oder den Einsatz eines Routers. Auf andere alternative Lösungen wird an dieser Stelle nicht weiter eingegangen.

## Teil II: ab Kapitel 9

Ab Kapitel 9 benötigen Sie das Betriebssystem Windows Server 2016 (oder 2019). Sie arbeiten mit mehreren virtuellen Maschinen, die sich in einer gemeinsamen Domäne befinden. Die minimale Testumgebung besteht aus zwei virtuellen Maschinen: einem Domänencontroller (Windows Server 2016 oder 2019) und einem Domänen-Client (Windows 10).

Im Folgenden werden die nötigen Änderungen für den zweiten Teil des Buches beschrieben.

## A.6 Mehrere virtuelle Computer in einem gemeinsamen virtuellen Netzwerk betreiben

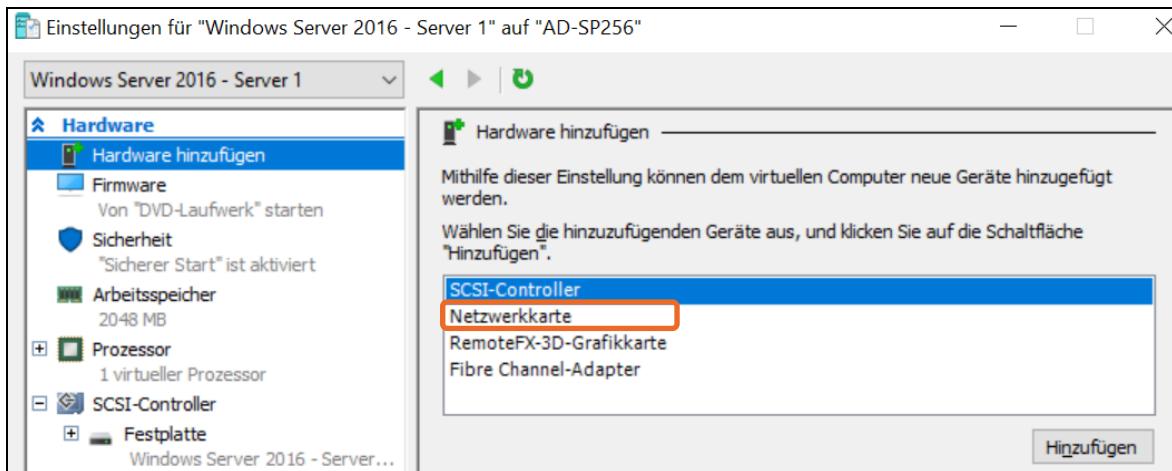
### Dasselbe virtuelle Netzwerk wählen

Da der Netzwerkadapter virtueller Computer für die ersten Kapitel mit einem externen Switch ausgestattet wurde, sehen sich mehrere virtuelle Computer untereinander nicht. Alle verhalten sich gleichberechtigt neben-einander, als seien sie der Hostrechner, auf dem sie installiert sind.

Für die Übungen im zweiten Teil des Buches sollen sich aber mehrere virtuelle Computer in einem einzigen virtuellen Netzwerk befinden, einander sehen und zusammenarbeiten. Dies ist möglich, es sind nur wenige Schritte Ihrerseits vonnöten:

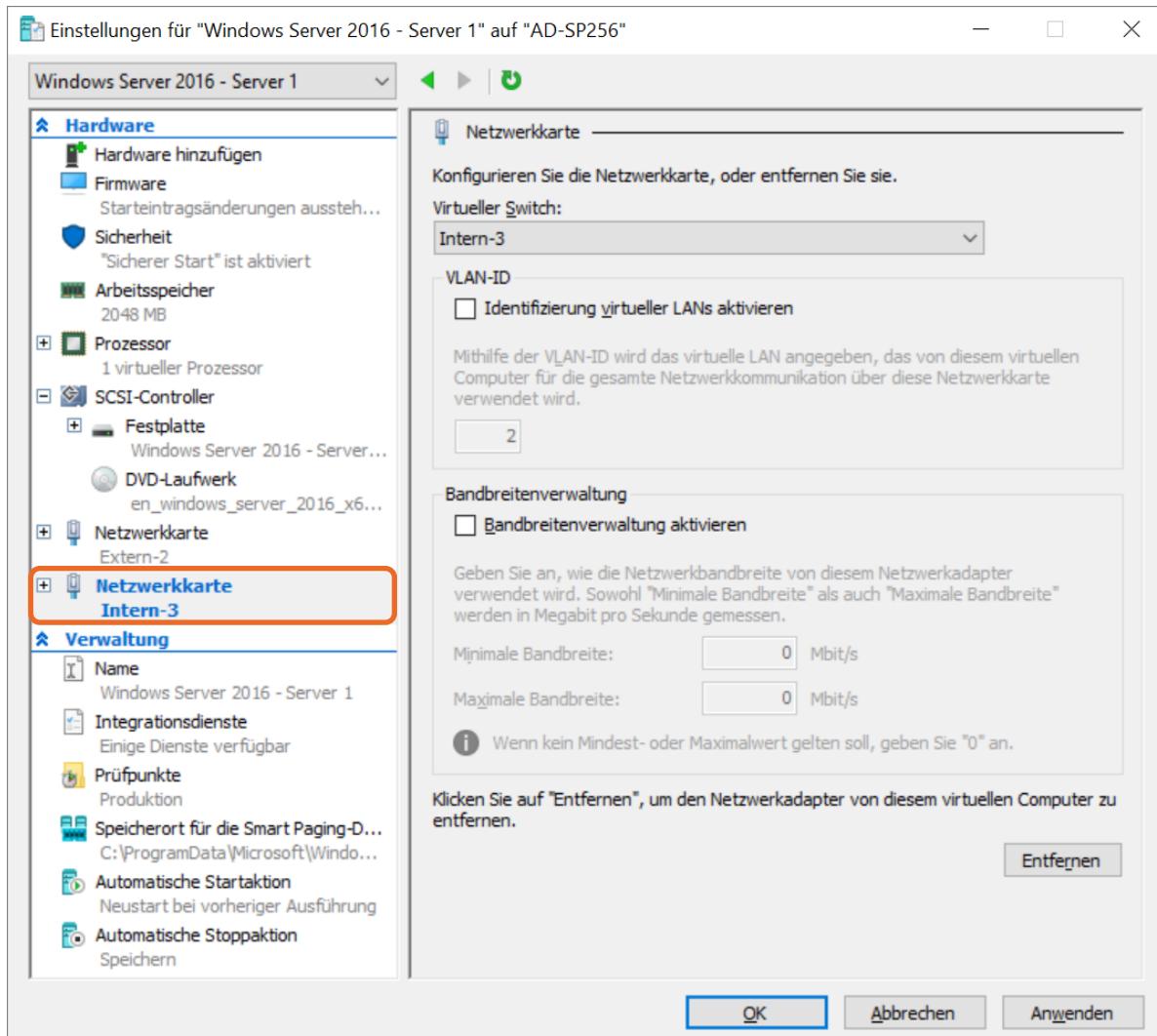
Wenn der virtuelle Computer momentan nicht läuft:

- ▶ Öffnen Sie den Hyper-V-Manager und wählen Sie den virtuellen Computer, für den Sie die Einstellung vornehmen wollen, durch Linksklick aus.
- ▶ Klicken Sie im Kontextmenü des markierten Computers auf *Einstellungen*.



