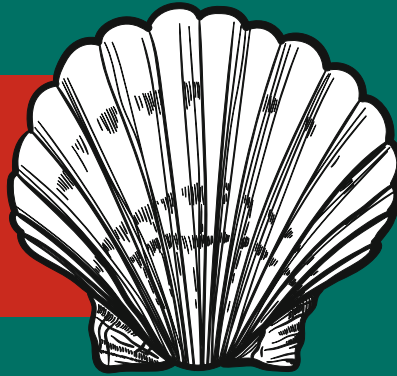


Holger SCHWICHTENBERG

Für
Windows,
Linux und
macOS

PowerShell 7 und Windows PowerShell 5



DAS PRAXISBUCH

5. Auflage



Im Internet: Codebeispiele
und PowerShell-Kurzreferenz

HANSER

www.IT-Visions.de
Dr. Holger Schwichtenberg

Schwichtenberg

PowerShell 7 und Windows PowerShell 5 – das Praxisbuch



Bleiben Sie auf dem Laufenden!

Unser **Computerbuch-Newsletter** informiert Sie monatlich über neue Bücher und Termine. Profitieren Sie auch von Gewinnspielen und exklusiven Leseproben. Gleich anmelden unter:

www.hanser-fachbuch.de/newsletter



Holger Schwichtenberg

PowerShell 7 und Windows PowerShell 5

Das Praxisbuch

5., aktualisierte Auflage

HANSER

Alle in diesem Buch enthaltenen Informationen, Verfahren und Darstellungen wurden nach bestem Wissen zusammengestellt und mit Sorgfalt getestet. Dennoch sind Fehler nicht ganz auszuschließen. Aus diesem Grund sind die im vorliegenden Buch enthaltenen Informationen mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autoren und Verlag übernehmen infolgedessen keine juristische Verantwortung und werden keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieser Informationen – oder Teilen davon – entsteht.

Ebenso übernehmen Autoren und Verlag keine Gewähr dafür, dass beschriebene Verfahren usw. frei von Schutzrechten Dritter sind. Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Buch berechtigt deshalb auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.



Bibliografische Information der Deutschen Nationalbibliothek:

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Dieses Werk ist urheberrechtlich geschützt.

Alle Rechte, auch die der Übersetzung, des Nachdruckes und der Vervielfältigung des Buches, oder Teilen daraus, vorbehalten. Kein Teil des Werkes darf ohne schriftliche Genehmigung des Verlages in irgendeiner Form (Fotokopie, Mikrofilm oder ein anderes Verfahren) – auch nicht für Zwecke der Unterrichtsgestaltung – reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

© 2022 Carl Hanser Verlag München, www.hanser-fachbuch.de

Lektorat: Sylvia Hasselbach

Copy editing: Matthias Bloch, Bochum, und Sandra Gottmann, Wasserburg

Umschlagdesign: Marc Müller-Bremer, www.rebranding.de, München

Umschlagrealisation: Max Kostopoulos

Titelmotiv: © shutterstock.com/Irina Kolesnichenko

Satz: Eberl & Koesel Studio, Kempten (revised version)

Druck und Bindung: Hubert & Co. GmbH & Co. KG BuchPartner, Göttingen

Printed in Germany

Print-ISBN: 978-3-446-47296-9

E-Book-ISBN: 978-3-446-47446-8

E-Pub-ISBN: 978-3-446-47574-8

Inhalt

Vorwort	XXIV
Über den Autor	XXXII
Teil A: PowerShell-Basiswissen	1
1 Fakten zur PowerShell	3
1.1 Was ist die PowerShell?	3
1.2 Geschichte der PowerShell	4
1.3 Welche Varianten und Versionen der PowerShell gibt es?	6
1.4 Windows PowerShell versus PowerShell Core versus PowerShell 7.x	6
1.5 Motivation zur PowerShell	8
1.6 Betriebssysteme mit vorinstallierter PowerShell	11
1.7 Support der PowerShell	13
1.8 Einflussfaktoren auf die Entwicklung der PowerShell	15
1.9 Anbindung an Klassenbibliotheken	16
1.10 PowerShell versus WSH	17
2 Erste Schritte mit der PowerShell	20
2.1 Windows PowerShell herunterladen und auf anderen Windows-Betriebssystemen installieren	20
2.2 Die Windows PowerShell testen	24
2.3 Woher kommen die PowerShell-Befehle?	33
2.4 PowerShell Community Extensions (PSCX) herunterladen und installieren ...	34
2.5 Den Windows PowerShell-Editor „ISE“ verwenden	41
2.6 PowerShell 7 installieren und testen	45
3 Einzelbefehle der PowerShell	57
3.1 Commandlets	57
3.2 Aliase	70
3.3 Ausdrücke	78
3.4 Externe Befehle (klassische Kommandozeilenbefehle)	79
3.5 Dateinamen	81

4	Hilfefunktionen	82
4.1	Auflisten der verfügbaren Befehle	82
4.2	Praxistipp: Den Standort eines Kommandozeilenbefehls suchen	83
4.3	Anzahl der Befehle	84
4.4	Volltextsuche	86
4.5	Erläuterungen zu den Befehlen	86
4.6	Hilfe zu Parametern	87
4.7	Hilfe mit Show-Command	89
4.8	Hilfefenster	90
4.9	Allgemeine Hilfetexte	92
4.10	Aktualisieren der Hilfedateien	92
4.11	Online-Hilfe	94
4.12	Fehlende Hilfetexte	95
4.13	Dokumentation der .NET-Klassen	96
5	Objektorientiertes Pipelining	98
5.1	Befehlsübersicht	98
5.2	Pipeline-Operator	99
5.3	.NET-Objekte in der Pipeline	100
5.4	Pipeline Processor	101
5.5	Pipelining von Parametern	103
5.6	Pipelining von klassischen Befehlen	105
5.7	Zeilenumbrüche in Pipelines	107
5.8	Schleifen	108
5.9	Zugriff auf einzelne Objekte aus einer Menge	111
5.10	Zugriff auf einzelne Werte in einem Objekt	112
5.11	Methoden ausführen	114
5.12	Analyse des Pipeline-Inhalts	116
5.13	Filtern	131
5.14	Zusammenfassung von Pipeline-Inhalten	136
5.15	„Kastrierung“ von Objekten in der Pipeline	136
5.16	Sortieren	137
5.17	Duplikate entfernen	138
5.18	Gruppierung	139
5.19	Objekte verbinden mit Join-String	145
5.20	Berechnungen	146
5.21	Zwischenschritte in der Pipeline mit Variablen	146
5.22	Verzweigungen in der Pipeline	147
5.23	Vergleiche zwischen Objekten	149
5.24	Weitere Praxislösungen	150

6	PowerShell-Skripte	152
6.1	Skriptdateien	152
6.2	Start eines Skripts	154
6.3	Aliase für Skripte verwenden	155
6.4	Parameter für Skripte	156
6.5	Skripte dauerhaft einbinden (Dot Sourcing)	157
6.6	Das aktuelle Skriptverzeichnis	158
6.7	Sicherheitsfunktionen für PowerShell-Skripte	158
6.8	Skripte mit vollen Rechten (Elevation)	160
6.9	Blockierte PowerShell-Skripte	161
6.10	PowerShell-Skripte im Kontextmenü des Windows Explorers	162
6.11	Anforderungsdefinitionen von Skripten	164
6.12	Skripte anhalten	165
6.13	Versionierung und Versionsverwaltung von Skripten	165
7	PowerShell-Skriptsprache	168
7.1	Hilfe zur PowerShell-Skriptsprache	168
7.2	Befehlstrennung	168
7.3	Kommentare	169
7.4	Variablen	170
7.5	Variablenbedingungen	182
7.6	Zahlen	183
7.7	Zeichenketten (Strings)	187
7.8	Reguläre Ausdrücke	197
7.9	Datum und Uhrzeit	203
7.10	Objekte	204
7.11	Arrays	205
7.12	ArrayList	208
7.13	Assoziative Arrays (Hash-Tabellen)	209
7.14	Operatoren	210
7.15	Überblick über die Kontrollkonstrukte	214
7.16	Bedingungen	219
7.17	Unterroutinen (Prozedur/Funktionen)	222
7.18	Eingebaute Funktionen	228
7.19	Fehlerausgabe	229
7.20	Fehlerbehandlung	231
7.21	Laufzeitfehler erzeugen	243
7.22	Objektorientiertes Programmieren mit Klassen	243

8	Ausgaben	246
8.1	Ausgabe-Commandlets	246
8.2	Benutzerdefinierte Tabellenformatierung	249
8.3	Benutzerdefinierte Listenausgabe	251
8.4	Mehrspaltige Ausgabe	251
8.5	Out-GridView	252
8.6	Standardausgabe	254
8.7	Einschränkung der Ausgabe	256
8.8	Seitenweise Ausgabe	256
8.9	Ausgabe einzelner Werte	258
8.10	Details zum Ausgabeoperator	260
8.11	Ausgabe von Methodenergebnissen und Unterobjekten in Pipelines	263
8.12	Ausgabe von Methodenergebnissen und Unterobjekten in Zeichenketten	264
8.13	Unterdrückung der Ausgabe	264
8.14	Ausgaben an Drucker	265
8.15	Ausgaben in Dateien	266
8.16	Umleitungen (Redirection)	266
8.17	Fortschrittsanzeige	267
8.18	Sprachausgabe	267
9	Das PowerShell-Navigationsmodell (PowerShell Provider)	269
9.1	Einführungsbeispiel: Navigation in der Registrierungsdatenbank	269
9.2	Provider und Laufwerke	270
9.3	Navigationsbefehle	272
9.4	Pfadangaben	273
9.5	Beispiel	275
9.6	Eigene Laufwerke definieren	276
10	Fernausführung (Remoting)	277
10.1	RPC-Fernabfrage ohne WS-Management	278
10.2	Anforderungen an PowerShell Remoting	279
10.3	Rechte für PowerShell-Remoting	280
10.4	Einrichten von PowerShell Remoting	280
10.5	Überblick über die Fernausführungs-Commandlets	283
10.6	Interaktive Fernverbindungen im Telnet-Stil	283
10.7	Fernausführung von Befehlen	285
10.8	Parameterübergabe an die Fernausführung	289
10.9	Fernausführung von Skripten	290
10.10	Ausführung auf mehreren Computern	291
10.11	Sitzungen	292
10.12	Implizites Remoting	297

10.13	Zugriff auf entfernte Computer außerhalb der eigenen Domäne	298
10.14	Verwaltung des WS-Management-Dienstes	301
10.15	PowerShell Direct für Hyper-V	302
10.16	Praxislösung zu PowerShell Direct	304
11	PowerShell-Werkzeuge	307
11.1	PowerShell-Standardkonsole	307
11.2	Windows Terminal	322
11.3	Erweiterung der Konsolen	327
11.4	PowerShell Integrated Scripting Environment (ISE)	329
11.5	PowerShell Script Analyzer	340
11.6	PowerShell Analyzer	345
11.7	PowerShell Tools for Visual Studio	346
11.8	PowerShell Pro Tools for Visual Studio	348
11.9	Visual Studio Developer PowerShell	348
11.10	NuGet Package Manager Console (PMC)	351
11.11	Visual Studio Code mit PowerShell-Erweiterung	352
11.12	PowerShell-Erweiterungen für andere Editoren	354
11.13	PowerShell Web Access (PSWA)	355
11.14	Azure Cloud Shell	360
11.15	ISE Steroids	360
11.16	PowerShellPlus	361
11.17	PoshConsole	364
11.18	PowerGUI	365
11.19	PrimalScript	365
11.20	CIM Explorer for PowerShell ISE	367
12	Windows PowerShell Core 5.1 in Windows Nano Server	369
12.1	Installation	369
12.2	PowerShell-Skriptsprache	369
12.3	Werkzeuge	369
12.4	Fehlende Funktionen	370
13	PowerShell 7 für Windows, Linux und macOS	371
13.1	Motivation für den Einsatz der PowerShell 7 auf Linux und macOS	371
13.2	Basis der PowerShell 7	372
13.3	Identifizierung der PowerShell 7	373
13.4	Funktionsumfang der PowerShell 7	373
13.5	Entfallene Befehle in PowerShell 7	376
13.6	Erweiterungsmodule nutzen in PowerShell 7	382
13.7	Geänderte Funktionen in PowerShell 7	387

13.8	Neue Funktionen der PowerShell 7	389
13.9	PowerShell 7-Konsole	392
13.10	Praxislösung: Fallunterscheidung für PowerShell-Varianten	393
13.11	VSCoDe-PowerShell als Editor für PowerShell 7	394
13.12	Verwendung von PowerShell 7 auf Linux und macOS	398
13.13	PowerShell-Remoting via SSH	404
13.14	Performance-Vorteile der PowerShell 7	407
13.15	Dokumentation zur PowerShell 7	408
13.16	Quellcode zur PowerShell 7	410

Teil B: PowerShell-Aufbauwissen **413**

14 Verwendung von .NET-Klassen **415**

14.1	.NET versus .NET Core	415
14.2	Ermitteln der verwendeten .NET-Version	416
14.3	.NET-Bibliotheken	417
14.4	Microsoft Docs	419
14.5	Überblick über die Verwendung von .NET-Klassen	420
14.6	Erzeugen von Instanzen	420
14.7	Parameterbehaftete Konstruktoren	422
14.8	Initialisierung von Objekten	423
14.9	Nutzung von Attributen und Methoden	424
14.10	Statische Mitglieder in .NET-Klassen und statische .NET-Klassen	426
14.11	Generische Klassen nutzen	429
14.12	Zugriff auf bestehende Objekte	431
14.13	Laden von Assemblies	431
14.14	Liste der geladenen Assemblies	433
14.15	Verwenden von NuGet-Assemblies	434
14.16	Objektanalyse	436
14.17	Aufzählungstypen (Auflistungen/Enumerationen)	437

15 Verwendung von COM-Klassen **441**

15.1	Unterschiede zwischen COM und .NET	441
15.2	Erzeugen von COM-Instanzen	442
15.3	Abruf der Metadaten	442
15.4	Nutzung von Attributen und Methoden	443
15.5	Liste aller COM-Klassen	444
15.6	Holen bestehender COM-Instanzen	445
15.7	Distributed COM (DCOM)	445

16	Zugriff auf die Windows Management Instrumentation (WMI)	447
16.1	Einführung in WMI	447
16.2	WMI in der PowerShell	474
16.3	Open Management Infrastructure (OMI)	476
16.4	Abruf von WMI-Objektmenen	476
16.5	Fernzugriffe	477
16.6	Filtern und Abfragen	478
16.7	Liste aller WMI-Klassen	481
16.8	Hintergrundwissen: WMI-Klassenprojektion mit dem PowerShell-WMI-Objektadapter	482
16.9	Beschränkung der Ausgabeliste bei WMI-Objekten	486
16.10	Zugriff auf einzelne Mitglieder von WMI-Klassen	488
16.11	Werte setzen in WMI-Objekten	488
16.12	Umgang mit WMI-Datumsangaben	490
16.13	Methodenaufrufe	491
16.14	Neue WMI-Instanzen erzeugen	492
16.15	Instanzen entfernen	493
16.16	Commandlet Definition XML-Datei (CDXML)	493
17	Dynamische Objekte	497
17.1	Erweitern bestehender Objekte	497
17.2	Komplett dynamische Objekte	499
18	Einbinden von C# und Visual Basic .NET	501
19	Win32-API-Aufrufe	503
20	Benutzereingaben	506
20.1	Read-Host	506
20.2	Benutzerauswahl	507
20.3	Grafischer Eingabedialog	508
20.4	Dialogfenster	509
20.5	Authentifizierungsdiallog	509
20.6	Zwischenablage (Clipboard)	511
21	Fehlersuche	512
21.1	Detailinformationen	512
21.2	Einzelschrittmodus	513
21.3	Zeitmessung	514
21.4	Ablaufverfolgung (Tracing)	515
21.5	Erweiterte Protokollierung aktivieren	517

21.6	Script-Debugging in der ISE	518
21.7	Kommandozeilenbasiertes Script-Debugging	518
22	Transaktionen	520
22.1	Commandlets für Transaktionen	520
22.2	Start und Ende einer Transaktion	521
22.3	Zurücksetzen der Transaktion	522
22.4	Mehrere Transaktionen	523
23	Standardeinstellungen ändern mit Profilskripten	524
23.1	Profilpfade	524
23.2	Ausführungsreihenfolge	526
23.3	Beispiel für eine Profildatei	526
23.4	Starten der PowerShell ohne Profilskripte	528
24	Digitale Signaturen für PowerShell-Skripte	529
24.1	Zertifikat erstellen	529
24.2	Skripte signieren	531
24.3	Verwenden signierter Skripte	533
24.4	Mögliche Fehlerquellen	533
25	Hintergrundaufträge („Jobs“)	534
25.1	Voraussetzungen	534
25.2	Architektur	534
25.3	Starten eines Hintergrundauftrags	535
25.4	Hintergrundaufträge abfragen	536
25.5	Warten auf einen Hintergrundauftrag	537
25.6	Abbrechen und Löschen von Aufträgen	537
25.7	Analyse von Fehlermeldungen	537
25.8	Fernausführung von Hintergrundaufträgen	538
25.9	Praxislösung: Einen Job auf mehreren Computern starten	538
26	Geplante Aufgaben und zeitgesteuerte Jobs	540
26.1	Geplante Aufgaben (Scheduled Tasks)	540
26.2	Zeitgesteuerte Jobs	544
27	PowerShell-Workflows	550
27.1	Ein erstes Beispiel	550
27.2	Unterschiede zu einer Function bzw. einem Skript	554
27.3	Einschränkungen bei Workflows	555
27.4	Workflows in der Praxis	556
27.5	Workflows in Visual Studio erstellen	564

28	Ereignissystem	582
28.1	WMI-Ereignisse	582
28.2	WMI-Ereignisabfragen	582
28.3	WMI-Ereignisse seit PowerShell 1.0	584
28.4	Registrieren von WMI Ereignisquellen seit PowerShell 2.0	585
28.5	Auslesen der Ereignisliste	586
28.6	Reagieren auf Ereignisse	588
28.7	WMI-Ereignisse seit PowerShell-Version 3.0	590
28.8	Registrieren von .NET-Ereignissen	590
28.9	Erzeugen von Ereignissen	591
29	Datenbereiche und Datendateien	593
29.1	Datenbereiche	593
29.2	Datendateien	595
29.3	Mehrsprachigkeit/Lokalisierung	596
30	Desired State Configuration (DSC)	599
30.1	Grundprinzipien	600
30.2	DSC für PowerShell 7	600
30.3	Ressourcen	601
30.4	Verfügbare DSC-Ressourcen	602
30.5	Eigenschaften einer Ressource	605
30.6	Aufbau eines DSC-Dokuments	605
30.7	Commandlets für die Arbeit mit DSC	606
30.8	Ein erstes DSC-Beispiel	606
30.9	Kompilieren und Anwendung eines DSC-Dokuments	607
30.10	Variablen in DSC-Dateien	609
30.11	Parameter für DSC-Dateien	610
30.12	Konfigurationsdaten	611
30.13	Entfernen einer DSC-Konfiguration	614
30.14	DSC Pull Server	617
30.15	DSC-Praxislösung 1: IIS installieren	624
30.16	DSC-Praxislösung 2: Software installieren	626
30.17	DSC-Praxislösung 3: Software deinstallieren	628
30.18	Realisierung einer DSC-Ressource	629
30.19	Weitere Möglichkeiten	629
31	PowerShell-Snap-Ins	630
31.1	Einbinden von Snap-Ins	630
31.2	Liste der Commandlets	634

32 PowerShell-Module	635
32.1 Überblick über die Commandlets	635
32.2 Modularchitektur	636
32.3 Aufbau eines Moduls	637
32.4 Module aus dem Netz herunterladen und installieren mit PowerShellGet	638
32.5 Module manuell installieren	644
32.6 Doppeldeutige Namen	644
32.7 Auflisten der verfügbaren Module	646
32.8 Importieren von Modulen	647
32.9 Entfernen von Modulen	650
33 Ausgewählte PowerShell-Erweiterungen	651
33.1 PowerShell-Module in Windows 8.0 und Windows Server 2012	652
33.2 PowerShell-Module in Windows 8.1 und Windows Server 2012 R2	654
33.3 PowerShell-Module in Windows 10 und Windows Server 2019	656
33.4 PowerShell Community Extensions (PSCX)	660
33.5 PowerShellPack	664
33.6 www.IT-Visions.de: PowerShell Extensions	666
33.7 Quest Management Shell for Active Directory	666
33.8 Microsoft Exchange Server	668
33.9 System Center Virtual Machine Manager	669
33.10 PowerShell Management Library for Hyper-V (pshyperv)	669
33.11 PowerShell Configurator (PSConfig)	670
34 Delegierte Administration/Just Enough Administration (JEA) ..	672
34.1 JEA-Konzept	672
34.2 PowerShell-Sitzungskonfiguration erstellen	672
34.3 Sitzungskonfiguration nutzen	676
34.4 Delegierte Administration per Webseite	677
35 Tipps und Tricks zur PowerShell	678
35.1 Alle Anzeigen löschen	678
35.2 Befehlsgeschichte	678
35.3 System- und Hostinformationen	679
35.4 Anpassen der Eingabeaufforderung (Prompt)	680
35.5 PowerShell-Befehle aus anderen Anwendungen heraus starten	681
35.6 ISE erweitern	682
35.7 PowerShell für Gruppenrichtlinienskripte	683
35.8 Einblicke in die Interna der Pipeline-Verarbeitung	686

Teil C:PowerShell im Praxiseinsatz	687
36 Dateisystem	689
36.1 Laufwerke	690
36.2 Ordnerinhalte	695
36.3 Dateieigenschaften verändern	702
36.4 Eigenschaften ausführbarer Dateien	703
36.5 Kurznamen	705
36.6 Lange Pfade	705
36.7 Dateisystemoperationen	706
36.8 Praxislösung: Dateien umorganisieren	706
36.9 Praxislösung: Zufällige Dateisystemstruktur erzeugen	708
36.10 Praxislösung: Leere Ordner löschen	709
36.11 Praxislösung: Geschwindigkeitsmessung des Dateisystems (beim Kopieren von Dateien)	711
36.12 Einsatz von Robocopy in der PowerShell	712
36.13 NTFS-Komprimierung	715
36.14 Dateisystemkataloge	716
36.15 Papierkorb leeren	716
36.16 Dateieigenschaften lesen	717
36.17 Praxislösung: Fotos nach Aufnahmedatum sortieren	717
36.18 Datei-Hash	718
36.19 Finden von Duplikaten	719
36.20 Verknüpfungen im Dateisystem	721
36.21 Komprimierung	726
36.22 Dateisystemfreigaben	730
36.23 Überwachung des Dateisystems	741
36.24 Dateiversionsverlauf	742
36.25 Windows Explorer öffnen	743
36.26 Windows Server Backup	743
37 Festplattenverschlüsselung mit BitLocker	745
37.1 Übersicht über das BitLocker-Modul	746
37.2 Verschlüsseln eines Laufwerks	747
38 Dokumente	748
38.1 Textdateien	748
38.2 CSV-Dateien	750
38.3 Analysieren von Textdateien	753
38.4 INI-Dateien	757
38.5 XML-Dateien	757

38.6	HTML- und Markdown-Dateien	769
38.7	JSON-Dateien	772
38.8	Binärdateien	783
38.9	Praxislösung: Grafikdateien verändern	784
38.10	Praxislösung: Drucken vieler Dateien	785
39	Microsoft Office	786
39.1	Allgemeine Informationen zur Office-Automatisierung per PowerShell	786
39.2	Praxislösung: Terminserien aus Textdateien anlegen in Outlook	787
39.3	Praxislösung: Outlook-Termine anhand von Suchkriterien löschen	789
39.4	Praxislösung: Grafiken aus einem Word-Dokument (DOCX) extrahieren	790
40	Datenbanken	793
40.1	ADO.NET-Grundlagen	793
40.2	Beispieldatenbank	799
40.3	Datenzugriff mit den Bordmitteln der PowerShell	800
40.4	Hilfsroutinen für den Datenbankzugriff (DBUtil.ps1)	813
40.5	Datenzugriff mit den PowerShell-Erweiterungen	816
40.6	Datenbankzugriff mit SQLPS	820
40.7	Datenbankzugriff mit SQLPSX	820
41	Microsoft-SQL-Server-Administration	821
41.1	PowerShell-Integration im SQL Server Management Studio	822
41.2	SQL-Server-Laufwerk „SQLSERVER:“	823
41.3	Die SQLPS-Commandlets	826
41.4	Die SQL Server Management Objects (SMO)	828
41.5	SQLPSX	831
41.6	Microsoft-SQL-Server-Administration mit der PowerShell in der Praxis	838
42	ODBC-Datenquellen	844
42.1	ODBC-Treiber und -Datenquellen auflisten	845
42.2	Anlegen einer ODBC-Datenquelle	846
42.3	Zugriff auf eine ODBC-Datenquelle	847
43	Registrierungsdatenbank (Registry)	849
43.1	Schlüssel auslesen	849
43.2	Schlüssel anlegen und löschen	850
43.3	Laufwerke definieren	850
43.4	Werte anlegen und löschen	851
43.5	Werte auslesen	852
43.6	Praxislösung: Windows-Explorer-Einstellungen	853
43.7	Praxislösung: Massenanlegen von Registry-Schlüsseln	853

44	Computer- und Betriebssystemverwaltung	855
44.1	Computerinformationen	855
44.2	Versionsnummer des Betriebssystems	857
44.3	Zeitdauer seit dem letzten Start des Betriebssystems	857
44.4	BIOS- und Startinformationen	858
44.5	Windows-Produktaktivierung	859
44.6	Umgebungsvariablen	859
44.7	Schriftarten	863
44.8	Computernamen und Domäne	863
44.9	Herunterfahren und Neustarten	864
44.10	Windows Updates installieren	865
44.11	Wiederherstellungspunkte verwalten	869
45	Windows Defender	870
46	Hardwareverwaltung	871
46.1	Hardwarebausteine	871
46.2	Plug-and-Play-Geräte	873
46.3	Druckerverwaltung (ältere Betriebssysteme)	873
46.4	Druckerverwaltung (seit Windows 8 und Windows Server 2012)	875
47	Softwareverwaltung	877
47.1	Softwareinventarisierung	877
47.2	Installation von Anwendungen	880
47.3	Deinstallation von Anwendungen	881
47.4	Praxislösung: Installationstest	882
47.5	Praxislösung: Installierte .NET SDKs aufräumen	883
47.6	Windows 10 Apps verwalten	887
47.7	Installationen mit PowerShell Package Management („OneGet“)	890
47.8	Versionsnummer ermitteln	893
47.9	Servermanager	894
47.10	Windows-Features installieren auf Windows-Clientbetriebssystemen	905
47.11	Praxislösung: IIS-Installation	907
47.12	Softwareeinschränkungen mit dem PowerShell-Modul „AppLocker“	909
48	Prozessverwaltung	915
48.1	Prozesse auflisten	915
48.2	Prozesse starten	916
48.3	Prozesse mit vollen Administratorrechten starten	917
48.4	Prozesse unter einem anderen Benutzerkonto starten	918
48.5	Prozesse beenden	919
48.6	Warten auf das Beenden einer Anwendung	920

49	Windows-Systemdienste	921
49.1	Dienste auflisten	921
49.2	Dienstzustand ändern	924
49.3	Diensteigenschaften ändern	924
49.4	Dienste hinzufügen	925
49.5	Dienste entfernen	926
50	Netzwerk	927
50.1	Netzwerkkonfiguration	927
50.2	DNS-Client-Konfiguration	932
50.3	DNS-Namensauflösung	936
50.4	Erreichbarkeit prüfen (Ping)	937
50.5	Windows Firewall	938
50.6	Remote Desktop (RDP) einrichten	945
50.7	E-Mails senden (SMTP)	946
50.8	Auseinandernehmen von E-Mail-Adressen	947
50.9	Abruf von Daten von einem HTTP-Server	947
50.10	Praxislösung: Linkprüfer für eine Website	954
50.11	Aufrufe von SOAP-Webdiensten	957
50.12	Aufruf von REST-Diensten	960
50.13	File Transfer Protocol (FTP)	962
50.14	Hintergrunddatentransfer mit BITS	963
51	Ereignisprotokolle (Event Log)	967
51.1	Protokolleinträge auslesen	967
51.2	Ereignisprotokolle erzeugen	969
51.3	Protokolleinträge erzeugen	969
51.4	Protokollgröße festlegen	969
51.5	Protokolleinträge löschen	969
52	Leistungsdaten (Performance Counter)	970
52.1	Zugriff auf Leistungsindikatoren über WMI	970
52.2	Get-Counter	971
53	Sicherheitseinstellungen	973
53.1	Aktueller Benutzer	973
53.2	Grundlagen	974
53.3	Zugriffsrechtelisten auslesen	979
53.4	Einzelne Rechteinträge auslesen	980
53.5	Besitzer auslesen	982
53.6	Benutzer und SID	982

53.7	Hinzufügen eines Rechteeintrags zu einer Zugriffsrechteliste	986
53.8	Entfernen eines Rechteeintrags aus einer Zugriffsrechteliste	988
53.9	Zugriffsrechteliste übertragen	990
53.10	Zugriffsrechteliste über SDDL setzen	991
53.11	Zertifikate verwalten	992
54	Optimierungen und Problemlösungen	995
54.1	PowerShell-Modul „TroubleshootingPack“	995
54.2	PowerShell-Modul „Best Practices“	999
55	Active Directory	1001
55.1	Benutzer- und Gruppenverwaltung mit WMI	1003
55.2	Einführung in System.DirectoryServices	1003
55.3	Basiseigenschaften	1015
55.4	Benutzer- und Gruppenverwaltung im Active Directory	1017
55.5	Verwaltung der Organisationseinheiten	1025
55.6	Suche im Active Directory	1026
55.7	Navigation im Active Directory mit den PowerShell Extensions	1033
55.8	Verwendung der Active-Directory-Erweiterungen von www.IT-Visions.de	1034
55.9	PowerShell-Modul „Active Directory“ (ADPowerShell)	1036
55.10	PowerShell-Modul „ADDSDeployment“	1065
55.11	Informationen über die Active Directory-Struktur	1068
56	Gruppenrichtlinien	1071
56.1	Verwaltung der Gruppenrichtlinien	1071
56.2	Verknüpfung der Gruppenrichtlinien	1073
56.3	Gruppenrichtlinienberichte	1075
56.4	Gruppenrichtlinienvererbung	1077
56.5	Weitere Möglichkeiten	1078
57	Lokale Benutzer und Gruppen	1079
57.1	Modul „Microsoft.PowerShell.LocalAccounts“	1079
57.2	Lokale Benutzerverwaltung in älteren PowerShell-Versionen	1080
58	Microsoft Exchange Server	1083
58.1	Daten abrufen	1083
58.2	Postfächer verwalten	1084
58.3	Öffentliche Ordner verwalten	1085
59	Internet Information Services (IIS)	1086
59.1	Überblick	1086
59.2	Navigationsprovider	1088

59.3	Anlegen von Websites	1090
59.4	Praxislösung: Massenanlegen von Websites	1091
59.5	Ändern von Website-Eigenschaften	1093
59.6	Anwendungspool anlegen	1094
59.7	Virtuelle Verzeichnisse und IIS-Anwendungen	1095
59.8	Website-Zustand ändern	1095
59.9	Anwendungspools starten und stoppen	1096
59.10	Löschen von Websites	1096
60	Virtuelle Systeme mit Hyper-V	1097
60.1	Das Hyper-V-Modul von Microsoft	1098
60.2	Die ersten Schritte mit dem Hyper-V-Modul	1100
60.3	Virtuelle Maschinen anlegen	1104
60.4	Umgang mit virtuellen Festplatten	1110
60.5	Konfiguration virtueller Maschinen	1113
60.6	Praxislösungen: Ressourcennutzung überwachen	1117
60.7	Dateien kopieren in virtuelle Systeme	1119
60.8	PowerShell Management Library for Hyper-V (für ältere Betriebssysteme)	1120
61	Windows Nano Server	1123
61.1	Das Konzept von Nano Server	1123
61.2	Einschränkungen von Nano Server	1125
61.3	Varianten des Nano Servers	1127
61.4	Installation eines Nano Servers	1127
61.5	Docker-Image	1128
61.6	Fernverwaltung mit PowerShell	1129
61.7	Windows Update auf einem Nano Server	1131
61.8	Nachträgliche Paketinstallation	1131
61.9	Abgespeckter IIS unter Nano Server	1133
61.10	Nano-Serververwaltung aus der Cloud heraus	1134
62	Docker-Container	1135
62.1	Container-Varianten für Windows	1135
62.2	Docker-Installation auf aktuellem Windows 10 und Windows 11	1139
62.3	Docker-Installation auf älteren Windows 10-Clients	1147
62.4	Docker-Installation auf Windows Server	1149
62.5	Docker PowerShell installieren	1151
62.6	Docker-Basiswissen	1152
62.7	Container mit modernem .NET	1155
62.8	Container mit IIS-Webserver und klassischem ASP.NET	1164
62.9	Container mit Linux und PowerShell 7	1173

62.10	Container mit Linux und Microsoft SQL Server	1175
62.11	Docker-Container mit Visual Studio	1177
62.12	Weitere Container-Befehle	1182
63	Microsoft Azure	1188
63.1	Azure-Konzepte	1188
63.2	Kommandozeilenwerkzeuge für die Azure-Verwaltung	1190
63.3	Benutzeranmeldung und Informationsabfrage	1193
63.4	Azure Ressourcen-Gruppen	1194
63.5	Azure Web-Apps	1194
63.6	Azure SQL Server	1196
63.7	Azure Kubernetes Services (AKS)	1197
63.8	Azure DevOps (ADO)	1221
64	Grafische Benutzeroberflächen (GUI)	1242
64.1	Einfache Nachfragedialoge	1242
64.2	Einfache Eingabe mit Inputbox	1243
64.3	Komplexere Eingabemasken	1244
64.4	Universelle Objektdarstellung	1246
64.5	WPF PowerShell Kit (WPK)	1247
64.6	Direkte Verwendung von WPF	1255
Teil D: Profiwissen – Erweitern der PowerShell		1257
65	Unit Tests mit Pester	1259
65.1	Einführung in das Konzept des Unit Testing	1259
65.2	Pester installieren	1260
65.3	Befehle in Pester	1260
65.4	Testen einer PowerShell-Funktion	1261
65.5	Testgenerierung	1262
65.6	Tests starten	1262
65.7	Prüf-Operationen	1264
65.8	Mock-Objekte	1264
65.9	Test von Dateisystemoperationen	1265
66	Entwicklung von Commandlets in der PowerShell-Skriptsprache	1267
66.1	Aufbau eines skriptbasierten Commandlets	1267
66.2	Verwendung per Dot Sourcing	1269
66.3	Parameterfestlegung	1270
66.4	Fortgeschrittene Funktion (Advanced Function)	1276