*Eine weitere Netzwerkkarte hinzufügen*

- ▶ Wählen Sie im Bereich *Hardware hinzufügen* den Eintrag *Netzwerkkarte* und klicken Sie auf *Hinzufügen*.
- ▶ Geben Sie einen Namen Ihrer Wahl ein und wählen Sie als Verbindungstyp *Internes Netzwerk*.



Eine weitere Netzwerkkarte im Hyper-V-Manager hinzufügen

Diese Schritte wiederholen Sie für alle virtuellen Computer, die Sie als Mitglied Ihres virtuellen Netzwerks konfigurieren möchten.

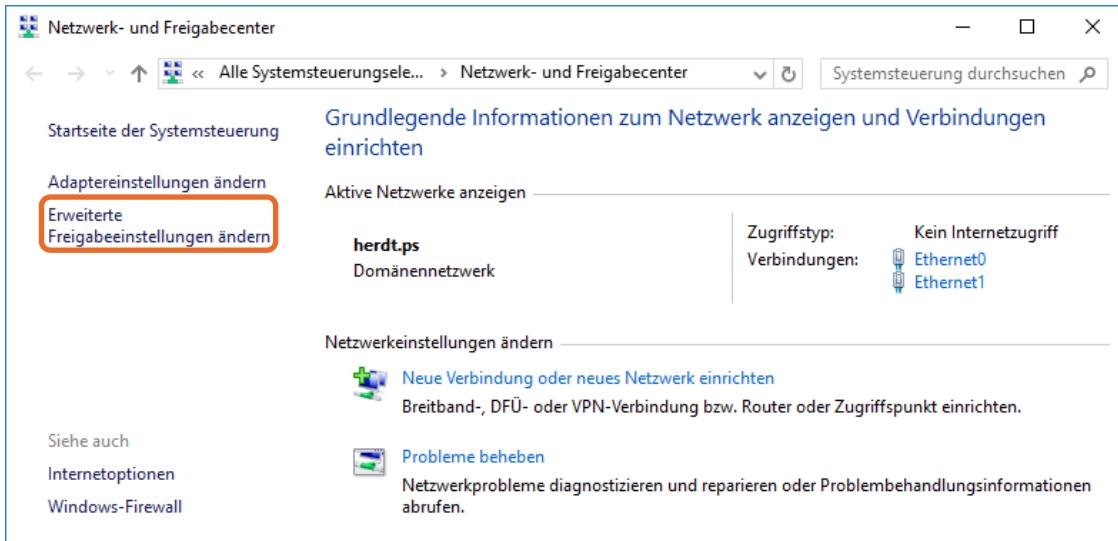
### IP-Adressen einrichten und Kommunikation testen

Im folgenden Abschnitt werden Sie eine virtuelle Maschine mit dem Betriebssystem Windows Server 2016 zum Domänencontroller machen. Aus Gründen der Einfachheit konfigurieren Sie alle verwendeten virtuellen Maschinen mit einer statischen IPv4-Adresse.

- ▶ Wechseln Sie auf den Desktop und klicken Sie rechts unten mit der rechten Maustaste auf die Schaltfläche.
- ▶ Richten Sie den Netzwerkadapter *Ethernet* des Servers, der später als Domänencontroller fungieren soll, mit folgender Adresse ein:
  - ✓ IP-Adresse: 192.168.100.10
  - ✓ Subnetmask: 255.255.255.0
 Alle weiteren virtuellen Maschinen erhalten dieselbe Subnetmask.
- ▶ Richten Sie die IP-Adressen der Maschinen mit den Werten 192.168.100.11 bzw. 192.168.100.12 usw. ein. Eine IP-Adresse muss innerhalb eines Netzwerks eindeutig sein.
- ▶ Falls Sie in Ihrem Netzwerk über einen Router ins Internet verfügen, tragen Sie dessen IP-Adresse im Feld *Standardgateway* ein.