66.5	Mehrere Parameter und Parametersätze	1279
66.6	Unterstützung für Sicherheitsabfragen (-whatif und -confirm)	1281
66.7	Kaufmännisches Beispiel: Test-CustomerID	1283
66.8	Erweitern bestehender Commandlets durch Proxy-Commandlets	1286
66.9	Dokumentation	1292
67	Entwicklung eigener Commandlets mit C#	1296
67.1	Technische Voraussetzungen	1297
67.2	Grundkonzept der .NET-basierten Commandlets	1299
67.3	Schrittweise Erstellung eines minimalen Commandlets	1301
67.4	Erstellung eines Commandlets mit einem Rückgabeobjekt	1309
67.5	Erstellung eines Commandlets mit mehreren Rückgabeobjekten	1311
67.6	Erstellen eines Commandlets mit Parametern	1315
67.7	Verarbeiten von Pipeline-Eingaben	1317
67.8	Verkettung von Commandlets	1320
67.9	Fehlersuche in Commandlets	1324
67.10	Statusinformationen	1327
67.11	Unterstützung für Sicherheitsabfragen (-whatif und -confirm)	1332
67.12	Festlegung der Hilfeinformationen	1334
67.13	Erstellung von Commandlets für den Zugriff auf eine Geschäftsanwendung	1339
67.14	Konventionen für Commandlets	1340
67.15	Weitere Möglichkeiten	1342
68	PowerShell-Module erstellen	1343
68.1	Erstellen eines Skriptmoduls	1343
68.2	Praxislösung: Umwandlung einer Skriptdatei in ein Modul	1345
68.3	Erstellen eines Moduls mit Binärdateien	1345
68.4	Erstellen eines Moduls mit Manifest	1346
68.5	Erstellung eines Manifest-Moduls mit Visual Studio	1353
69	Hosting der PowerShell	1355
69.1	Voraussetzungen für das Hosting	1356
69.2	Hosting mit PSHost	1357
69.3	Vereinfachtes Hosting seit PowerShell 2.0	1360
Anhang A: Crashkurs Objektorientierung		1363
Anhang B: Crashkurs .NET		1371
B.1	Was ist das .NET Framework?	1374
B.2	Was ist .NET Core/.NET?	1375
B.3	Eigenschaften von .NET	1376

B.4	.NET-Klassen	1377
B.5	Namensgebung von .NET-Klassen (Namensräume)	1377
B.6	Namensräume und Softwarekomponenten	1379
B.7	Bestandteile einer .NET-Klasse	1380
B.8	Vererbung	1381
B.9	Schnittstellen	1381
Anhang C: Weitere Informationen im Internet		1382
Anhang D: Abkürzungsverzeichnis		1383
Stichwortverzeichnis		1407

Vorwort

Liebe Leserin, lieber Leser,

willkommen zur aktuellen Auflage meines PowerShell-Buchs! Es handelt sich hierbei um die fünfte Auflage des Windows PowerShell 5-Buches und die neunte Auflage des PowerShell-Buches insgesamt, das erstmalig 2007 bei Addison-Wesley erschienen ist.

Was ist das Thema dieses Buchs?

Das vor Ihnen liegende Fachbuch behandelt die Windows PowerShell in der Version 5.1 sowie die plattformneutrale PowerShell 7.2 von Microsoft wie auch ergänzende Werkzeuge von Microsoft und Drittanbietern (z. B. PowerShell Community Extensions). Es gibt in dem Buch auch Ausblicke auf die PowerShell 7.3, die derzeit in der Entwicklung ist.

Das Buch ist aber auch für Sie geeignet, wenn Sie noch Windows PowerShell 2.0/3.0/4.0/5.0 oder PowerShell Core 6.x bzw. PowerShell 7.0/7.1 einsetzen. Welche Funktionen neu hinzugekommen sind, wird jeweils in diesem Buch erwähnt.

Wer bin ich?

Mein Name ist Holger Schwichtenberg, ich bin derzeit 49 Jahre alt und habe im Fachgebiet Wirtschaftsinformatik promoviert. Ich lebe (in Essen, im Herzen des Ruhrgebiets) davon, dass mein Team und ich im Rahmen unserer Firma www.IT-Visions.de anderen Unternehmen bei der Entwicklung von .NET-, Web- und PowerShell-Anwendungen beratend und schulend zur Seite stehen. Zudem entwickeln wir im Rahmen der MAXIMAGO GmbH (www.MAXIMAGO.de) Software im Auftrag von Kunden in zahlreichen Branchen.

Es ist nur ein Hobby, IT-Fachbücher zu schreiben, denn damit kann man als Autor kaum Geld verdienen. Dieses Buch ist, unter Mitzählung aller nennenswerten Neuauflagen, das 92. Buch, das ich allein oder mit Co-Autoren geschrieben habe. Meine weiteren Hobbys sind Mountain Biking, Fotografie und Reisen.

Natürlich verstehe ich das Bücherschreiben auch als Werbung für die Arbeit unserer Unternehmen, und wir hoffen, dass der ein oder andere von Ihnen uns beauftragen wird, Ihre Organisation durch Beratung, Schulung und Auftragsentwicklung zu unterstützen.

Wer sind Sie?

Damit Sie den optimalen Nutzen aus diesem Buch ziehen können, möchte ich – so genau es mir möglich ist – beschreiben, an wen sich dieses Buch richtet. Hierzu habe ich einen Fragebogen ausgearbeitet, mit dem Sie schnell erkennen können, ob das Buch für Sie geeignet ist.

Sind Sie Systemadministrator in einem Windows-Netzwerk?	<input type="radio"/> Ja	<input type="radio"/> Nein
Laufen die für Sie relevanten Computer mit den von PowerShell unterstützten Betriebssystemen? (Windows 7/8/8.1/10/11, Windows Server 2008/2008 R2/2012/2012 R2/2016/2019/2022, macOS, Linux)	<input type="radio"/> Ja	<input type="radio"/> Nein
Sie besitzen zumindest rudimentäre Grundkenntnisse im Bereich des (objektorientierten) Programmierens?	<input type="radio"/> Ja	<input type="radio"/> Nein
Wünschen Sie einen kompakten Überblick über die Architektur, Konzepte und Anwendungsfälle der PowerShell?	<input type="radio"/> Ja	<input type="radio"/> Nein
Sie können auf Schritt-für-Schritt-Anleitungen verzichten?	<input type="radio"/> Ja	<input type="radio"/> Nein
Sie können auf formale Syntaxbeschreibungen verzichten und lernen lieber an aussagekräftigen Beispielen?	<input type="radio"/> Ja	<input type="radio"/> Nein
Sie erwarten nicht, dass in diesem Buch alle Möglichkeiten der PowerShell detailliert beschrieben werden?	<input type="radio"/> Ja	<input type="radio"/> Nein
Sind Sie, nachdem Sie ein Grundverständnis durch dieses Buch gewonnen haben, bereit, Detailfragen in der Dokumentation der PowerShell, von .NET und WMI nachzuschlagen, da das Buch auf rund 1400 Seiten nicht alle Details erläutern, sondern – in dem Sinn „Hilfe zur Selbsthilfe“ – nur ausgewählte Aspekte darstellen kann, anhand deren Sie dann Ihre eigenen Lösungen für Ihre spezifischen Szenarien entwickeln?	<input type="radio"/> Ja	<input type="radio"/> Nein

Wenn Sie alle obigen Fragen mit „Ja“ beantwortet haben, ist dieses Fachbuch richtig für Sie. In anderen Fällen sollten Sie sich erst mit einführender Literatur beschäftigen.

Was ist neu in diesem Buch?

Gegenüber der vorherigen Auflage zur PowerShell 5.1/PowerShell 7.0 wurde das Buch um die neuen Commandlets, Funktionen und Operationen in PowerShell 7.1 und 7.2 erweitert.

Zudem wurden die bestehenden Inhalte des Buchs an vielen Stellen optimiert. Das Kapitel zu „Docker-Container“ wurde in weiten Teilen überarbeitet. Zum Dateisystem, zur Dokumentenverarbeitung, zum Netzwerk, zu Hyper-V und zu Azure DevOps-Pipelines habe ich Praxislösungen ergänzt.

Zudem wurde das Feedback einiger Leser eingearbeitet, um Beispiele und Texte zu optimieren.

Sind in diesem Buch alle Features der PowerShell beschrieben?

Die PowerShell umfasst mittlerweile mehrere Tausend Commandlets mit jeweils zahlreichen Optionen. Zudem gibt es unzählige Erweiterungen mit vielen Hundert weiteren Commandlets. Außerdem existieren zahlreiche Zusatzwerkzeuge. Es ist allein schon aufgrund der Vorgaben des Verlags für den Umfang des Buchs nicht möglich, alle Commandlets und Parameter hier auch nur zu erwähnen. Zudem habe ich – obwohl ich selbst fast jede Woche mit der PowerShell in der Praxis arbeite – immer noch nicht alle Commandlets und alle Parameter jemals selbst eingesetzt.

Ich beschreibe in diesem Buch, was ich selbst in der Praxis, in meinen Schulungen und bei Kundeneinsätzen verwende. Es macht auch keinen Sinn, hier jedes Detail der PowerShell zu dokumentieren. Stattdessen gebe ich Ihnen **Hilfe zur Selbsthilfe**, damit Sie die Konzepte gut

verstehen und sich dann Ihre spezifischen Lösungen anhand der Dokumentation selbst erarbeiten können.

Wie aktuell ist dieses Buch?

Die Informationstechnik hat sich immer schon schnell verändert. Seit aber auch Microsoft die Themen „Agilität“ und „Open Source“ für sich entdeckt hat, ist die Entwicklung nicht mehr nur schnell, sondern zum Teil rasant:

- Es erscheinen in kurzer Abfolge immer neue Produkte.
- Produkte erscheinen schon in frühen Produktstadien als „Preview“ mit Versionsnummern wie 0.1.
- Produkte ändern sich sehr häufig, teilweise im Abstand von drei Wochen (z. B. Visual Studio und Azure DevOps).
- Aufwärts- und Abwärtskompatibilität ist kein Ziel bei Microsoft mehr. Es wird erwartet, dass Sie Ihre Lösungen ständig den neuen Gegebenheiten anpassen.
- Produkte werden nicht mehr so ausführlich dokumentiert wie früher. Teilweise erscheint die Dokumentation erst deutlich nach dem Erscheinen der Software. Oft bleibt die Dokumentation auch dauerhaft lückenhaft.
- Produkte werden schnell auch wieder abgekündigt, wenn sie sich aus der Sicht der Hersteller bzw. aufgrund des Nutzerfeedbacks nicht bewährt haben.



HINWEIS: Nicht nur Microsoft geht so vor, sondern viele andere Softwarehersteller (z. B. Google) agieren genauso.

Unter diesen neuen Einflüssen steht natürlich auch dieses etablierte Fachbuch. Leider kann man ein gedrucktes Buch nicht so schnell ändern wie Software. Verlage definieren nicht unerhebliche Mindestauflagen, die abverkauft werden müssen, bevor neu gedruckt werden darf. Das E-Book ist keine Alternative. Die Verkaufszahlen zeigen, dass nur eine kleine Menge von Lesern technischer Literatur ein E-Book statt eines gedruckten Buchs kauft. Das E-Book wird offenbar nur gerne als Ergänzung genommen. Das kann ich gut verstehen, denn ich selbst lese auch lieber gedruckte Bücher und nutze E-Books nur für eine Volltextsuche.

Daher kann es passieren, dass – auch schon kurz nach dem Erscheinen dieses Buchs – einzelne Informationen in diesem Buch nicht mehr zu neueren Versionen passen. Wenn Sie so einen Fall feststellen, schreiben Sie bitte eine Nachricht an mich (siehe unten). Ich werde dies dann in Neuauflagen des Buchs berücksichtigen.

Zudem ist zu beachten, dass zwischen Abgabe des Manuskripts beim Verlag und Auslieferung des Buchs aus der Druckerei an den Buchhandel meist vier bis fünf Monate liegen.

Welche PowerShell-Versionen werden besprochen?

Das Buch bespricht sowohl die Windows PowerShell 5.1 als auch die PowerShell 7.2. Es gibt in dem Buch auch Ausblicke auf die PowerShell 7.3, die derzeit in der Entwicklung ist.

- Bei der Windows PowerShell 5.1 wird die RTM-Version besprochen, die Microsoft in der aktuellen Version von Windows 10/11 bzw. Windows Server 2019/2022 mitliefert.

- Bei PowerShell 7.2 nutzen wir die RTM-Version vom 8. November 2021 ein.
- Bei PowerShell 7.3 gibt es zum Redaktionsschluss erst die Version Preview 2. Die PowerShell 7.3 wird voraussichtlich Ende 2022 erscheinen.

Warum behandelt das Buch auch noch Version 5.1 und nicht nur Version 7.2?

Windows PowerShell 5.1 ist heute in den Unternehmen in Deutschland der Standard, denn diese Version der PowerShell wird mit Windows 10/11 und Windows Server 2016, Windows Server 2019 sowie Windows Server 1709, Windows Server 1909 und Windows Server 2022 ausgeliefert.

Die PowerShell 7.2 wird bisher mit keinem einzigen Betriebssystem ausgeliefert, sondern muss getrennt heruntergeladen und installiert werden. Eine Zusatzinstallation ist in vielen Unternehmen mit stark abgeschotteten Systemen gar nicht möglich.

Ein zweites Argument für die Beibehaltung der Version 5.1 in diesem Fachbuch ist, dass die PowerShell 7.2 der Windows PowerShell 5.1 funktional immer noch nicht ganz ebenbürtig ist. Einige Befehle sind weiterhin nur in der Windows PowerShell verfügbar.

Daher wird die Windows PowerShell 5.1 auch weiterhin eine große Bedeutung haben und in diesem Buch auch weiterhin behandelt.

Welche Betriebssysteme werden besprochen?

Der Schwerpunkt des Buchs liegt auf der Nutzung der PowerShell unter Windows. Es gibt Hinweise und Beispiele für die Nutzung der PowerShell unter Linux (am Beispiel Ubuntu) und macOS.

Bei Windows gibt es Hinweise auf Unterschiede zwischen verschiedenen Windows-Varianten (Client/Server) und Windows-Versionen.

Auch wenn Windows 11 bereits erschienen ist, ist Windows 10 das im professionellen Einsatz vorherrschende Betriebssystem. Das Buch geht auf existierende kleinere Unterschiede zwischen Windows 10 und Windows 11 ein, die meisten Screenshots sind aber mit Windows 10 gemacht. Einige Screenshots sind mit älteren Windows-Versionen geschossen, was aber kein Problem ist, denn inhaltlich hat sich nichts geändert (nur optisch an der Titelleiste und der Schriftart).

Woher bekommt man die Beispiele aus diesem Buch?

Unter <http://www.powershell-doktor.de/leser> biete ich ein **ehrenamtlich betriebenes** Webportal für Leser meiner Bücher an. Bei der Erstregistrierung müssen Sie das Lösungswort **Boba Fett** angeben. Nach erfolgter Registrierung erhalten Sie dann ein persönliches Zugangskennwort per E-Mail.

In diesem Portal können Sie

- die Codebeispiele aus diesem Buch in einem Archiv herunterladen,
- eine PowerShell-Kurzreferenz „Cheat Sheet“ (zwei DIN-A4-Seiten als Hilfe für die tägliche Arbeit) kostenlos herunterladen sowie
- Feedback zu diesem Buch geben (Bewertung abgeben und Fehler melden).

Kurzreferenz ("Cheat Sheet") Windows PowerShell

Autor: Dr. Holger Schwichtenberg (www.IT-Visions.de) 11.5.2 / 22.03.2018



Hilfe

- Alle installierten Module
Get-Module -ListAvailable | # Name, ModulTyp, ExportedCommands
- Alle Befehle mit "Get-"
Get-Command Get-*
- Alle Befehle aus einem Modul
Get-Command | Where-Object module -like "ActiveDirectory" | FT Name, Module
- Komplette Hilfe zu einem Befehl
Get-Help Stop-Process -full
- Auflisten aller "About"-Dokumente
Get-Help about
- Anzeigen des Hilfedokuments zu WMI
Get-Help about WMI
- Anzeigen aller Eigenschaften der Ergebnisobjekte
Get-Watch | Get-Member

Wichtige Navigations-Commandlets

Mit den Navigations-Commandlets kann man nicht nur in Dateisystem, sondern auch andere Flächen und hierarchischen Mengen arbeiten. z.B. Registry (HKLM, HKCU), Umgebungsvariablen (env), Zertifikate (cert), Active Directory (AD), usw. arbeiten. z.B.

Dir HKLM:\Software\NewItems HKLM:\Software\ITVisions ID HKLM:\Software\ITVisions

Get-PSDrive	Laufwerkliste
Get-Location (pwd)	Abrufen des aktuellen Standorts
Set-Location (cd)	Festlegung des aktuellen Standorts
Get-Item (g)	Holt ein Element
Get-Childitem (dir, ls, gci)	Auflisten der Unter Elemente
Get-Content (type, cat, gc)	Abrufen eines Elementinhalts (z.B. Dateiinhalt)
Set-Content (li)	Elementinhalt festlegen
Add-Content (cat, add)	Elementinhalt ergänzen
New-Item (ni, mkdir)	Erstellen eines Elements (Art oder Datei)
Get-ItemProperty (gpi)	Attribut abrufen
Set-ItemProperty (spi)	Attribut eines Elements festlegen (z.B. ändern wenn nicht vorhanden)
Remove-Item (del, rmdir, mv, erase)	Element löschen
Move-Item (move, mv)	Element verschieben
Copy-Item (copy, cp, cpi)	Element kopieren
Rename-Item (ren, mv)	Element umbenennen

Active Directory-Commandlets

Diese Commandlets erfordern das Active Directory-PowerShell-Modul auf dem Client und ADWS (Active Directory WebServices) auf dem AD-Server:

Get-ADObject	Abrufen beliebiger Objekte aus dem AD
Get-User, Get-ADGroup, Get-ADOrganizationalUnit, Get-ADDomain, Get-ADComputer, ...	Abruf von spezifischen AD-Elementen
Set-ADObject, Set-ADUser, Set-ADGroup, Set-ADComputer, ...	Setzen von Eigenschaften eines Objekts
New-ADUser, New-ADGroup, New-ADOrganizationalUnit, ...	Anlegen eines neuen AD-Objekts
Remove-ADObject	Löschen eines AD-Objekts
Rename-ADObject	Umbenennen eines AD-Objekts
Move-ADObject	Verschieben eines AD-Objekts
Set-ADAccountPassword	Festlegen eines Kennworts
Get-ADGroupMember	Liste der Gruppenmitglieder
Add-ADGroupMember	Mitglied einer Gruppe hinzufügen
Remove-ADGroupMember	Mitglied aus einer Gruppe entfernen

Weitere wichtige Commandlets

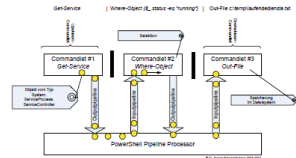
Get-Date / Set-Date	Datum und Zeit abrufen/festlegen
Get-Service	Windows-Systemdienste
Start-Stop/Suspend/Resume-Service	Dienststatus ändern
Get-Process	Laufende Prozesse
Start-Process/Stop-Process	Prozess starten/beenden
Wait-Process	Warten auf Ende eines Prozesses
Get-Counter	Leistungsindikatoren abrufen
Get-Eventlog	Ereignisprotokolleinträge
Write-Eventlog	Eintrag im Ereignisprotokoll
Get-Eventing	Größe des Ereignisprotokolls setzen
Get-Random	Zufallszahl
Find-Module	Module in PowerShell Gallery suchen
Install-Module	Module aus PowerShell Gallery herunterladen und installieren

Pipelining-Grundkonzept

Beliebig viele Commandlets können mit dem Pipe-Symbol | verkett werden. Get-Service | where { \$_.status -eq 'running' } | Out-File c:\temp\laufendediene.txt

Alternativ kann man Zwischenergebnisse in Variablen, die mit \$ beginnen, ablegen. \$service = Get-Service | Where-Object { \$_.status -eq 'running' }; Out-File c:\temp\laufendediene.txt

Die Pipeline befördert .NET-Objekte. Die Beförderung ist synchron (außer bei eigenen „blockierenden“ Commandlets wie Set-Objekt).



Wichtige Pipelining-Commandlets

Where-Object (where, fi)	Filtern mit Bedingungen
Select-Object (select)	Abschneiden der Ergebnismenge vorne/hinten bzw. Reduktion der Attribute der Objekte
Sort-Object (sort)	Sortieren der Objekte
Group-Object (group)	Gruppieren der Objekte
ForEach-Object (ForEach-Obj, %)	Schleife über alle Objekte
Get-Member (gmi)	Aufliste der Methoden (Reflection)
Measure-Object (measure)	Berechnung: min, max, sum, average
Compare-Object (compare, diff)	Vergleichen von zwei Objektinstanzen

Vergleichsoperatoren

Da die Zeichen < und > für Umkehrungen der Ausgabemenge verwendet werden, können PowerShell eher ungewöhnliche Operatoren zum Einsatz:

Vergleich unter Ignorierung der Groß-/Kleinschreibung	Vergleich unter Berücksichtigung der Groß-/Kleinschreibung	Bedeutung
-lt / -lti	-lti	Kleiner
-le / -le	-le	Kleiner oder gleich
-gt / -gt	-gt	Größer
-ge / -ge	-ge	Größer oder gleich
-eq / -eq	-eq	Gleich
-ne / -ne	-ne	Nicht gleich
-like / -like	-like	Ähnlichkeit zwischen Zeichenketten, Einsatz von Platzholdern (*) und ? möglich
-notlike / -notlike	-notlike	Keine Ähnlichkeit zwischen Zeichenketten, Einsatz von Platzholdern (*) und ? möglich
-match / -match	-match	Vergleich mit regulärem Ausdruck

Vorderseite der PowerShell-Kurzreferenz

Kurzreferenz ("Cheat Sheet") Windows PowerShell

Autor: Dr. Holger Schwichtenberg (www.IT-Visions.de) 11.5.2 / 22.03.2018



-notmatch / -notmatch	-notmatch	Stimmt nicht mit regulärem Ausdruck überein
-is		Typgleichheit, z.B. (Get-Date) -is [DateTime]
-in / -contains		Ist enthalten in Menge
-notin / -notcontains		Ist nicht enthalten in Menge

Für die logische Verknüpfung werden -and und -or sowie -not (alles !) verwendet. Beispiel: ((MB > 150 + \$?) -and 100KB) -and (\$? -lt 2KB) KB, MB, GB, TB und PB sind gültige Abkürzungen für Speichergrößen.

Ein- und Ausgabe-Commandlets

Format-Table (ft)	Tabellenausgabe
Format-List (fl)	detaillierte Liste
Format-Wide (fw)	mehrspaltige Liste
Out-Host (oh)	Ausgabe an Konsolen mit Optionen zur Farbe und selbstenwertigen Ausgabe
Out-GridView (ogv)	Grafische Tabelle mit Filter- und Sortieroptionen
Out-File	Speichern in Datei
Out-Printer (ip)	Ausgabe an Drucker
Out-CliBoard	Ausgabe in Zwischenablage
Out-Speech	Sprachausgabe (PSSC)
Out-Null	Die Objekte der Pipeline werden nicht weitergegeben
Read-Host	Eingabe von Konsole einlesen
Import-Export-CSV	CSV-Datei importieren/exportieren
Import-Export-CliML	XML-Datei importieren/exportieren

Benutzerdefinierte Tabellenausgabe
Get-Process | @('Label1', 'Expression' = {\$_ .ID}; Width=5), @('Label2', 'Name'; Expression = {\$_ .ProcessName}; Width=20), @('Label3', 'Spacer'; Expression = {\$_ .WorkingSet4 / 1MB}; Width=1); Format-Table (0:0000:0:0)

Zeichenketten und Ausdrücke

Einbetten einer Variablen in eine Zeichenkette "Der Befehl ist \$Befehl"
Hier muss {} zur Abgrenzung vom Doppelpunkt eingesetzt werden "Z(Befehl); erfolgreich ausgeführt"
Der Unterdruck in \$() geklammert werden "(\$(\$Process.Count) Objekte in der Ergebnismenge)"
Einsatz des Formatoperators
Get-Process | % { (0..40) | (1..0,000,00)MB -f \$(\$_.Name, (\$_.WorkingSet4 / 1MB)); }
Auflösen einer Zeichenkette als Befehl
\$Befehl = "Get-Service -q"
\$Befehl = " |" where status -eq "Running"
\$Ergebnis = Invoke-Expression \$Befehl

Objektorientierter Zugriff auf Pipeline-Objekte

Anzahl der Objekte in der Pipeline (Get-Service | where { \$_.status -eq 'Running' }) .Count

Einzige Eigenschaften der Pipeline-Objekte ausgeben (Get-Date).Dir(\$Dir) (Get-Process) .Name (Get-Process | sort -w -desc)[0].Name

Methodenaufruf in allen Pipeline-Objekten (Get-Process explore) | sort -w -desc; Kill()

PowerShell-Datentypen

[char], [string]	[byte], [int], [long]	[int?]
[bool]	[single], [double]	[array], [Hashtable]
[DateTime]		[VM], [ADSI]

PowerShell-Skriptsprache

Bedingung (if) (Get-Date).Year -le 2014 { "AP" } else { "NW" }

Schleifen
for(\$i = 1; \$i -le 10; \$i++) { \$i }
while(\$i -le 10) { \$i; \$i++ }
do { \$i = 1; while (\$i -le 10) { } }
foreach (\$p in (Get-Process explore)) { \$p.Kill() }

Unterfunktion mit Pflichtparameter und optionalen Parameter
function Get-DLL([Parameter(Mandatory=\$)]\$Name, [string]\$Dir = "") {
return Get-Childitem \$Dir -File -Filter "\$Name" }
Get-DLL -c:Windows\System32

NET Framework-Klassen

PowerShell kann alle auf dem lokalen System vorhandenen .NET-Klassen auch direkt (d.h. ohne Einsatz von Commandlets) verwenden.

Zugriff auf statische Mitglieder (System.Environment::MachineName (System.Console).BasePath, 500, 500)

Instanzierung und Zugriff auf Instanzmitglieder
\$b = New-Object System.DirectoryServices.DirectoryEntry("WinNT://Server/HT")
\$b.FullName
\$b.Description = "Autor des PowerShell Cheat Sheets"
\$b.SetInfo()

Zusätzliche Assembly laden und nutzen (System.Reflection.Assembly::LoadWithPartialName("Microsoft.VisualBasic") \$env:obj = [Microsoft.VisualBasic.Interaction]::InputBox("Frage", "Titel")

Component Object Model (COM)

PowerShell kann alle installierten COM-Komponenten verwenden. \$ie = New-Object -com "InternetExplorer.Application" \$ie.Navigate("http://www.powershell-doktor.de") \$ie.Visible = \$true

Windows Management Instrumentation (WMI)

PowerShell kann alle lokalen oder entfernten WMI-Klassen verwenden. Get-CimData -Namespace root\cimv2 -Computer MyServer

Liste aller WMI-Klassen aus einem Nomenraum von einem Computer Get-CimData -Namespace root\cimv2 -Computer MyServer

Liste aller Instanzen einer WMI-Klasse auf einem Computer Get-CimData -Namespace root\cimv2 -Computer MyServer

WQL-Abfrage auf einem Computer Get-CimData -Query "Select * from Win32_NetworkAdapter where adapterType like '802.11'" -Computer MyServer

Zugriff auf eine Instanz und Änderung der Instanz \$c = Get-CimInstance Win32_LogicalDisk -Namespace root\cimv2 -Filter "DriveID='C:'" -Computer MyServer \$c.VolumeName = "System" \$c.CimInstance \$c

Alternativ mit allen WMI-Commandlets \$c = (WMI) "MyServer\root\cimv2:Win32_LogicalDisk.DeviceID='C:'" \$c.VolumeName = "System" \$c.Put()

Aufruf einer WMI-Methode Invoke-CimMethod -Path "MyServer\root\cimv2:Win32_ComputerSystem.Name='MyServer' -Name 'Rename' -ArgumentList 'MyNewServer'

Links

technet.microsoft.com/scriptcenter
blogs.msdn.com/powershell
www.powershell.com
www.gotwired.de
www.it-visions.de/scripting/powershell

Über den Autor

Dr. Holger Schwichtenberg gehört zu den bekanntesten Experten für die Programmierung mit Microsoft-Produkten in Deutschland. Er hat zahlreiche Bücher zu .NET und PowerShell veröffentlicht und spricht regelmäßig auf Fachkonferenzen. Er hat mehrere Bücher zur PowerShell geschrieben. Sie können ihn und sein Team für Schulungen, Beratungen und Projekte buchen. E-Mail: anfragen@IT-Visions.de



Rückseite der PowerShell-Kurzreferenz

Alle registrierten Leser erhalten auch meinen Newsletter (zwei- bis viermal im Jahr) mit aktuellen Produktinformationen, Einladungen zu kostenlosen Community-Veranstaltungen sowie Vergünstigungen bei unseren öffentlichen Seminaren zu .NET und zur PowerShell.

Wie sind die Programmcodebeispiele organisiert?

Die Beispiele sind in der Archivdatei (.zip) organisiert nach den Buchteilen und innerhalb der Buchteile nach Kapitelnamen nach folgendem Schema:

Buchteilname\Kapitelname\Dateiname

Die Namen sind zum Teil etwas verkürzt (z. B. „Einsatzgebiete“ statt „PowerShell im Praxis-einsatz“), da sich sonst zu lange Dateinamen ergeben.

In diesem Buch wird für den Zugriff auf die Skriptdateien das x:-Laufwerk verwendet. Bitte legen Sie entweder ein Laufwerk x: an oder ändern Sie den Laufwerksbuchstaben in den Skripten.

```
PS T:\> dir x:\

Verzeichnis: x:\

Mode                LastWriteTime         Length Name
----                -
d-r---             29.06.2017   23:56           1_Basiswissen
d-r---             28.06.2017   17:09           2_Aufbauwissen
d-r---             02.06.2017   10:38           3_Einsatzgebiete
d-r---             30.06.2017   17:22           4_Profiwissen
```

Verzeichnisstruktur der Beispielsammlung mit vier Hauptordnern entsprechend den vier Buchteilen

```
PS T:\> dir x:\1_Basiswissen\

Verzeichnis: x:\1_Basiswissen

Mode                LastWriteTime         Length Name
----                -
d-----             29.06.2017   23:56           Aliase
d-r---             24.04.2017   09:52           Ausgaben
d-r---             30.05.2017   00:28           Commandlets
d-----             26.06.2017   10:40           ErsteSchritte
d-r---             29.06.2017   23:34           Hilfe
d-----             30.05.2017   20:59           Module
d-r---             26.03.2014   12:49           Navigation
d-r---             04.06.2017   11:21           Pipelining
d-----             30.05.2017   21:15           PowerShellLanguage
d-----             29.05.2017   23:57           PowerShell100P
d-----             30.06.2017   18:47           PScore
d-r---             30.05.2017   20:46           Scripting
d-r---             26.03.2014   12:49           TippsAndTricks
d-r---             26.03.2014   12:49           Werkzeuge
d-r---             26.03.2014   12:49           WPS versus VBS
d-----             03.05.2016   14:12           Zeichenkettenbearbeitung
```

Inhalt eines der Hauptordner aus der vorherigen Abbildung, d. h. eines Buchteils

Im Buch werden Sie außerdem noch Zugriffe auf ein w:-Laufwerk finden. Dies sind Dateisystemordner mit Dokumenten, die in den Skripten verarbeitet werden. Sofern die Dateien einen bestimmten Inhalt haben müssen (Eingabedateien für Skripte), dann finden Sie diese Eingabedateien auch in der Archivdatei in dem Ordner, wo sich das Skript befindet (oder einem Unterordner). In einigen Fällen sind die konkreten Dateiinhalte aber gar nicht relevant (z. B. für ein Skript, das die Größen von Dateien ermittelt). In diesem Fall können Sie anstelle des w:-Laufwerks jedes beliebige Ihrer eigenen Laufwerke verwenden.

Warum gendern Sie nicht in diesem Buch?

Während ich in einigen Medien und Softwareprodukten (z. B. dem virtuellen Klassenraum <https://VK.IT-Visions.de>) das Gendern bereits verwende, habe ich in diesem Buch aufgrund der Lesbarkeit und des notwendigen Umfangs der Änderungen darauf verzichtet. Selbstverständlich spreche ich aber alle Personen jeglichen Geschlechts gleichermaßen an.

Grundsätzlich stehe ich dem Gendern offen gegenüber, bin aber sehr gespannt, wie sich die offiziellen Gesetzes- und Rechtschreibregeln in den kommenden Jahren entwickeln werden.

Wie wurde die Qualität gesichert?

Ich versichere Ihnen, dass die Befehls- und Skriptbeispiele auf mindestens zwei meiner Systeme liefen, bevor ich sie per Kopieren & Einfügen in das Manuskript zu diesem Buch übernommen und auf der Leser-Website zum Download veröffentlicht habe. Zudem haben einige Tausend Leser die bisherigen Auflagen verwendet, und Feedback dieser Leser habe ich in das Buch eingearbeitet.