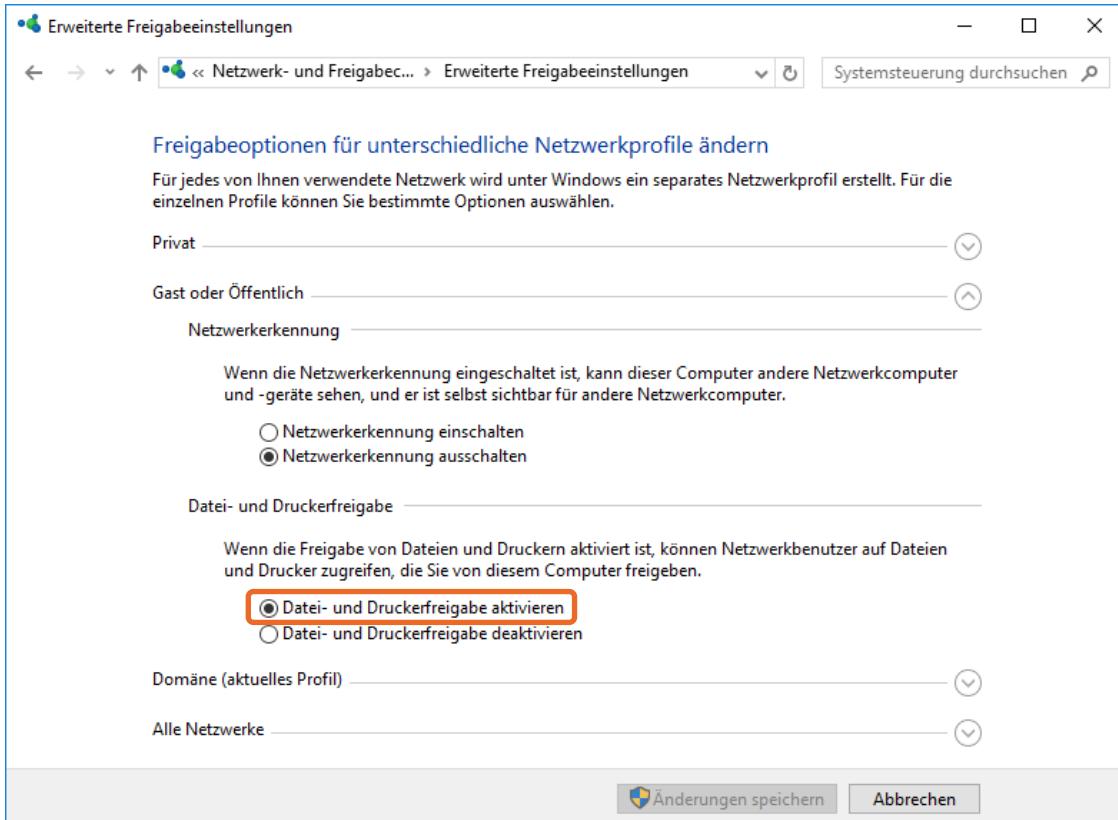
Üblicherweise testen Sie an dieser Stelle mithilfe des Befehls `ping <IP-Adresse>` in der Eingabeaufforderung, ob die Rechner miteinander kommunizieren können. Allerdings erhalten Sie die Rückmeldung: Zeitüberschreitung der Anforderung., auch wenn die Rechner korrekt konfiguriert wurden. Das liegt an den Standardeinstellungen der Betriebssysteme, die eine Antwort auf Anfragen über das Protokoll ICMP (und damit auch die Echo-Anfrage `ping`) untersagen.

- Öffnen Sie in der virtuellen Maschine das *Netzwerk- und Freigabecenter*. Um dies zu tun, geben Sie den Namen der Anwendung im Startbildschirm ein oder klicken Sie auf dem Desktop mit der rechten Maustaste auf das Netzwerksymbol rechts unten in der Taskleiste.



#### *Netzwerk- und Freigabecenter: IP-Adresskonfiguration und erweiterte Freigabeeinstellungen*

- Wählen Sie im linken Menü den Eintrag *Erweiterte Freigabeeinstellungen ändern*.



#### *Erweiterte Freigabeeinstellungen: Datei- und Druckerfreigabe*

- Aktivieren Sie im aktiven Netzwerkprofil die Datei- und Druckerfreigabe.

Damit wird automatisch eine Firewallregel aktiviert, die es dem konfigurierten Rechner im konfigurierten Netzwerkprofil erlaubt, auf Echoanforderungen anderer Hosts zu reagieren. Diese Schritte wiederholen Sie für alle virtuellen Maschinen.

Durch diese Einstellungen haben Sie erreicht, dass sich die virtuellen Maschinen gegenseitig „anpingen“ können bzw. ihre Kontaktaufnahme mit einer Antwort gewürdigt wird.

## A.7 Windows Server 2016 als Domänencontroller einrichten

### Rolle Active Directory-Domänendienste hinzufügen

Im zweiten Teil des Buches sind die Arbeit der PowerShell mit Active Directory sowie die Verwaltung von Domänen zentrale Themen. Aus diesem Grund richten Sie eine Domäne mit einem Windows 2016 Domänencontroller ein. Die Anleitung gilt gleichermaßen für einen Windows Server 2019.

- Starten Sie eine virtuelle Maschine mit dem Betriebssystem Windows Server 2016 (oder 2019).

Sie gelangen automatisch in den Server-Manager, über den Sie die Konfiguration vornehmen können.

- Klicken Sie im Menü rechts oben auf *Verwalten* und wählen Sie den Eintrag *Rollen und Features hinzufügen*.

Es öffnet sich der Assistent zum Hinzufügen von Rollen und Features.

- Überspringen Sie die Startseite und übernehmen Sie auf der zweiten Seite des Assistenten (*Installationstyp*) die Vorauswahl *Rollenbasierte oder featurebasierte Installation*.
- Auf der Seite *Serverauswahl* wählen Sie aus dem Serverpool Ihren eigenen Server aus.  
Es wird Ihnen wahrscheinlich auch kein anderer Server zur Auswahl angeboten.
- Wählen Sie auf der Seite *Serverrollen* die Rolle *Active Directory-Domänendienste* zur Installation aus.  
Es öffnet sich direkt im Anschluss ein Fenster mit der Nachfrage, ob die erforderlichen Features für diese Rolle hinzugefügt werden sollen.
- Bestätigen Sie diese Nachfrage, indem Sie auf die Schaltfläche *Features hinzufügen* klicken.
- Auf der Seite *Features* sehen Sie die bereits vorgenommene Auswahl der zusätzlichen Features. Ändern Sie nichts, klicken Sie auf *Weiter*. Auf der folgenden Seite *AD DS* klicken Sie ebenfalls auf *Weiter*.
- Auf der letzten Seite des Assistenten (*Bestätigung*) klicken Sie auf *Installieren*.

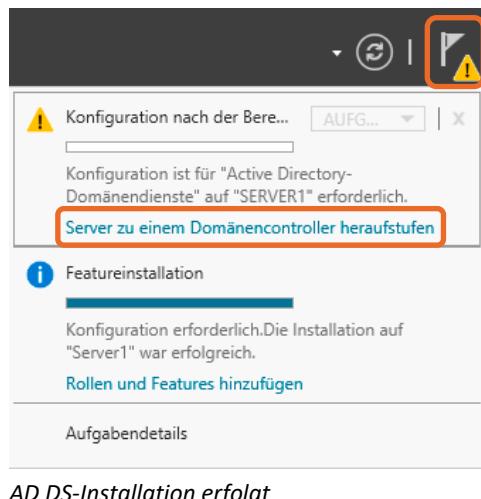


Die Installation der Rolle startet. Bei Windows Server 2016 (oder 2019) können Sie an dieser Stelle den Assistenten schließen, ohne die Ausführung dieser Aufgabe zu unterbrechen.

Nach erfolgter Installation zeigt das Benachrichtigungssymbol im Server-Manager (in der nebenstehenden Grafik rechts oben markiert) ein Ausrufezeichen. Wenn Sie auf das Symbol klicken, erhalten Sie die Auskunft, dass die Installation erfolgreich war und noch eine Konfiguration der Active Directory-Domänendienste auf diesem Rechner erforderlich ist.

- Darunter befindet sich ein Link, auf den Sie klicken, um diesen Rechner zu einem Domänencontroller heraufzustufen.

Der Befehl `dcpromo`, mit dem Sie einen Server mit einem älteren Betriebssystem zu einem Domänencontroller heraufstufen können, steht in Windows Server 2016 (oder 2019) für diese allgemeine Konfigurationsaufgabe nicht mehr zur Verfügung.



### Rechner zum Domänencontroller heraufstufen

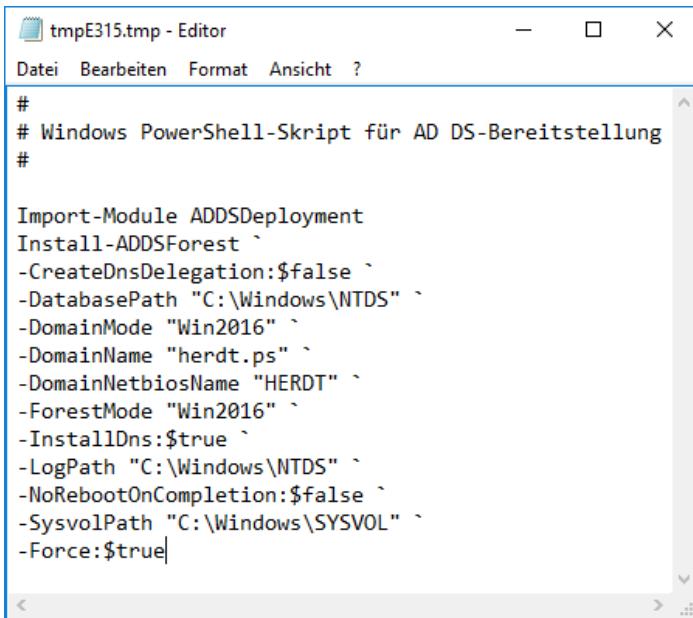
Nachdem Sie auf den Link *Server zu einem Domänencontroller heraufstufen* geklickt haben, öffnet sich ein Assistent zur Konfiguration.

- Wählen Sie auf der Seite *Bereitstellungskonfiguration* den Bereitstellungsvorgang *Neue Gesamtstruktur hinzufügen* aus und geben Sie den Namen der Stammdomäne an, den Sie verwenden wollen. In den Beispielen für das Buch ist der Name der Domäne *herdt.ps*. Klicken Sie dann auf *Weiter*.
- Auf der Seite *Domänencontrolleroptionen* sind mehrere Einstellungen vorzunehmen:
  - ✓ Als Funktionsebene der Gesamtstruktur und der Stammdomäne wählen Sie mindestens *Windows Server 2008 R2*. Für die Beispiele im Buch ist dies der Minimalwert, höhere Werte sind ebenfalls praktikabel.
  - ✓ Bei den Domänencontrollerfunktionen achten Sie darauf, dass das Häkchen bei *DNS-Server* gesetzt ist. Eine Windows-Domäne benötigt zwingend einen DNS-Server. Durch die Auswahl wird automatisch und ohne weitere Benutzereingabe ein DNS-Server für die Domäne installiert.
  - ✓ Geben Sie das Kennwort für den Verzeichnisdienst-Wiederherstellungsmodus ein.
 Klicken Sie anschließend auf *Weiter*.
- Auf der folgenden Seite (*DNS-Optionen*) erscheint eine Warnung, die Sie lesen und ignorieren können. Klicken Sie auf *Weiter*.
- Auf den Seiten *Zusätzliche Optionen* und *Pfade* ist keine Eingabe Ihrerseits erforderlich. Klicken Sie jeweils auf *Weiter*.

- ▶ Auf der Seite *Optionen prüfen* erhalten Sie eine Zusammenfassung Ihrer Konfigurationsangaben. Klicken Sie rechts unten auf die Schaltfläche *Skript anzeigen*.

Es wird eine Datei im Editor angezeigt, die Ihre kompletten Eingaben im Assistenten in ein PowerShell-Skript umgesetzt hat. Da die PowerShell in neuen Systemen die Basis bildet, werden Aktionen in den grafischen Oberflächen im Hintergrund in PowerShell-Befehle umgewandelt.

- ▶ Da Sie die Datei nicht benötigen, schließen Sie das Fenster wieder und setzen die Arbeit im Assistenten auf der Seite *Optionen prüfen* fort. Klicken Sie auf *Weiter*.
- ▶ Sie erhalten schließlich auf der Seite *Voraussetzungsüberprüfung* das Ergebnis einer Prüfung, ob alle Voraussetzungen erfüllt werden. Schließen Sie den Assistenten durch Klick auf die Schaltfläche *Installieren* ab.



```

tmpE315.tmp - Editor
Datei Bearbeiten Format Ansicht ?

#
# Windows PowerShell-Skript für AD DS-Bereitstellung
#
Import-Module ADDSDeployment
Install-ADDSForest `

-CREATEDNSDELEGATION:$false `

-DATABASEPATH "C:\Windows\NTDS" `

-DOMAINMODE "Win2016" `

-DOMAINNAME "herdt.ps" `

-DOMAINNETBIOSNAME "HERDT" `

-FORESTMODE "Win2016" `

-INSTALLDNS:$true `

-LOGPATH "C:\Windows\NTDS" `

-NOREBOOTONCOMPLETION:$false `

-SYSVOLPATH "C:\Windows\SYSVOL" `

-FORCE:$true|
```

Im Hintergrund arbeitet auch hier die PowerShell

Die Installation wird nun gemäß Ihren Angaben vorgenommen. Der Assistent bleibt dabei geöffnet. Er zeigt den Fortschritt der einzelnen Installationsschritte auf der Seite *Installation*. Am Ende der Heraufstufung wird der Server automatisch neu gestartet. Der Server fungiert jetzt als Domänencontroller für Ihre definierte Domäne, hier *herdt.ps*.