Dennoch gibt es leider Gründe, warum die Beispiele bei Ihnen als Leser dieses Fachbuchs nicht laufen könnten:

- Eine abweichende Systemkonfiguration (in der heutigen komplexen Welt der vielen Varianten und Versionen von Betriebssystemen und Anwendungen nicht unwahrscheinlich). Es ist einem Fachbuchautor nicht möglich, alle Konfigurationen durchzutesten.
- Änderungen, die sich seit der Erstellung der Beispiele ergeben haben (mittlerweile gibt es sehr regelmäßig umfangreiche Breaking Changes in den Microsoft-Produkten, insbesondere beim Versionsnummernwechsel an der ersten Stelle, d. h. Windows PowerShell 5.1 und PowerShell 6.0 sowie PowerShell 6.2 und PowerShell 7.x).
- Schließlich sind auch menschliche Fehler des Autors möglich. Bitte bedenken Sie, dass das Fachbuchschreiben nur ein Hobby ist. Es gibt nur sehr wenige Menschen in Deutschland, die hauptberuflich als Fachbuchautor arbeiten und so professionell Programmcodebeispiele erstellen und testen können wie kommerziellen (bezahlten) Programmcode.

Wenn Beispiele bei Ihnen nicht laufen, kontaktieren Sie mich bitte mit einer sehr genauen Fehlerbeschreibung (Systemumgebung, Skriptcode, vollständiger Fehlertext usw.). Bitte verwenden Sie dazu das Kontaktformular auf www.powershell-doktor.de. Ich bemühe mich, Ihnen binnen zwei Wochen zu antworten. Im Einzelfall kann es wegen dienstlicher oder privater Abwesenheit aber auch länger dauern.

Wo kann man Verbesserungsvorschläge melden?

Nicht nur wenn Sie Fehler in den Befehls- und Skriptbeispielen finden, sondern auch wenn Sie allgemeine Verbesserungsvorschläge für die nächste Auflage haben, können Sie sich gerne bei mir melden. Vielleicht sind Ihnen noch Bugs in der PowerShell aufgefallen? Oder Sie haben noch eine funktionelle Anomalie der PowerShell bemerkt, die im Buch nicht erwähnt ist? Oder es gibt ein Feature, das erwähnt werden sollte?

Es kann sein, dass ich einige Punkte bewusst weggelassen habe. Es kann aber auch sein, dass ich diesen Bug, diese Anomalie bzw. dieses Feature selbst noch nicht bemerkt bzw. verwendet habe. Bitte bedenken Sie, dass kein Mensch jemals alle PowerShell-Befehle (einige Tausend) bzw. .NET-Programmierschnittstellen (einige Hunderttausend, wenn man alle Methoden und Eigenschaften einzeln zählt) in der Praxis benutzt hat oder bis zu seinem Lebensende benutzen wird.

Ich freue mich immer über konstruktives Feedback und Verbesserungsvorschläge. Bitte verwenden Sie dazu das Kontaktformular unter www.powershell-doktor.de/Leserfeedback.

Wann wird die nächste Auflage erscheinen?

Von meinen selbst verlegten Fachbüchern sind Sie es gewohnt, dass ich in kurzen Abständen von mehreren Wochen neue Versionen des Buchs veröffentliche.

Bitte beachten Sie, dass ständig neue Auflagen dieses Fachbuchs leider nicht möglich sind, da der Carl Hanser Verlag längere Produktionsprozesse hat und Bücher auf Vorrat für einen längeren Zeitraum druckt. Zwischen zwei Auflagen dieses Buchs lagen in der Vergangenheit daher immer ein bis zwei Jahre.

Wo kann man sich schulen oder beraten lassen?

Unter der E-Mail-Adresse Anfrage@IT-Visions.de stehen Ihnen mein Team und ich für Anfragen bezüglich Schulung, Beratung und Entwicklungstätigkeiten zur Verfügung – nicht nur zum Thema PowerShell und .NET/.NET Core, sondern zu fast allen modernen Techniken der Entwicklung und des Betriebs von Software in großen Unternehmen. Wir besuchen Sie gerne in Ihrem Unternehmen an einem beliebigen Standort oder unterstützen Sie per Videokonferenz.

Wem ist zu danken?

Folgenden Personen möchte ich meinen ausdrücklichen Dank für ihre Mitwirkung an diesem Buch aussprechen:

- meinem Kollegen Peter Monadjemi, der rund 100 Seiten mit Beispielen zu der 3. Auflage dieses Buchs beigetragen hat und dessen Inhalte zum Teil noch im Buch enthalten sind (Themen: Workflows, Bitlocker, ODBC, Hyper-V, DNS-Client, Firewall und Microsoft SQL Server-Administration),
- meinem Kollegen André Krämer, der die PowerShell 7 auf macOS getestet hat, da ich selbst kein macOS-Gerät besitze,
- Frau Sylvia Hasselbach, die mich schon seit 20 Jahren als Lektorin begleitet und die dieses Buchprojekt beim Carl Hanser Verlag koordiniert und vermarktet,
- Frau Sandra Gottmann, die meine Tippfehler gefunden und sprachliche Ungenauigkeiten eliminiert hat,
- den Lesern Alexander Grober und Mario Severing für ihre ausführlichen Hinweise auf von den Korrektoren früherer Auflagen nicht gefundene Tippfehler sowie inhaltliche Optimierungsmöglichkeiten in der Voraufgabe,
- meiner Frau und meinen Kindern dafür, dass sie mir das Umfeld geben, um neben meinem Hauptberuf an Büchern wie diesem zu arbeiten.

Zum Schluss dieses Vorworts . . .

... wünsche ich Ihnen viel Spaß und Erfolg mit der PowerShell!

Dr. Holger Schwichtenberg

Essen, im Sommer 2022

Über den Autor

- Studienabschluss Diplom-Wirtschaftsinformatik an der Universität Essen
- Promotion an der Universität Essen im Fachgebiet komponenten-basierter Softwareentwicklung
- Seit 1996 in der IT tätig als Softwareentwickler, Softwarearchitekt, Berater, Dozent und Fachjournalist
- Fachlicher Leiter des Expertenteams bei *www.IT-Visions.de* in Essen
- Chief Technology Expert (CTE) der Softwareentwicklung bei der MAXIMAGO GmbH in Dortmund (*www.MAXIMAGO.de*)
- Über 90 Fachbücher bei verschiedenen Verlagen, u. a. Carl Hanser Verlag, O'Reilly, APress, Microsoft Press, Addison Wesley sowie im Selbstverlag
- Mehr als 1400 Beiträge in Fachzeitschriften und Online-Portalen
- Gutachter in den Wettbewerbsverfahren der EU gegen Microsoft (2006–2009)
- Ständiger Mitarbeiter der Zeitschriften iX (seit 1999), dotnetpro (seit 2000) und Windows Developer (seit 2010) sowie beim Online-Portal heise.de (seit 2008)
- Regelmäßiger Sprecher auf nationalen und internationalen Fachkonferenzen (z. B. enterJS, BASTA!, Microsoft TechEd, Microsoft Summit, Microsoft IT Forum, OOP, IT Tage, .NET Architecture Camp, Advanced Developers Conference, Developer Week, DOTNET Cologne, MD DevDays, Community in Motion, DOTNET-Konferenz, VS One, NRW.Conf, Net.Object Days, Windows Forum, Container Conf)
- Zertifikate und Auszeichnungen von Microsoft:
 - Microsoft Most Valuable Professional (MVP), kontinuierlich ausgezeichnet seit 2004
 - Microsoft Certified Solution Developer (MCSD)
- Thematische Schwerpunkte:
 - Softwarearchitektur, mehrschichtige Softwareentwicklung, Softwarekomponenten
 - Visual Studio, Continuous Integration (CI) und Continuous Delivery (CD) mit Azure DevOps



www.IT-Visions.de[®]
Dr. Holger Schwichtenberg



MAXIMAGO

- Microsoft .NET (.NET Framework, .NET Core), C#, Visual Basic
- .NET-Architektur, Auswahl von .NET-Techniken
- Einführung von .NET, Migration auf .NET
- Webanwendungsentwicklung und Cross-Plattform-Anwendungen mit HTML/CSS, JavaScript/TypeScript und C# sowie Webframeworks wie Angular, Vue.js ASP.NET (Core) und Blazor
- Verteilte Systeme/Webservices mit .NET, insbesondere WebAPI, gRPC und WCF
- Relationale Datenbanken, XML, Datenzugriffsstrategien
- Objektrelationales Mapping (ORM), insbesondere ADO.NET Entity Framework und Entity Framework Core
- PowerShell
- Architektur- und Code-Reviews
- Performance-Analysen und -Optimierung
- Entwicklungsrichtlinien
- Ehrenamtliche Community-Tätigkeiten:
 - Vortragender für die International .NET Association (INETA) und .NET Foundation
 - Betrieb diverser Community-Websites:
www.dotnet-lexikon.de, *www.dotnetframework.de*, *www.windows-scripting.de*,
www.aspnetdev.de u. a.
- Firmenwebsites: *www.IT-Visions.de* und *www.MAXIMAGO.de*
- Weblog: *www.dotnet-doktor.de*

**HINWEIS:**

- Kontakt für Anfragen zu Schulung und Beratung:
kundenteam@IT-Visions.de, Telefon 0201/64 95 90 – 50
- Kontakt für Anfragen zu Softwareentwicklungsprojekten:
hsc@MAXIMAGO.de
Telefon 0231/58 69 67 – 12
- Kontakt für Feedback zu diesem Buch:
www.dotnet-doktor.de/Leserfeedback

A

Teil A: PowerShell-Basiswissen

Dieser Buchteil informiert über die Basiskonzepte der PowerShell, insbesondere Commandlets, Pipelines, Navigation und Skripte. Außerdem werden am Ende dieses Teils Werkzeuge vorgestellt.

1

Fakten zur PowerShell

Mit der Windows PowerShell (WPS) besitzt Microsoft seit dem Jahr 2006 eine Kommandozeile, die es mit den Unix-Shells aufnehmen kann und diese in Hinblick auf Eleganz und Robustheit in einigen Punkten auch überbieten kann. Die PowerShell ist eine Adaption des Konzepts von Unix-Shells auf Windows unter Verwendung des .NET Frameworks und mit Anbindung an die Windows Management Instrumentation (WMI).

Dieses Kapitel liefert Ihnen einige wichtige Fakten, die Sie zum Verständnis der PowerShell kennen sollten.

■ 1.1 Was ist die PowerShell?

Die **PowerShell (PS)** ist eine .NET-basierte Umgebung für interaktive Systemadministration und Scripting. Während die Windows PowerShell nur auf Windows läuft, gibt es die PowerShell seit Version 6.0 auch für Linux und macOS.

Die Kernfunktionen der PowerShell sind:

- Zahlreiche eingebaute Befehle, die „Commandlets“ (oft abgekürzt „Cmdlets“) genannt werden
- Zugang zu allen Systemobjekten, die durch COM-Bibliotheken, das .NET Framework und die Windows Management Instrumentation (WMI) bereitgestellt werden
- Robuster Datenaustausch zwischen Commandlets durch Pipelines basierend auf typisierten Objekten
- Ein einheitliches Navigationsparadigma für verschiedene Speicher (z.B. Dateisystem, Registrierungsdatenbank, Zertifikatsspeicher, Active Directory und Umgebungsvariablen)
- Eine einfach zu erlernende, aber mächtige Skriptsprache mit wahlweise schwacher oder starker Typisierung
- Ein Sicherheitsmodell, das die Ausführung unerwünschter Skripte unterbindet
- Integrierte Funktionen für Ablaufverfolgung und Debugging
- Die PowerShell kann um eigene Befehle erweitert werden.
- Die PowerShell kann in eigene Anwendungen integriert werden (PowerShell Hosting).

■ 1.2 Geschichte der PowerShell

Vorgänger der PowerShell sind die DOS-Eingabeaufforderung und das Active Scripting mit Windows Script Host (WSH) und Visual Basic Script.

Das DOS-ähnliche Kommandozeilenfenster hat viele Windows-Versionen in beinahe unveränderter Form überlebt. Es bot textbasierte Kommandozeilenbefehle nur für einige, aber nicht alle administrativen Aufgaben.

Das Active Scripting war einigen Administratoren zu komplex, weil es viel Wissen über objektorientiertes Programmieren und das Component Object Model (COM) voraussetzt. Die vielen Ausnahmen und Ungereimtheiten im Active Scripting erschweren das Erlernen von Windows Script Host (WSH) und der zugehörigen Komponentenbibliotheken.

Schon im Zuge der Entwicklung des Windows Server 2003 gab Microsoft zu, dass man Unix-Administratoren zum Interview über ihr tägliches Handwerkzeug gebeten hatte. Das kurzfristige Ergebnis war eine große Menge zusätzlicher Kommandozeilenwerkzeuge. Langfristig setzt Microsoft jedoch auf eine Ablösung des DOS-ähnlichen Konsolenfensters durch eine neue Scripting-Umgebung.

Mit dem Erscheinen des .NET Frameworks im Jahre 2002 wurde lange über einen WSH.NET spekuliert. Microsoft stellte jedoch die Neuentwicklung des WSH für das .NET Framework ein, als abzusehen war, dass die Verwendung von .NET-basierten Programmiersprachen wie C# und Visual Basic .NET dem Administrator nur noch mehr Kenntnisse über objektorientierte Softwareentwicklung abverlangen würde.

Microsoft beobachtete in der Unix-Welt eine hohe Zufriedenheit mit den dortigen Kommandozeilen-Shells und entschloss sich daher, das Konzept der Unix-Shells, insbesondere das Pipelining, mit dem .NET Framework zusammenzubringen und daraus eine .NET-basierte Windows Shell zu entwickeln. Diese sollte noch einfacher als eine Unix-Shell, aber dennoch so mächtig wie das .NET Framework sein.

In einer ersten Beta-Version wurde die neue Shell schon unter dem Codenamen „Monad“ auf der Professional Developer Conference (PDC) im Oktober 2003 in Los Angeles vorgestellt. Nach den Zwischenstufen „Microsoft Shell (MSH)“ und „Microsoft Command Shell“ trägt die neue Skriptumgebung seit Mai 2006 den Namen „Windows PowerShell (WPS)“.

Die PowerShell 1.0 erschien am 6. 11. 2006 zeitgleich mit Windows Vista, war aber dort nicht enthalten, sondern musste heruntergeladen und nachinstalliert werden.

Eine Entwicklergruppe außerhalb von Microsoft hatte 2008 mit einer Linux-Implementierung der PowerShell unter dem Namen „PASH“ begonnen (siehe <https://github.com/Pash-Project/Pash>). Dieses Projekt hatte aber in der Praxis nie eine große Bedeutung.

Die PowerShell 2.0 ist zusammen mit Windows 7/Windows Server 2008 R2 am 22. 7. 2009 erschienen.

Die PowerShell 3.0 ist zusammen mit Windows 8/Windows Server 2012 am 15. 8. 2012 erschienen.

Die PowerShell 4.0 ist zusammen mit Windows 8.1/Windows Server 2012 R2 am 9. 9. 2013 erschienen.

Die PowerShell 5.0 ist als Teil von Windows 10 erschienen am 29. 7. 2015. Abweichend von den bisherigen Gepflogenheiten ist die PowerShell 5.0 als Erweiterung für Windows Server

2008 R2 (mit Service Pack 1) und Windows Server 2012/2012 R2 erst deutlich später, am 16.12.2015 erschienen. Für Windows 7 und Windows 8.1 sollte es erst gar keine Version mehr geben. Doch am 18.12.2015 hatte Microsoft ein Einsehen mit den Kunden und lieferte die PowerShell 5.0 auch für diese Betriebssysteme nach. Kurioserweise musste Microsoft den Download dann am 23.12.2015 wegen eines gravierenden Fehlers für einige Wochen vom Netz nehmen. Microsoft hatte das Produkt im neuen Agilitäts-Wahn nicht richtig getestet.

Der Windows Server 2016 (erschieden am 26.9.2016) enthält PowerShell 5.1 und Windows 10 und wurde mit dem Windows 10 Anniversary Update (Version 1607, Codename „Redstone 1“) am 2.8.2016 aktualisiert. PowerShell 5.1 ist erst seit 19.1.2017 als Add-on für Windows 7, Windows 8.1, Windows Server 2008 R2, Windows 2012 und Windows 2012 R2 verfügbar.

Eine reduzierte „Core“-Version der Windows PowerShell ist als „Windows PowerShell Core 5.1“ enthalten in Windows Nano Server, im ersten Release 2016 als Standardpaket, im zweiten in Release „1709“ als Option.

Microsoft hat sich seit dem Jahr 2015 für andere Betriebssysteme und die Entwicklung von „Open Source Software“ (OSS) geöffnet. Am 18. August 2016 hat Microsoft angekündigt, die PowerShell nun plattformunabhängig zu entwickeln. Die PowerShell ist seitdem auch ein Open-Source-Projekt (MIT-Lizenz), das Microsoft auf Github.com vor den Augen der Öffentlichkeit entwickelt [github.com/PowerShell/PowerShell]. Die erste Version der „PowerShell Core“ (ohne Windows im Namen!) ist mit der Versionsnummer 6.0 am 20.01.2018 erschienen. Danach folgten die Versionen 6.1 und 6.2.



HINWEIS: Die Version 6.x der PowerShell trug den Namen „PowerShell Core 6.x“. Microsoft drückte damit aus, dass diese Version auf .NET Core, dem plattformunabhängigen .NET Framework, basiert, und zugleich auch, dass PowerShell Core genau wie .NET ein Neustart der Entwicklung ist, die nicht den Anspruch hat, kompatibel zum Vorgänger zu sein. Bei .NET hat sich Microsoft daher auch dazu entschlossen, die Versionszählung wieder bei 1.0 zu beginnen. Bei PowerShell Core setzt Microsoft die Versionszählung der Windows PowerShell fort.