## A.8 Andere virtuelle Maschinen der Domäne hinzufügen

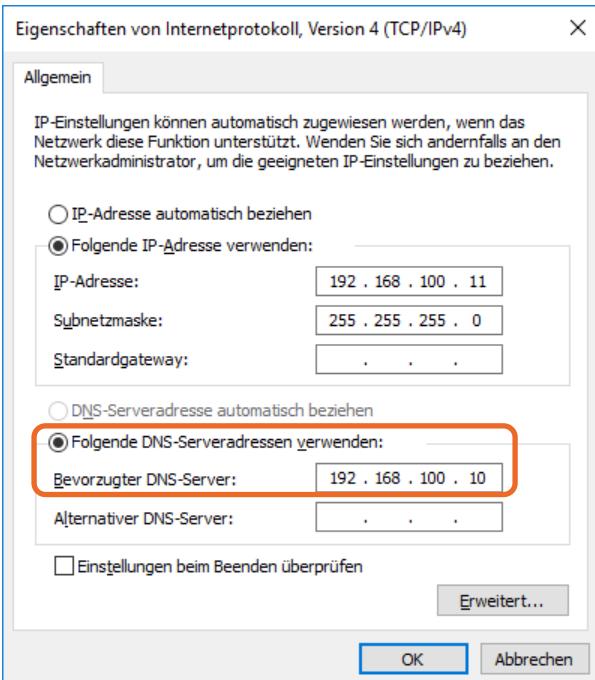
Im letzten Schritt der Vorbereitungen Ihrer virtuellen Umgebung fügen Sie die virtuelle Maschine mit dem Betriebssystem Windows 10 der angelegten Domäne hinzu. Optional wiederholen Sie dies für andere virtuelle Umgebungen, mit denen Sie arbeiten möchten.

### DNS-Server beim Client eintragen

Damit Sie einen Rechner einer Domäne hinzufügen können, müssen Sie die IP-Konfiguration des Rechners anpassen, indem Sie einen DNS-Servereintrag hinzufügen, der für Ihre Domäne konfiguriert ist. Einen solchen DNS-Server haben Sie bereits in Ihrer Domäne: Es ist der Domänencontroller, auf dem Sie zusätzlich einen DNS-Server installiert haben.

Ohne einen Eintrag eines für die Domäne zuständigen DNS-Servers kann ein Rechner keinen Domänencontroller finden, bei dem er sich anmelden könnte.

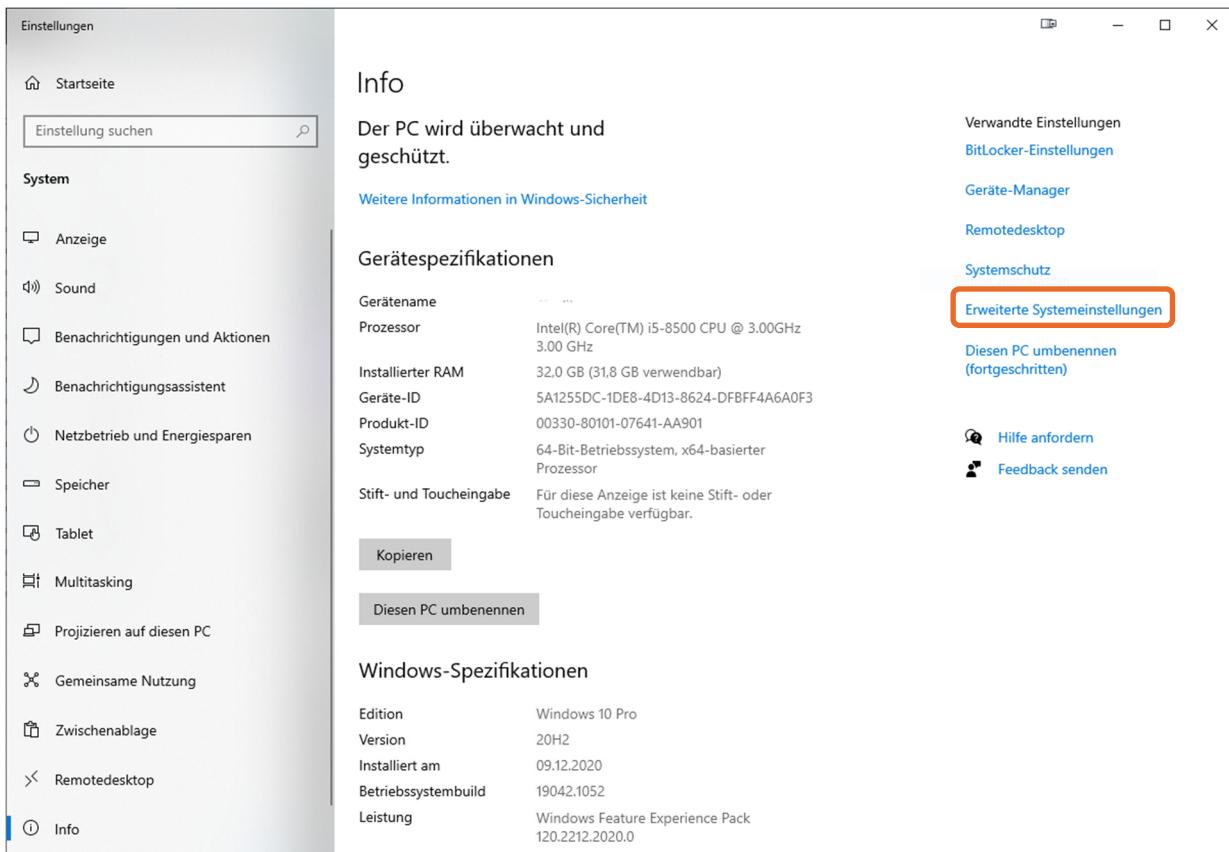
- ▶ Öffnen Sie die Eigenschaften des Netzwerkadapters *Ethernet* und wählen Sie dort die Schaltfläche *Eigenschaften*.
- ▶ Öffnen Sie die Eigenschaften für das *Internetprotokoll Version 4 (TCP/IPv4)*.
- ▶ Tragen Sie zusätzlich zu den Einträgen für IP-Adresse und Subnetzmaske im Feld *Bevorzugter DNS-Server* die IP-Adresse Ihres Domänencontrollers (192.168.100.10) ein.
- ▶ Schließen Sie alle Konfigurationsfenster.