Seit Version 7.0 trägt nur noch den Namen „PowerShell“ und versteht sich als Zusammenführung von Windows PowerShell und PowerShell Core [<https://github.com/PowerShell/PowerShell/pull/9513>]. Dennoch sind nicht alle Funktionen aus der Windows PowerShell in der PowerShell 7.x enthalten.



HINWEIS: Während es für die Windows PowerShell 5.1 nur noch Fehlerbehebungen und Sicherheitsupdates geben wird, arbeitet Microsoft intensiv an der Weiterentwicklung der plattformneutralen PowerShell 7.x. Die PowerShell Core 6.x wird von Microsoft nicht mehr gepflegt.

■ 1.3 Welche Varianten und Versionen der PowerShell gibt es?

Es gibt mittlerweile vier Varianten der PowerShell:

- die **Windows PowerShell** (in den Versionen 1.0 bis 5.1) sowie hierzu passend den Editor Windows PowerShell Integrated Scripting Environment (ISE)
- die **Windows PowerShell Core** (nur Version 5.1) für Windows Nano Server
- die **PowerShell Core** (in den Versionen 6.0 bis 6.2)
- die **PowerShell** (seit Version 7.0)



HINWEIS: Der Begriff „PowerShell“ wird von Microsoft, in der Nutzergemeinde und auch in diesem Buch nicht nur für die Versionen ab 7.0, sondern als Oberbegriff für alle vier Varianten verwendet.

Microsoft fasst die Windows PowerShell und die Windows PowerShell ISE als „Desktop Edition“ zusammen und die Windows PowerShell Core 5.1, die PowerShell Core 6.x sowie die PowerShell 7.x als „Core Edition“.

■ 1.4 Windows PowerShell versus PowerShell Core versus PowerShell 7.x

Die Windows PowerShell 5.1 war weit mächtiger als die PowerShell Core 6.x, weil die PowerShell Core einen Neustart des PowerShell-Entwicklungsprojekts im Hinblick auf Plattformunabhängigkeit darstellt. In PowerShell Core fehlten viele Commandlets und Funktionen der Grundausstattung der Windows PowerShell, und viele der verfügbaren PowerShell-Erweiterungsmodule liefen nicht in der PowerShell Core.

Die PowerShell 7.x versteht sich als gemeinsamer Nachfolger von Windows PowerShell 5.1 und PowerShell Core 6.2. Das PowerShell-Entwicklungsteam ist damit schon Vorreiter bei der Zusammenführung von klassischer Produktlinie und Core-Produktlinie. Analog hat das .NET-Entwicklungsteam .NET Framework und .NET Core zu .NET vereint.

Mit PowerShell 7.0 hat sich der Funktionsabstand zur Windows PowerShell deutlich reduziert, weil das moderne .NET, die Basis von PowerShell 6.x und 7.x, funktional stark aufgeholt hat. Aber weiterhin gilt, dass einige Commandlets und Funktionen der Windows PowerShell sowie einige Erweiterungsmodule nicht verfügbar sind. Und es gilt insbesondere: PowerShell 7.x auf Linux und macOS kann weit weniger als PowerShell 7.x auf Windows.

Details zu den Funktionseinschränkungen der PowerShell Core lesen Sie im Kapitel „PowerShell 7 für Windows, Linux und macOS“.

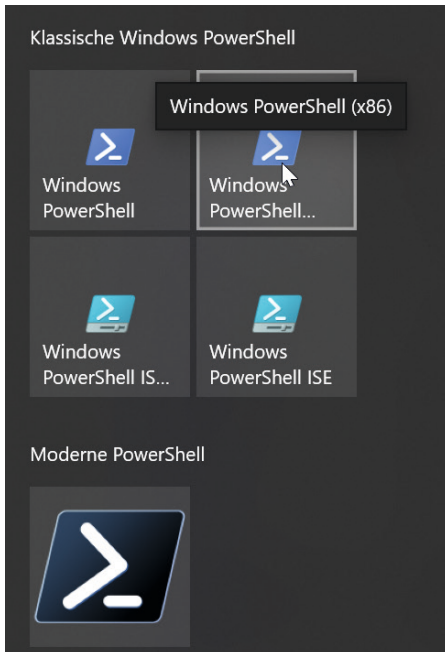


HINWEIS: Für die meisten Windows-Administratoren wird die Windows PowerShell weiterhin die erste Wahl bleiben, denn diese Variante der PowerShell ist in Windows bereits vorhanden. Es gibt nur wenige neue Befehle in PowerShell 7.x gegenüber Windows PowerShell 5.1, während andererseits in Version 7.x Befehle aus der Windows PowerShell 5.1 fehlen. Der Anreiz einer Zusatzinstallation ist also nicht sehr groß. In einigen Unternehmen sind solche Zusatzinstallationen auf Servern auch gar nicht möglich.

Unter Linux und macOS gibt es keine Windows PowerShell. Hier können Sie die PowerShell 7.x verwenden. Der Wert der PowerShell Core unter Linux und macOS liegt in den mächtigen Pipelining- sowie Ein- und Ausgabe-Commandlets. Für konkrete Zugriffe auf das Betriebssystem gibt es hingegen für die PowerShell Core unter macOS und Linux noch fast keine Commandlets. Man wird also hier immer klassische Linux- und macOS-Kommandozeilenbefehle mit zeichenkettenbasierter Verarbeitung in die PowerShell einbinden. Wie dies geht, wird im Kapitel „PowerShell 7 für Windows, Linux und macOS“ erklärt.

Tabelle 1.1 Vergleich der PowerShell-Varianten

	Windows PowerShell	Windows PowerShell Core	PowerShell Core	PowerShell
Versionen	1.0 bis 5.1	5.1	6.0 bis 6.2	Ab 7.0
Edition	Desktop	Core	Core	Core
Basis	.NET Framework	.NET Core	.NET Core	.NET Core (7.0) bzw. .NET (seit 7.1)
Unterstützte Betriebssysteme	Windows Client ab 7, Windows Server ab 2012	Windows Nano Server	Windows, Linux, macOS	Windows, Linux, macOS
Wird weiterentwickelt	Nein	Nein	Nein	Ja
Start ohne Setup	In aktuellen Windows-Versionen: ja, sonst nein	Ja in Nano Server	Ja, möglich (Self Contained App in Archivdatei)	Ja, möglich (Self Contained App in Archivdatei)

**Bild 1.1**

Logos im Vergleich: Windows PowerShell 5.1, Windows PowerShell ISE, sowie PowerShell 7.x

■ 1.5 Motivation zur PowerShell

Falls Sie eine Motivation brauchen, sich mit der PowerShell zu beschäftigen, wird dieses Kapitel sie Ihnen liefern. Es stellt die Lösung für eine typische Scripting-Aufgabe sowohl im „alten“ Windows Script Host (WSH) als auch in der „neuen“ PowerShell vor.

Zur Motivation, sich mit der PowerShell zu beschäftigen, soll folgendes Beispiel aus der Praxis dienen. Es soll ein Inventarisierungsskript für Software erstellt werden, das die installierten MSI-Pakete mit Hilfe der Windows Management Instrumentation (WMI) von mehreren Computern ausliest und die Ergebnisse in einer CSV-Datei (*softwareinventar.csv*) zusammenfasst. Die Namen (oder IP-Adressen) der abzufragenden Computer sollen in einer Textdatei (*computernamen.txt*) stehen.

Die Lösung mit dem WSH benötigt 90 Codezeilen (inklusive Kommentare und Parametrisierungen). In der PowerShell lässt sich das Gleiche in nur 13 Zeilen ausdrücken. Wenn man auf die Kommentare und die Parametrisierung verzichtet, dann reicht sogar genau eine Zeile. Das PowerShell-Skript läuft in der Windows PowerShell und auch in der PowerShell 6/7 unter Windows, aber nicht unter Linux und macOS, da es dort noch keine Implementierung des für den Zugriff auf die installierte Software notwendigen Web Based Enterprise Management (WBEM) und des Common Information Model (CIM) für die PowerShell gibt.

Listing 1.1 Softwareinventarisierung – Lösung 1 mit dem WSH

[3_Einsatzgebiete/Software/Software_Inventory.vbs]

```

' -----
' Skriptname: Software_inventar.vbs
' Autor: Dr. Holger Schwichtenberg
' -----
' Dieses Skript erstellt eine Liste
' der installierten Software
' -----
Option Explicit

' --- Einstellungen
Const Trennzeichen = ";" ' Trennzeichen für Spalten in der Ausgabedatei
Const Eingabedateiname = "computernamen.txt"
Const Ausgabedateiname = "softwareinventar.csv"
Const Bedingung = "SELECT * FROM Win32_Product where not Vendor like '%Microsoft%'"

Dim objFSO ' Dateisystem-Objekt
Dim objTX ' Textdatei-Objekt für die Liste der zu durchsuchenden computer
Dim i ' Zähler für Computer
Dim computer ' Name des aktuellen computers
Dim Eingabedatei ' Name und Pfad der Eingabedatei
Dim Ausgabedatei ' Name und Pfad der Ausgabedatei

' --- Startmeldung
WScript.Echo "Softwareinventar.vbs"
WScript.Echo "(C) Dr. Holger Schwichtenberg, http://www.Windows-Scripting.de"

' --- Global benötigtes Objekt
Set objFSO = CreateObject("Scripting.FileSystemObject")

' --- Ermittlung der Pfade
Eingabedatei = GetCurrentPfad & "\" & Eingabedateiname
Ausgabedatei = GetCurrentPfad & "\" & Ausgabedateiname

' --- Auslesen der computerliste
Set objTX = objFSO.OpenTextFile(Eingabedatei)

' --- Meldungen
WScript.Echo "Eingabedatei: " & Eingabedatei
WScript.Echo "Ausgabedatei: " & Ausgabedatei

' --- Überschriften einfügen
Ausgabe _
"computer" & Trennzeichen & _
"Name" & Trennzeichen & _
"Beschreibung" & Trennzeichen & _
"Identifikationsnummer" & Trennzeichen & _
"Installationsdatum" & Trennzeichen & _
"Installationsverzeichnis" & Trennzeichen & _
"Zustand der Installation" & Trennzeichen & _
"Paketzwischenspeicher" & Trennzeichen & _
"SKU Nummer" & Trennzeichen & _
"Hersteller" & Trennzeichen & _
"Version"

' --- Schleife über alle Computer
Do While Not objTX.AtEndOfStream
    computer = objTX.ReadLine

```

```

    i = i + 1
    WScript.Echo "=== Computer #" & i & ": " & computer

GetInventar computer

Loop

' --- Eingabedatei schließen
objTX.Close
' --- Abschlußmeldung
WScript.echo "Softwareinventarisierung beendet!"

' === Softwareliste für einen computer erstellen
Sub GetInventar(computer)

Dim objProduktMenge
Dim objProdukt
Dim objWMIDienst

' --- Zugriff auf WMI
Set objWMIDienst = GetObject("winmgmts:" &
    "{impersonationLevel=impersonate}!\" & computer &
    "\root\cimv2")
' --- Liste anfordern
Set objProduktMenge = objWMIDienst.ExecQuery _
    (Bedingung)
' --- Liste ausgeben
WScript.Echo "Auf " & computer & " sind " &
objProduktMenge.Count & " Produkte installiert."
For Each objProdukt In objProduktMenge
    Ausgabe _
        computer & Trennzeichen &
        objProdukt.Name & Trennzeichen &
        objProdukt.Description & Trennzeichen &
        objProdukt.IdentifyingNumber & Trennzeichen &
        objProdukt.InstallDate & Trennzeichen &
        objProdukt.InstallLocation & Trennzeichen &
        objProdukt.InstallState & Trennzeichen &
        objProdukt.PackageCache & Trennzeichen &
        objProdukt.SKUNumber & Trennzeichen &
        objProdukt.Vendor & Trennzeichen &
        objProdukt.Version
WScript.Echo    objProdukt.Name
Next
End Sub

' === Ausgabe
Sub Ausgabe(s)
Dim objTextFile
' Ausgabedatei öffnen
Set objTextFile = objFSO.OpenTextFile(Ausgabedatei, 8, True)
objTextFile.WriteLine s
objTextFile.Close
'WScript.Echo s
End Sub

' === Pfad ermitteln. in dem das Skript liegt
Function GetCurrentPfad
GetCurrentPfad = objFSO.GetFile (WScript.ScriptFullName).ParentFolder
End Function

```

Listing 1.2 Softwareinventarisierung – Lösung 2 als PowerShell-Skript

```
[3_Einsatzgebiete/Software/SoftwareInventory_WMI_Script.ps1]

# Einstellungen
$InputFileName = "computernamen.txt"
$OutputFileName = "softwareinventar.csv"
$query = "SELECT * FROM Win32_Product where not Vendor like '%Microsoft%'"

# Eingabedatei auslesen
$Computers = Get-Content $InputFileName

# Schleife über alle Computer
$Software = $Computers | foreach { Get-CimInstance -query $query -computername $_ }
# Ausgabe in CSV
$Software | select Name, Description, IdentifyingNumber, InstallDate,
InstallLocation, InstallState, SKUNumber, Vendor, Version | export-csv
$OutputFileName -notypeinformation
```

Listing 1.3 Softwareinventarisierung – Lösung 3 als PowerShell-Pipeline-Befehl

```
[3_Einsatzgebiete/Software/SoftwareInventory_WMI_Pipeline.ps1]

Get-Content "computers.txt" | foreach {Get-CimInstance -computername $_ -query
"SELECT * FROM Win32_Product where not Vendor like '%Microsoft%'" } | export-csv
"Softwareinventory.csv" -notypeinformation
```

■ 1.6 Betriebssysteme mit vorinstallierter PowerShell

Die folgende Tabelle zeigt, in welchen Betriebssystemen welche Version der PowerShell mitgeliefert wird bzw. wo sie nachträglich installierbar ist.

Tabelle 1.2 Verfügbarkeit der PowerShell auf verschiedenen Betriebssystemen

Betriebssystem	Mitgelieferte PowerShell-	Nachträglich installierbare PowerShell
Windows 2000, Windows 9x, Windows ME, Windows NT 4.0	PowerShell nicht enthalten	Nachträgliche Installation nicht von Microsoft unterstützt
Windows XP	PowerShell nicht enthalten	PowerShell 1.0 und PowerShell 2.0
Windows Server 2003	PowerShell nicht enthalten	PowerShell 1.0 und PowerShell 2.0
Windows Server 2003 R2	PowerShell nicht enthalten	PowerShell 1.0 und PowerShell 2.0
Windows Vista	PowerShell nicht enthalten	PowerShell 1.0 und PowerShell 2.0

(Fortsetzung nächste Seite)

Tabelle 1.2 Verfügbarkeit der PowerShell auf verschiedenen Betriebssystemen (*Fortsetzung*)

Betriebssystem	Mitgelieferte PowerShell-	Nachträglich installierbare PowerShell
Windows Server 2008	PowerShell 1.0 enthalten als optionales Feature	PowerShell 2.0, PowerShell 3.0
Windows Server 2008 R2	PowerShell 1.0 enthalten	PowerShell 2.0, PowerShell 3.0
Windows 7	PowerShell 2.0 enthalten	PowerShell 3.0, PowerShell 4.0, PowerShell 5.0, PowerShell 5.1, PowerShell Core 6.x, PowerShell 7
Windows Server 2008 R2	PowerShell 2.0 enthalten	PowerShell 3.0, PowerShell 4.0, PowerShell 5.0, PowerShell 5.1, PowerShell Core 6.x, PowerShell 7
Windows Server 2008 Core	PowerShell nicht enthalten	PowerShell 3.0, PowerShell Core 6.x, PowerShell 7
Windows Server 2008 R2 Core	PowerShell 2.0 enthalten als optionales Feature	PowerShell Core 6.x, PowerShell 7
Windows 8.0	PowerShell 3.0 enthalten	Achtung: PowerShell 4.0 und 5.0/5.1 können nur durch ein (vor- heriges) Update auf Windows 8.1 nachgerüstet werden.
Windows Server 2012 inkl. Core	PowerShell 3.0 enthalten	PowerShell 4.0, PowerShell 5.0, PowerShell 5.1, PowerShell Core 6.x, PowerShell 7
Windows 8.1	PowerShell 4.0 enthalten	PowerShell 5.0, PowerShell 5.1, PowerShell Core 6.x, PowerShell 7
Windows Server 2012 R2 inkl. Core	PowerShell 4.0 enthalten	PowerShell 5.0, PowerShell 5.1, PowerShell Core 6.x, PowerShell 7
Windows 10	PowerShell 5.0 enthalten	PowerShell Core 6.x, PowerShell 7
Windows 10 seit Creators Update (Redstone 2, Version 1703, April 2017)	PowerShell 5.1 enthalten	PowerShell Core 6.x, PowerShell 7
Windows 11	PowerShell 5.1 enthalten	PowerShell Core 6.x, PowerShell 7
Windows Server 2016	PowerShell 5.1 enthalten	PowerShell Core 6.x, PowerShell 7
Windows Server 1709	PowerShell 5.1 enthalten	PowerShell Core 6.x, PowerShell 7
Windows Nano Server 2016	Reduzierte PowerShell Core 5.1 enthalten	PowerShell Core 6.x, PowerShell 7
Windows Nano Server seit Version 1709	PowerShell Core wurde aus dem Standardinstallationsumfang ent- fernt, vgl. docs.microsoft.com/de- de/windows-server/get-started/ nano-in-semi-annual-channel	PowerShell Core 5.1, PowerShell Core 6.x, PowerShell 7
Windows Server 2019	PowerShell 5.1 enthalten	PowerShell Core 6.x, PowerShell 7
Windows Server 2022	PowerShell 5.1 enthalten	PowerShell Core 6.x, PowerShell 7

Betriebssystem	Mitgelieferte PowerShell-	Nachträglich installierbare PowerShell
OpenSUSE-Linux ab Version 42.3	-	PowerShell Core 6.x, PowerShell 7
Ubuntu-Linux ab Version 16.04	-	PowerShell Core 6.x, PowerShell 7
macOS/X ab Version 10.12	-	PowerShell Core 6.x, PowerShell 7
Debian ab Version 9		PowerShell Core 6.x, PowerShell 7
Red Hat Enterprise Linux (RHEL) ab Version 7		PowerShell Core 6.x, PowerShell 7
Fedora ab Version 30		PowerShell Core 6.x, PowerShell 7
CentOS ab Version 7		PowerShell Core 6.x, PowerShell 7
Alpine ab Version 3.9		PowerShell 7
Raspberry Pi OS (Raspbian) ab Version 2		PowerShell 7

Es gibt außerdem noch PowerShell 7.x-Unterstützung für andere Linux-Derivate (z. B. Kali, Arch) durch Projekte außerhalb von Microsoft, vgl. <https://docs.microsoft.com/en-us/powershell/scripting/install/community-support>

■ 1.7 Support der PowerShell

Ein Produkt, das von Microsoft offiziell als „unterstützt“ gilt, bekommt Updates für Fehler und Sicherheitslücken und die Kunden können den technischen Support von Microsoft bei Problemen kontaktieren.