## Zur Domäne hinzufügen

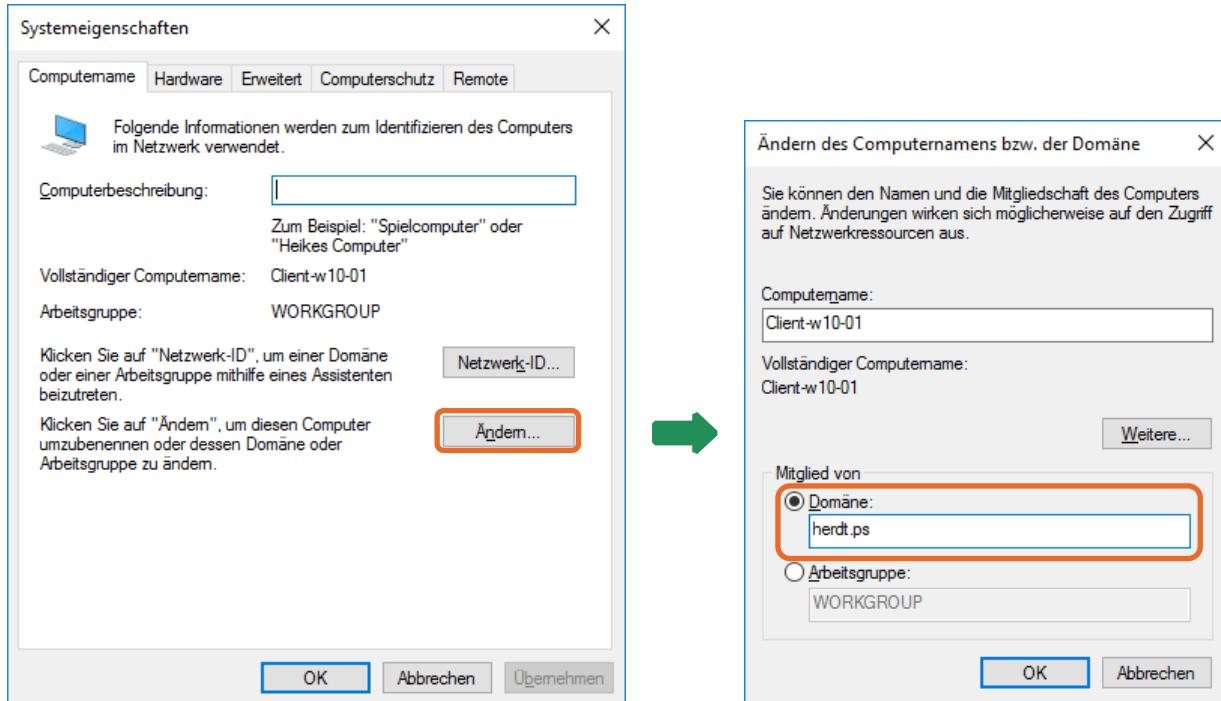
*DNS-Servereintrag für die Domäne hinzufügen*

- ▶ Rufen Sie in Windows 10 die *Einstellungen* auf und wählen Sie den Eintrag *System*.



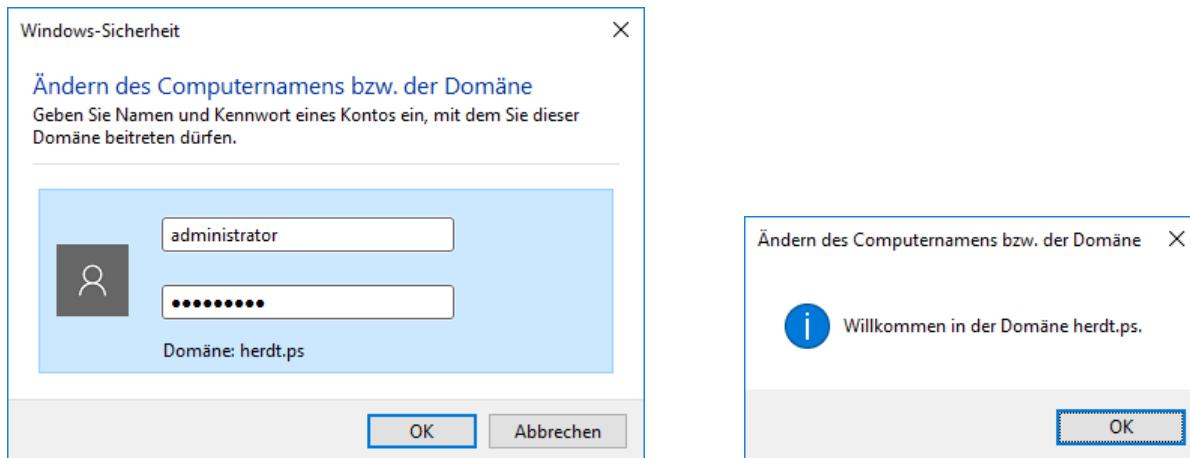
*Windows 10: Einstellungen – System*

- ▶ Klicken Sie unter dem Eintrag *Info* auf den Link *Erweiterte Systemeinstellungen* im rechten Bildbereich. Beachten Sie, dass Sie für diese Aktion erweiterte Rechte benötigen. Es öffnet sich ein Fenster für die Systemeigenschaften Ihres Rechners.



#### *Rechner einer Domäne hinzufügen*

- Ist alles richtig konfiguriert, erscheint ein Fenster, in das Sie den Namen und das Kennwort eines Kontos eintragen müssen, mit dem Sie der Domäne beitreten dürfen. Verwenden Sie hier Name und Kennwort des Domänenadministrators.



#### *Anmeldeinformationen eines Kontos eingeben, mit dem Sie der Domäne beitreten dürfen*

Sie werden in der Domäne begrüßt und erhalten die Meldung, dass der Computer neu gestartet werden muss. Der Rechner ist nun Domänenmitglied, für alle weiteren Arbeiten können Sie sich mit einem Domänenkonto anmelden (hier für Übungszwecke auch mit dem Konto des Domänenadministrators mit seinen umfassenden Rechten).

Schulversion

**\$**

\$?	91
\$_	41, 43, 120
\$Error	91
\$ErrorActionPreference	90
\$HOME	81, 91
\$host	121
\$input	120
\$MaximumAliasCount	90
\$MaximumHistoryCount	30, 91
\$null	93
\$PSSHOME	81
\$psUnsupportedConsole- Applications	20
\$WhatIfPreference	91
.	
.NET-Framework	5

**A**

Active Directory-Konten	165
Active Directory-Konten aktivieren	166
Active Directory-Konten deaktivieren	166
Active Directory-Konten, Ablaufdatum	166
Active Directory-Konten, Kennwörter verwalten	168
Active Directory-Konten, Konto entsperren	169
Active Directory-Konten, Suche nach	167
Active Directory-Modul für Windows PowerShell	137
Active Directory-Papierkorb	175
Active Directory- Verwaltungszentrum	156, 178
Add-ADFineGrained- PasswordPolicySubject	164
Add-ADGroupMember	149
Add-Content	70
Add-WindowsFeature	133
ADSI-Editor	139
Alias	74
Alias exportieren	76
Alias importieren	77
Alias löschen	77
Alias, eigener	75
Alias, vordefinierter	74
Anweisungsblock	102
Array	96
Aufgabenplanung	184
Automatische Vervollständigung	8

**B**

Befehlszeilenkonsole	5
Benutzerimport	206
Benutzerobjekte	145
Benutzerprofil	80
Betriebsmaster	170, 175
break	111, 204

**C**

cd	75
CIM (WMI)	191
Clear-	
ADAccountExpiration	166
Clear-Content	70
Clear-History	30
Clear-Item	65
Clear-ItemProperty	65
Clear-Variable	95
cls	75
Cmdlet	21
Cmdlet, Syntax	21
Common parameters	24
Compare-Object	37
Computer, virtuelle	216
Computerobjekte	147
continue	111
ConvertTo-Csv	51
ConvertTo-Html	52
ConvertTo-SecureString	53
ConvertTo-Xml	54
Copy-ItemProperty	65

**D**

Definierter Name	139
Deleted Objects, Container	178
DHCP	187
dir	61, 75
Disable-ADAccount	166
Disable	
-ADOptionalFeature	177
Do While	108
Dokumentation	123
Domänencontroller	222
dotsourced	189
Dotsourcing	189
Do-Until-Schleife	108
Do-While-Schleife	106

**E**

Enable-ADAccount	166
Enable	
-ADOptionalFeature	177

Enable-ServerManager- StandardUserRemoting	133
Enumerator	27
Export-Alias	76
Export-Clixml	49
Export-Csv	50
Externe Befehle	19

**F**

Fein abgestimmte Kennwortrichtlinien	159
Festplatten, virtuelle	215
Filter	122
Filter, Dokumentation	123
For	109
ForEach	110
ForEach-Object	42, 110, 209
ForEach-Schleife	106, 109
For-Schleife	106, 109
function	114
Funktion	114
Funktion, Begin	118
Funktion, benannter Parameter	115
Funktion, Dokumentation	123
Funktion, End	118
Funktion, Namengebung	115
Funktion, Pipeline	118
Funktion, Process	118
Funktion, Switch-Parameter	115
Funktionsparameter, Standardwert	116

**G**

Gelöschte Objekte	179
Gesamtstruktur	196
Gesamtstruktur installieren	196
Gesamtstrukturfunktionsmodus	175
Get-ADDomain	171
Get-ADFineGrained- PasswordPolicy	159
Get-ADFineGrained- PasswordPolicySubject	164
Get-ADForest	171
Get-ADGroupMember	149
Get-ADObject	151, 179
Get-ADOptionalFeature	177
Get-ADUserResultant- PasswordPolicy	165
Get-ChildItem	22, 27, 57, 61, 75
Get-Command	21, 26, 37
Get-Content	68
Get-Culture	51
Get-Date	27
Get-DisplayResolution	136

<b>G</b>	Get-DnsClientServer-Address	188	Kontrollstrukturen, Do-While-Schleife	106	<b>O</b>
	Get-ExecutionPolicy	85	Kontrollstrukturen, ForEach-Schleife	106, 109	Objektorientiert
	Get-Help	30	Kontrollstrukturen, For-Schleife	106	Objektverwaltung, allgemeine
	Get-History	28, 30	Kontrollstrukturen, If-Anweisung	102	Operator, arithmetischer
	Get-Item	65	Kontrollstrukturen, If-Else-Anweisung	103	Organisationseinheit (OU)
	Get-Location	62, 88	Kontrollstrukturen, Schleifen	106	Out-File
	Get-Member	28, 36	Kontrollstrukturen, Switch-Anweisung	105	Out-GridView
	Get-Module	28	Kontrollstrukturen, While-Schleife	106	Out-Host
	Get-NetAdapter	186, 197	Kontrollstrukturen, If-Anweisung mit ElseIf	104	Out-Printer
	Get-NetIPAddress	28, 187	<b>L</b>	<b>P</b>	
	Get-NetIPConfiguration	186	ls	61	Parameter
	Get-Process	28	<b>M</b>	Parameter, allgemeine	
	Get-PSDrive	28, 58	Measure-Object	44	Parameter, benannte
	Get-PSPowerProvider	28, 57	Modul	21, 128, 130, 197	Password Settings Container
	Get-PSReadlineOption	21	Modul ActiveDirectory	137, 143	Pipeline
	Get-ScheduledTask	184	Modul ADDSDeployment	197	Pipeline schrittweise entwickeln
	Get-Service	28, 75	Modul DnsServer	135	Pipeline, Einsatztipps
	Get-Variable	92	Modul NetAdapter	186	Pipelineoperator
	Get-WindowsFeature	133	Modul NetTCP/IP	187	Pop-Location
	Group-Object	43, 75	Modul ScheduledTasks	184	Positionsparameter
	Gruppenmitgliedschaft verwalten	149	Modul ServerCore	136	PowerShell ISE
	Gruppenobjekte	146	Modul ServerManager	133	PowerShell ISE, IntelliSense
<b>H</b>	Modul, verfügbares	129	Modul, verfügbares	129	PowerShell unter Windows
	<b>I</b>		Move-ADDSDirectoryServer-OperationMasterRole	171	PowerShell unter Windows Server
	if	104	Move-ADObject	151	PowerShell, 32-bit-Version
	If	102	<b>N</b>	187	PowerShell, 64-bit-Version
	If, Else	103, 104	Netzwerkadapter, Konfiguration	186	PowerShell-Fehlermeldungen
	Import-Module	132	New-		PowerShell-Konsole konfigurieren
	Install-ADDSDomain	197	ADFineGrained-PasswordPolicy	159	PowerShell-Konsole, Vervollständigung
	Install-ADDSDomain-Controller	197, 202	New-ADObject	151	PowerShell-Laufwerk
	Install-ADDSDForest	199	New-ADOrganizationalUnit	205	PowerShell-Laufwerke, virtuelle
	Install-WindowsFeature	133	New-ADUser	208	PowerShell-Provider
	IntelliSense	8, 139	New-Alias	75	Präinkrement
	Invoke-History	30	New-Item	82	Profil
	IP-Konfiguration	185	New-LocalUser	141	Profildatei
<b>K</b>	New-NetIPAddress	187, 197	Prompt (Eingabeaufforderung)	88	Powershell
	New-Object	37	ProtectedFrom-AccidentalDeletion	153	Präfix
	New-PSDrive	59	PSModulePath	128	Präfixe
	New-Variable	93	PSO (Password Settings Objects)	159	Präfixnamen
	<b>Q</b>		Provider Registry	71	Präfixnamenkonflikt
	New-NetIPAddress	187, 197	Push-Location	63	Präfixnamenkonflikt, Lösung
	New-Object	37	<b>R</b>		Präfixnamenkonflikt, Lösung, Beispiel
	New-PSDrive	59	Read-Host	169	Präfixnamenkonflikt, Lösung, Beispiel, Lösung
	New-Variable	93	Rechenregeln, mathematische	98	Präfixnamenkonflikt, Lösung, Beispiel, Lösung, Lösung
	<b>S</b>		Rechnen	19	Präfixnamenkonflikt, Lösung, Beispiel, Lösung, Lösung, Lösung