Die Windows PowerShell wird solange unterstützt wie das Betriebssystem, mit dem die PowerShell ausgeliefert wurde.





Für die moderne PowerShell gilt die „Microsoft Modern Lifecycle Policy“ [<https://docs.microsoft.com/en-us/lifecycle/policies/modern>]. Diese Richtlinie unterscheidet zwischen „Long-Term-Support“ (LTS) und „Current-Version“, neuerdings auch Short Term Support (STS) genannt.

Microsoft hat die Wartung und den Support der PowerShell Core Versionen 6.x bereits beendet.

Die Unterstützung der Version 7.x können Sie hier nachlesen: <https://docs.microsoft.com/en-us/powershell/scripting/install/powershell-support-lifecycle>

Windows

The following table is a list of PowerShell releases and the versions of Windows they are supported on. These versions are supported until either the version of PowerShell reaches end-of-support or the version of Windows reaches end-of-support.

- A  indicates that the version of the OS or PowerShell is still supported
- A  indicates that the version of the OS or PowerShell isn't supported
- A  indicates the version of PowerShell is no longer supported on that version of the OS
- When both the version of the OS and the version of PowerShell have , that combination is supported


















































Windows	7.0 (LTS)	7.1	7.2 (LTS-current)	7.3 (preview)
 Windows Server 2016, 2019, or 2022				
 Windows Server 2012 R2				
 Windows Server Core (2012 R2 or higher)				
 Windows Server Nano (1809 or higher)				
 Windows Server 2012				
 Windows Server 2008 R2				
 Windows 11				
 Windows 10 1607+				
 Windows 8.1				

Bild 1.2 Unterstützung der modernen PowerShell in Windows, Stand 15. 3. 2022

macOS

The following table is a list of currently supported PowerShell releases and the versions of Windows they are supported on. These versions remain supported until either the version of PowerShell reaches end-of-support or the version of [macOS reaches end-of-support][eol-windows].

- A  indicates that the version of the OS or PowerShell is still supported
- A  indicates that the version of the OS or PowerShell isn't supported
- A  indicates the version of PowerShell is no longer supported on that version of the OS
- When both the version of the OS and the version of PowerShell have , that combination is supported

























macOS	7.0 (LTS)	7.1	7.2 (LTS-current)	7.3 (preview)
 macOS Big Sur 11.5				
 macOS Catalina 10.15				
 macOS Mojave 10.14				
 macOS High Sierra 10.13				

Bild 1.3 Unterstützung der modernen PowerShell in macOS, Stand 15. 3. 2022

Ubuntu Linux

The following table is a list of currently supported PowerShell releases and the versions of Ubuntu they are supported on. These versions remain supported until either the version of PowerShell reaches end-of-support or the version of Ubuntu reaches end-of-support.

- A  indicates that the version of the OS or PowerShell is still supported
- A  indicates that the version of the OS or PowerShell isn't supported
- A  indicates the version of PowerShell is no longer supported on that version of the OS
- When both the version of the OS and the version of PowerShell have , that combination is supported
















Ubuntu	7.0 (LTS)	7.1	7.2 (LTS-current)	7.3 (preview)
 20.04 (LTS)				
 18.04 (LTS)				
 16.04 (LTS)				

Bild 1.4 Unterstützung der modernen PowerShell in Ubuntu, Stand 15.3.2022

Weitere Linux-Distributionen sind hier nicht wiedergegeben, um Platz zu sparen. Sie finden den aktuellen Status unter <https://docs.microsoft.com/en-us/powershell/scripting/install/powershell-support-lifecycle>.



TIPP: Man muss immer die aktuellste Patch-Version der PowerShell installiert haben, um Support zu bekommen!

1.8 Einflussfaktoren auf die Entwicklung der PowerShell

Die PowerShell ist eine Symbiose aus:

- dem klassischen Windows-Kommandozeilenfenster,
- den bekannten Skript- und Shell-Sprachen wie Perl, Ruby, ksh und bash,
- dem .NET Framework und
- der Windows Management Instrumentation (WMI).

Die PowerShell ist implementiert auf dem .NET Framework. Sie ist jedoch kein .NET Runtime Host mit der Möglichkeit, Befehle der Common Intermediate Language (CIL) auf der Common Language Runtime (CLR) auszuführen.

Die PowerShell verwendet ein völlig anderes Host-Konzept mit Commandlets, Objekt-Pipelines und einer neuen Sprache, die von Microsoft als PowerShell Language (PSL) bezeichnet wird. Sie ist Perl, Ruby, C# und einigen Unix-Shell-Sprachen sehr ähnlich, aber mit keiner Unix-Shell kompatibel. Nutzer der WMI Command Shell (*wmic.exe*), die mit Windows XP eingeführt wurde, werden sich in der PowerShell schnell zurechtfinden.

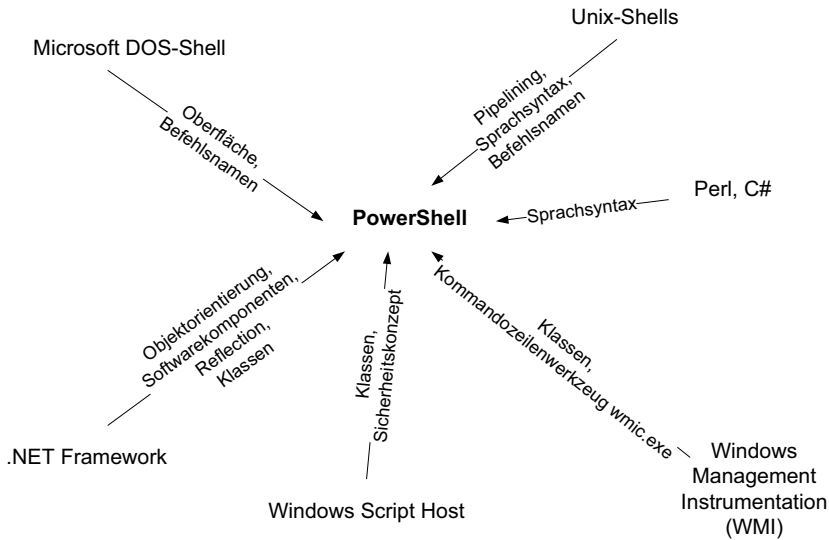


Bild 1.5 Einflussfaktoren auf die Architektur und die Umsetzung der PowerShell



ACHTUNG: Die PowerShell ist angetreten, vom Administrator weniger Kenntnisse in Objektorientierung und über Softwarekomponenten zu verlangen, als dies der Vorgänger Windows Script Host (WSH) tat. Tatsächlich kann man in der PowerShell viel erreichen, ohne sich mit dem zu Grunde liegenden .NET Framework zu beschäftigen. Dennoch: Wer alle Möglichkeiten der PowerShell nutzen will, braucht dann aber doch etwas Verständnis für objektorientiertes Programmieren und Erfahrung mit dem .NET Framework.

Wenn Sie sich hier noch nicht auskennen, lesen Sie bitte zuerst in diesem Buch Anhang A Crashkurs „Objektorientierung“ und Anhang B Crashkurs „.NET“.

1.9 Anbindung an Klassenbibliotheken

Die Version 1.0 der PowerShell enthielt sehr viele Commandlets für die Pipelining-Infrastruktur, aber nur sehr wenige Befehle, die tatsächlich Bausteine des Betriebssystems in die Pipeline werfen. Prozesse, Systemdienste, Dateien, Zertifikate und Registrierungsdatenbankeinträge sind die magere Ausbeute beim ersten Blick in die Commandlet-Liste. Drei Commandlets eröffnen der PowerShell aber neue Dimensionen: `New-Object` (für .NET- und COM-Objekte) und `Get-WmiObject` bzw. `Get-CimInstance` (für WMI-Objekte). Seit Version 2.0 gibt es – zumindest in Verbindung mit neueren Betriebssystemen – mehr PowerShell-Befehle, die tatsächlich auf das Betriebssystem zugreifen.



HINWEIS: Die Option, nicht nur alle WMI-Klassen, sondern auch alle .NET-Klassen direkt benutzen zu können, ist Segen und Fluch zugleich. Ein Segen, weil dem Skriptentwickler damit mehr Möglichkeiten als jemals zuvor zur Verfügung stehen. Ein Fluch, weil nur der Skriptentwickler die PowerShell-Entwicklung richtig beherrschen kann, der auch das .NET Framework kennt. Um die Ausmaße von .NET zu beschreiben, sei die Menge der Klassen genannt. In .NET 2.0 waren es 6358, in .NET 3.5 sind es 10758, in .NET 4.7 sind es 13526.

1.10 PowerShell versus WSH

Administratoren fragen sich oft, wie sich die PowerShell im Vergleich zum Windows Script Host (WSH) positioniert, womit man neue Scripting-Projekte beginnen sollte und ob der WSH bald aus Windows verschwinden wird. Die folgende Tabelle trägt Fakten zusammen und bewertet auch die beiden Scripting-Plattformen.

Tabelle 1.3 Vergleich WSH und PowerShell

	Windows Script Host (WSH)	Windows PowerShell (WPS)	Windows PowerShell Core	PowerShell Core 6.x/PowerShell 7
Erstmals erschienen	1998	2006	2017	2018
Nummer der ersten Version	1.0	1.0	5.1	6.0
Aktueller Versionsstand	5.8	5.1	Core 5.1	7.2 (7.3 Preview)
Läuft auf Windows-Betriebssystemen	Alle Windows-Betriebssysteme ab Windows 95/NT 4.0	Version 1.0 ab Windows XP, Version 5.1 ab Windows 7 und Windows Server 2008 R2	Windows Nano Server 2016 und Windows Nano Server 1709	Windows ab Version 7, Windows Server ab Version 2008 R2
Läuft auf anderen Betriebssystemen	Nein	Nein	Nein	diverse Linux-Distributionen, macOS
Basis-Programmierframework	Component Object Model (COM)	.NET Framework	.NET Core	.NET Core/.NET

(Fortsetzung nächste Seite)

Tabelle 1.3 Vergleich WSH und PowerShell (Fortsetzung)

	Windows Script Host (WSH)	Windows PowerShell (WPS)	Windows PowerShell Core	PowerShell Core 6.x/PowerShell 7
Derzeitiger Funktionsumfang	Sehr umfangreich	Funktionsumfang in Form von Commandlets abhängig vom Betriebssystem: <ul style="list-style-type: none"> ▪ nur wenige Commandlets vor Windows 7, ▪ bessere Unterstützung ab Windows 7, ▪ sehr umfangreich erst ab Windows 8 bzw. Windows Server 2012. <p>Wichtig: Auch ohne Commandlets steht auf den älteren Betriebssystemen ein hoher Funktionsumfang zur Verfügung, wenn man COM- oder .NET-Komponenten nutzt, was jedoch mehr Wissen voraussetzt.</p>	Teilmenge von Windows PowerShell 5.1	Teilmenge von Windows PowerShell 5.1 und einige zusätzliche neue Funktionen
Weiterentwicklung der Laufzeitumgebung	Nein, nur noch beheben von Fehlern und Sicherheitslücken	Nein, nur noch beheben von Fehlern und Sicherheitslücken	Nein, nur noch beheben von Fehlern und Sicherheitslücken	Ja
Weiterentwicklung der Bibliotheken	Gering, gelegentlich erscheinen noch neue COM-Bibliotheken.	Ja, Commandlet-Erweiterungen erscheinen immer wieder mit Microsoft-Produkten.	Ja, Commandlet-Erweiterungen erscheinen immer wieder mit Microsoft-Produkten.	Ja, Microsoft wird hier in den kommenden Jahren viel investieren.
Weiterentwicklung der Werkzeuge	Nein	Ja	Ja	Ja
Basissyntax	Mächtig	Sehr mächtig	Sehr mächtig	Sehr mächtig

	Windows Script Host (WSH)	Windows PowerShell (WPS)	Windows PowerShell Core	PowerShell Core 6.x/PowerShell 7
Direkte Scripting-Möglichkeiten	Alle COM-Komponenten mit IDispatch-Schnittstelle einschließlich WMI	Alle .NET-Komponenten, alle COM-Komponenten, alle WMI-Klassen	Alle .NET-Komponenten, alle COM-Komponenten, alle WMI-Klassen	Alle .NET Standard-Komponenten, COM und WMI nur unter Windows
Scripting-Möglichkeiten über Wrapper	Alle Betriebssystemfunktionen	Alle Betriebssystemfunktionen	Viele Betriebssystemfunktionen	Viele Betriebssystemfunktionen
Werkzeuge von Microsoft	Scriptgeneratoren, Debugger, aber kein Editor	Integrated Scripting Environment (ISE), PowerShell Tools für Visual Studio, PowerShell-Erweiterung für VSCode	Integrated Scripting Environment (ISE), PowerShell Tools für Visual Studio, PowerShell-Erweiterung für VSCode	PowerShell-Erweiterung für VSCode, unter Windows auch PowerShell Tools für Visual Studio
Werkzeuge von Drittanbietern	Editoren, Debugger, Scriptgeneratoren	Editoren, Debugger, Scriptgeneratoren	Editoren, Debugger, Scriptgeneratoren	Bisher nur für Windows, siehe „Windows PowerShell“
Einarbeitungsaufwand	Hoch	Mittel bis hoch (je nach Art der PowerShell-Nutzung)	Mittel bis hoch (je nach Art der PowerShell-Nutzung)	Mittel bis hoch (je nach Art der PowerShell-Nutzung)
Informationsverfügbarkeit	Hoch	Mittlerweile auch sehr hoch	Mittlerweile auch sehr hoch	Für die Nutzung unter Windows sehr hoch, für die anderen Betriebssysteme noch sehr gering
Startanwendung	cscript.exe und wscript.exe	powershell.exe	powershell.exe	pwsh.exe (Windows) bzw. pwsh (Linux und macOS)

2

Erste Schritte mit der PowerShell

In diesem Kapitel lernen Sie erste Schritte mit der PowerShell – zunächst mit der Windows PowerShell 5.1 und dann mit der plattformneutralen PowerShell 7.

■ 2.1 Windows PowerShell herunterladen und auf anderen Windows-Betriebssystemen installieren

Die Windows PowerShell 5.1 ist in Windows 10 (ab Anniversary Update), Windows 11 und Windows Server 2016/2019/2022 (einschließlich den Zwischenversionen 1709/1909) bereits im Standard installiert.

Wenn Sie nicht Windows 10/11 oder einen der oben genannten aktuellen Windows Server benutzen, müssen Sie die PowerShell 5.1 erst installieren.

Die nachträgliche Installation der Windows PowerShell 5.1 ist auf folgenden Betriebssystemen möglich:

- Windows Server 2012 R2
- Windows Server 2012
- Windows 2008 R2
- Windows 8.1
- Windows 7

Die Windows PowerShell 5.1 wird auf diesen Betriebssystemen als Teil des Windows Management Framework 5.1 (WMF) installiert [<http://www.microsoft.com/en-us/download/details.aspx?id=54616>].

Bei der Installation ist zu beachten, dass jeweils das .NET Framework 4.5.2 oder höher vorhanden sein muss. Auch mit .NET Framework 4.6.x und 4.7 funktioniert die PowerShell 5.1.

Das WMF-5.1-Installationspaket betrachtet sich als Update für Windows (KB3191566 für Windows 7 und Windows Server 2008 R2 bzw. KB3191564 für Windows 8.1 und Windows Server 2012 R2 sowie KB3191565 für Server 2012).