Registry	71	Set-Alias	75	<b>U</b>
Remoteserver-Verwaltungstools	130	Set-Content	69	
Remove-ADComputer	151	Set-DisplayResolution	137	
Remove-ADFineGrained-PasswordPolicy	159	Set-DnsClientServer-Address	188, 201	
Remove-ADFineGrained-PasswordPolicySubject	164	Set-ExecutionPolicy	86	
Remove-ADGroup	151	Set-ItemProperty	189	<b>V</b>
Remove-ADGroupMember	149, 150	Set-Location	63	
Remove		Set-NetIPAddress	187	
-ADOrganizationalUnit	151	Set-NetIPInterface	187	
Remove-ADUser	151	Set-Variable	94	
Remove-Item	66	Show-Command	29	
Remove-ItemProperty	66	Skript starten	113	
Remove-Module	132	Skript, Ausführungsrichtlinie	84	
Remove-NetIPAddress	187	Skript, Dokumentation	123	
Remove-PSDrive	59, 61	Skript, Pipeline	121	
Remove-Variable	95	Skripte, Ausführungsrichtlinie	113	
Remove-WindowsFeature	133	Skriptsprache	5	
Rename-ADObject	152	SnapIn	21	
Rename-Computer	197	Sort-Object	39, 75	
Rename-Item	66	Start-Job	199	
Restart-Computer	197	Start-Process	75	
Restore-ADObject	152, 180	Stop-Process	56	
Risikominderungsparameter	25	Switch	105	
RSAT (Remote Server Administration Tools)	130	Switch-Parameter	22	
		Sync-ADObject	152	
		Syntax, klassische	40	
		Syntax, vereinfachte	40	<b>W</b>
<hr/>				
<b>S</b>		<b>T</b>		
Save-Help	32	Tabulator-Vervollständigung	8	
Schleife, Do While	108	Tee-Object	45	
Schleife, For	109	Test-ADDSDomain-Controller-Installation	197	
Schleife, While	107	Test-ADDSDomain-Controller-Uninstallation	197	
Schleifen	106	Test-ADDSDomain-Installation	197	
Search-ADAccount	166, 167	Test-ADDSForest-Installation	197	
Select-Object	38, 202	Test-Path	82	<b>Z</b>
Server Core	188	Testumgebung	218	
Server Core, Standard-Shell	188	try{} ... catch{}	204	
Set-ADAccountExpiration	166			
Set-ADAccount-Password	166, 168			
Set-ADFineGrained-PasswordPolicy	159			
Set-ADForestMode	176			
Set-ADObject	152			

# Impressum

Matchcode: WPOW51

Autor: Andreas Dittfurth

Produziert im HERDT-Digitaldruck

1. Ausgabe, Juni 2021

HERDT-Verlag für Bildungsmedien GmbH  
Am Kümmerling 21–25  
55294 Bodenheim  
Internet: [www.herdt.com](http://www.herdt.com)  
E-Mail: [info@herdt.com](mailto:info@herdt.com)

© HERDT-Verlag für Bildungsmedien GmbH, Bodenheim

Alle Rechte vorbehalten. Kein Teil des Werkes darf in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder einem anderen Verfahren) ohne schriftliche Genehmigung des Verlags reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Dieses Buch wurde mit großer Sorgfalt erstellt und geprüft. Trotzdem können Fehler nicht vollkommen ausgeschlossen werden. Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

Wenn nicht explizit an anderer Stelle des Werkes aufgeführt, liegen die Copyrights an allen Screenshots beim HERDT-Verlag. Sollte es trotz intensiver Recherche nicht gelungen sein, alle weiteren Rechteinhaber der verwendeten Quellen und Abbildungen zu finden, bitten wir um kurze Nachricht an die Redaktion.

Die in diesem Buch und in den abgebildeten bzw. zum Download angebotenen Dateien genannten Personen und Organisationen, Adress- und Telekommunikationsangaben, Bankverbindungen etc. sind frei erfunden. Eventuelle Übereinstimmungen oder Ähnlichkeiten sind unbeabsichtigt und rein zufällig.

Die Bildungsmedien des HERDT-Verlags enthalten Verweise auf Webseiten Dritter. Diese Webseiten unterliegen der Haftung der jeweiligen Betreiber, wir haben keinerlei Einfluss auf die Gestaltung und die Inhalte dieser Webseiten. Bei der Bucherstellung haben wir die fremden Inhalte daraufhin überprüft, ob etwaige Rechtsverstöße bestehen. Zu diesem Zeitpunkt waren keine Rechtsverstöße ersichtlich. Wir werden bei Kenntnis von Rechtsverstößen jedoch umgehend die entsprechenden Internetadressen aus dem Buch entfernen.

Die in den Bildungsmedien des HERDT-Verlags vorhandenen Internetadressen, Screenshots, Bezeichnungen bzw. Beschreibungen und Funktionen waren zum Zeitpunkt der Erstellung der jeweiligen Produkte aktuell und gültig. Sollten Sie die Webseiten nicht mehr unter den angegebenen Adressen finden, sind diese eventuell inzwischen komplett aus dem Internet genommen worden oder unter einer neuen Adresse zu finden. Sollten im vorliegenden Produkt vorhandene Screenshots, Bezeichnungen bzw. Beschreibungen und Funktionen nicht mehr der beschriebenen Software entsprechen, hat der Hersteller der jeweiligen Software nach Drucklegung Änderungen vorgenommen oder vorhandene Funktionen geändert oder entfernt.