<input checked="" type="checkbox"/>	W2K12-KB3191565-x64.msu	20.6 MB
<input checked="" type="checkbox"/>	Win7AndW2K8R2-KB3191566-x64.zip	64.9 MB
<input checked="" type="checkbox"/>	Win7-KB3191566-x86.zip	42.7 MB
<input checked="" type="checkbox"/>	Win8.1AndW2K12R2-KB3191564-x64.msu	19.0 MB
<input checked="" type="checkbox"/>	Win8.1-KB3191564-x86.msu	14.5 MB

Bild 2.1 Installationspaket für PowerShell 5.1 als Erweiterung

Installationsordner

Die Windows PowerShell installiert sich in folgendes Verzeichnis: `%systemroot%\system32\WindowsPowerShell\V1.0` (für 32-Bit-Systeme).



ACHTUNG: Dabei ist das `V1.0` im Pfad tatsächlich richtig: Microsoft hat dies seit Version 1.0 nicht verändert. Geplant war wohl eine „Side-by-Side“-Installationsoption wie beim .NET Framework. Doch später hat sich Microsoft dazu entschieden, dass eine neue PowerShell immer die alte überschreibt.

Auf 64-Bit-Systemen gibt es die PowerShell zweimal, einmal als 64-Bit-Version in `%systemroot%\system32\WindowsPowerShell\V1.0` und einmal als 32-Bit-Version. Letztere findet man unter `%systemroot%\Syswow64\WindowsPowerShell\V1.0`. Die 32-Bit-Version braucht man, wenn man eine Bibliothek nutzen will, für die es keine 64Bit-Version gibt, z. B. für den Zugriff auf Microsoft-Access-Datenbanken.

Es handelt sich auch dabei nicht um einen Tippfehler: Die 64-Bit-Version befindet sich in einem Verzeichnis, das „32“ im Namen trägt, und die 32-Bit-Version in einem Verzeichnis mit „64“ im Namen!

Die 32-Bit-Version und die 64-Bit-Version der PowerShell sieht man im Startmenü: Die 32-Bit-Version hat den Zusatz „(x86)“. Die 64-Bit-Version hat keinen Zusatz. Auch den Editor „ISE“ gibt es in einer 32- und einer 64-Bit-Version.

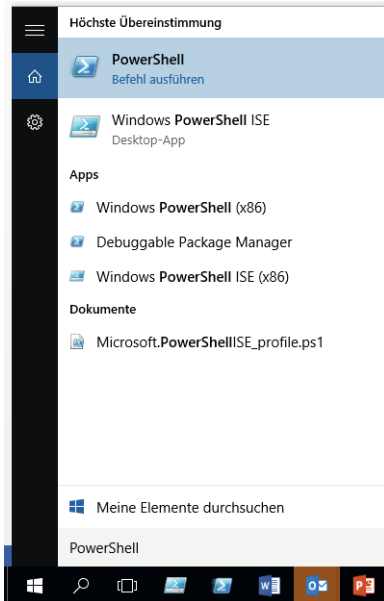


Bild 2.2
PowerShell-Einträge im Windows-10-Startmenü



TIPP: Unter Windows 8.x empfiehlt sich der Einsatz der Erweiterung www.classicshell.net, die das klassische Startmenü in Windows 8.x zurückbringt. Der Rückgriff auf ein Startmenü hat nicht nur mit Nostalgie zu tun, sondern auch ganz handfeste praktische Gründe: Der kachelbasierte Startbildschirm von Windows 8.x findet leider zum Suchbegriff „PowerShell“ weder die PowerShell ISE noch die 32-Bit-Variante der PowerShell.

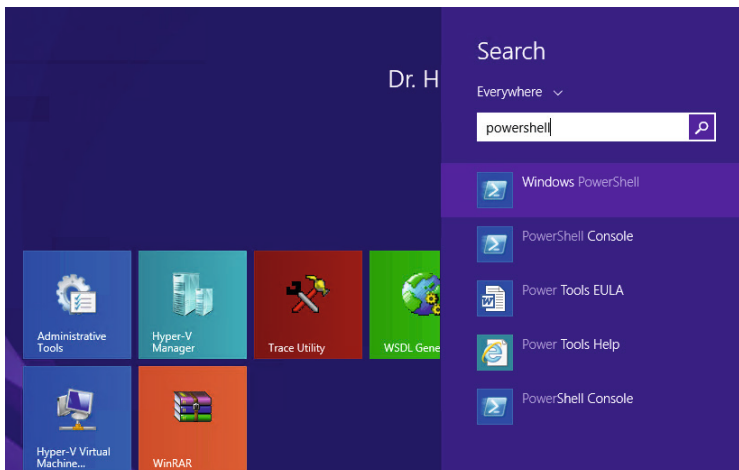


Bild 2.3 Versagen auf ganzer Linie: Der kachelbasierte Startbildschirm von Windows 8.x findet leider zum Suchbegriff „PowerShell“ weder die ISE noch die 32-Bit-Variante der PowerShell. Seit Windows 10 ist das behoben.

Unter Windows 8.x geht es mit Classic Shell:

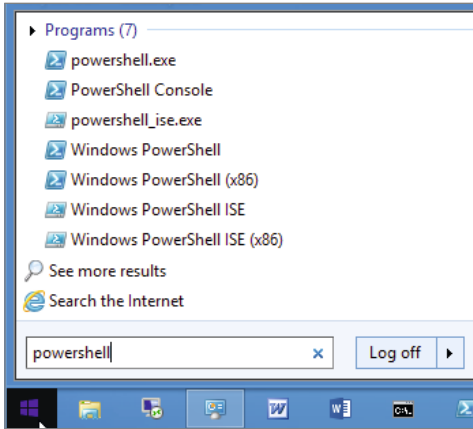


Bild 2.4

Die Classic Shell findet alle Einträge zur Windows PowerShell.

Ereignisprotokoll „PowerShell“

Durch die Installation der PowerShell wird in Windows auch ein neues Ereignisprotokoll „PowerShell“ angelegt, in dem die PowerShell wichtige Zustandsänderungen protokolliert.

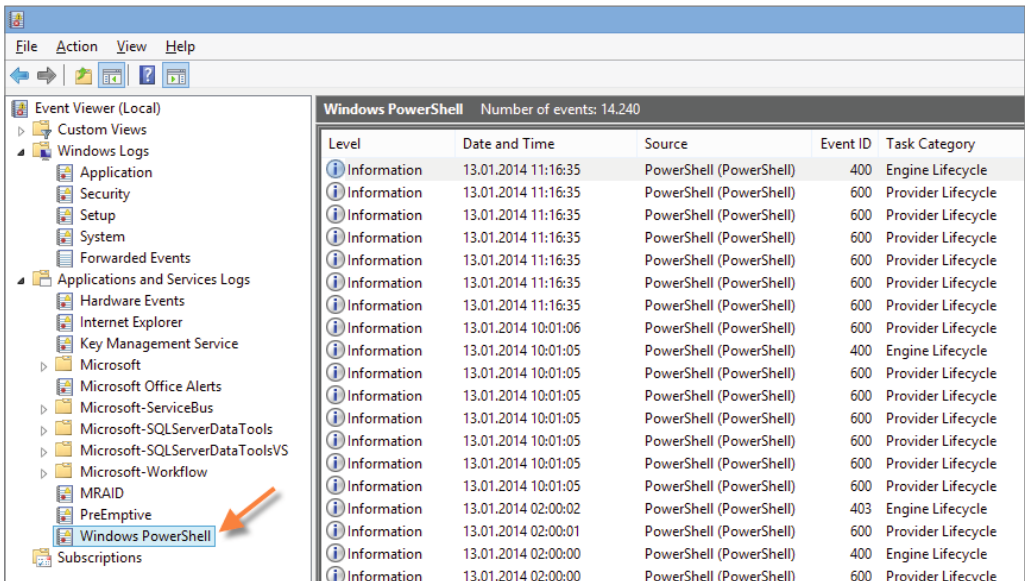


Bild 2.5 Ereignisprotokoll „Windows PowerShell“

Deinstallation

Falls man die PowerShell deinstallieren möchte, muss man dies in der Systemsteuerung unter „Programme und Funktionen/Installierte Updates anzeigen“ tun und dort das „Microsoft Windows Management Framework“ deinstallieren.

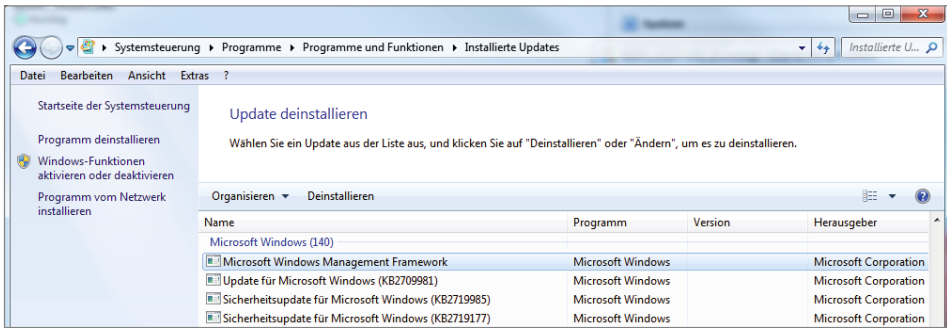


Bild 2.6 Deinstallation der PowerShell durch Deinstallation des WMF

■ 2.2 Die Windows PowerShell testen

Dieses Kapitel stellt einige Befehle vor, mit denen Sie die PowerShell-Funktionalität ausprobieren können. Die PowerShell verfügt über zwei Modi (interaktiver Modus und Skriptmodus), die hier getrennt behandelt werden.

2.2.1 PowerShell im interaktiven Modus

Der erste Test verwendet die PowerShell im interaktiven Modus.

Starten Sie bitte die PowerShell. Es erscheint ein leeres PowerShell-Konsolenfenster. Auf den ersten Blick ist kein großer Unterschied zur herkömmlichen Konsole zu erkennen. Allerdings steckt in der PowerShell mehr Kraft – im wahrsten Sinne des Wortes.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS W:\>
```

Bild 2.7 Leeres PowerShell-Konsolenfenster. Die Windows PowerShell 5.1 begrüßt den Nutzer mit Werbung für die Nachfolger. (Der Link hat auch in einem aktuellen Windows 10 noch „PSCore6“ im Namen, wird aber umgeleitet auf PowerShell 7. Microsoft hat einfach vergessen, den Link via Windows Update zu aktualisieren.)

```
C:\Users\HS\AppData\Roamin x + v
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\HS>
```

Bild 2.8 In Windows 11 ist der Link geändert, führt aber auch nicht zur Startseite von PowerShell 7, sondern zu einem Migrationsdokument [<https://docs.microsoft.com/en-us/powershell/scripting/whats-new/migrating-from-windows-powershell-51-to-powershell-7>].

Geben Sie an der Eingabeaufforderung „Get-Process“ ein (wobei die Groß-/Kleinschreibung irrelevant ist. Das gilt nicht nur für Windows, sondern auch macOS und Linux!) und drücken Sie dann die **Enter**-Taste. Es erscheint eine Liste aller Prozesse, die auf dem lokalen Computer laufen. Dies war Ihre erste Verwendung eines einfachen PowerShell-Commandlets.



HINWEIS: Beachten Sie bitte, dass die Groß-/Kleinschreibung keine Rolle spielt, da PowerShell keine Unterschiede zwischen groß- und kleingeschriebenen Commandlet-Namen macht.

Geben Sie an der Eingabeaufforderung „Get-Service i*“ ein. Jetzt erscheint eine Liste aller installierten Dienste auf Ihrem Computer, deren Namen mit dem Buchstaben „i“ beginnen. Hier haben Sie ein Commandlet mit einem Parameter verwendet.

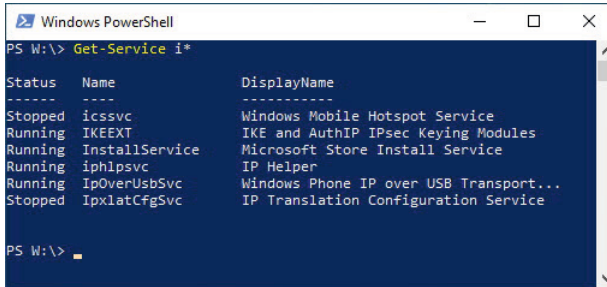
```

Windows PowerShell
PS C:\Documents\hs> get-process

Handles  NPM(K)  PM(K)  WS(K)  UM(K)  CPU(s)  Id  ProcessName
-----  -
90       5       6504   7752   79     0.03    1336  Bildschirmpausenremindersdienst
107      3       1236   1260   30     0.06    4232  cidaemon
82       3       1260   1400   30     0.06    4712  cidaemon
82       3       1336   1000   30     0.06    4892  cidaemon
414     8       2764   636    43     0.95    1376  csisc
1002    9       2460   6360   29     5.41    1652  csrss
69      3       496    2876   16     0.09    3936  ctfman
29      1       368    1612   16     0.00    1408  DeMatch
214     4       2364   4008   44     0.16    1468  dlusp
130     10      1436   3088   30     0.02    2244  dlpwdnt
57      2       668    2172   18     0.00    1416  dlsdnt
33      2       864    3308   27     0.02    4072  Launcher
563     17      21588  19532  129    28.61   2276  explorer
226     14      13120  20056  81     35.30   4004  FolderShare
180     5       2832   672    65     0.08    2640  GoogleToolbarNotifier
106     4       1732   5848   44     0.09    3804  GrooveMonitor
0       0       0       0       0       0.00    0      Idle
696     19      32596  6396   178    2.91    4724  iexplore
643     63      17016  17040  118    0.69    1504  inetinfo
63      3       528    2080   17     0.23    1524  ISRSservice
712     26      9504   11588  57     1.52    1868  Icss
55      3       3588   7404   34     0.27    3852  Matrox.PowerDesk SE
304     9       24608  24204  156    1.59    3996  Matrox.PowerDesk.PDeskNet
29      1       284    1560   14     0.02    1572  Matrox.PowerDesk.Services
24      1       204    1500   14     0.00    1536  Matrox.PowerDesk.Services
3       0       1048   3508   30     0.03    1604  mdm
418     8       9484   600    74     0.19    816  MOMHost
235     5       2996   460    44     0.41    856  MOMHost
154     9       6140   8376   60     1.06    1604  MOMService
251     131     4636   8088   47     0.13    2624  msqsv
51      2       1900   4040   24     0.39    5536  mscoresvw
165     16      1752   434    25     0.05    1148  msdtc
281     15      9472   820    75     0.28    1076  ntfers
86      3       2228   4896   33     0.17    3688  NtfsTray
83      3       1132   4304   33     0.05    3640  nvaidservice
232     6       29212  28372  133    3.33    5884  powershell
136     5       2504   5080   44     0.09    3964  rapimg
324     10      25628  28688  118    0.63    444  Rtuscan
465     13      2300   4592   73    22.88    1856  services
18      1       164    504    4     0.06    1352  smss
385     14      11544  11492  155    4.17    5284  Snagit32
287     9       6100   8612   54     0.36    1112  spoolsv
61      3       716    2400   14     0.00    1632  sqlbrowser
335     9       37472  1392  1493    0.38    1788  sqlservr
70      2       1412   4048   20     0.00    1768  sqlwriter
372     21      1748   4316   24     1.50    344  suchost
1169    58      21616  29484  191    49.17   400  suchost
156     7       3900   4684   41     0.03    456  suchost
175     6       1184   3360   22     0.02    536  suchost
39      1       300    1284   7     0.00    1332  suchost
56      2       544    2120   16     0.00    1444  suchost
95      3       1872   3220   21     0.09    2028  suchost
158     11      2968   5980   39     0.11    2084  suchost
122     4       3596   5180   26     0.05    2200  suchost
155     7       4156   7028   35     0.06    2652  suchost
77      3       2148   4052   19     0.02    2672  suchost
163     5       2420   4448   56     0.08    3832  suchost
221     8       2896   4004   32     0.09    3800  suchost
2436    0       0       236    2     20.47    4  System
35      2       656    2916   26     0.02    4620  TscHlp
94      5     26972  29004  68    19.06    2460  TSUWCache
106     4       3896   7344   38     0.89    2112  TITtoRc
64      2       1876   3420   24     0.00    3896  unsecapp
103     3       2324   4180   36     0.14    2476  UPTray
251     18      24148  33680  346    5.61    5328  w3wp
193     5     13520  14868  43     0.19    1868  wscnterm
594     86      7568   3144   53     1.30    1740  winlogon
589     27     50268  82920  400    18.36  4484  WINWORD
115     3       1404   4344   25     0.05    2136  winiprse
196     5     42120  6540   42     2.06    2464  winiprse
200     5     2392   6996   43     1.95    2772  winiprse

```

Bild 2.9 Die Liste der Prozesse ist das Ergebnis nach Ausführung des Commandlets „Get-Process“.



```

Windows PowerShell
PS W:\> Get-Service i*

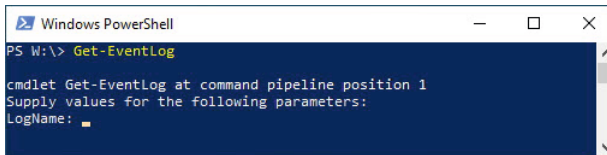
-----
Status  Name                DisplayName
-----
Stopped icssvc              Windows Mobile Hotspot Service
Running IKEEXT             IKE and AuthIP IPsec Keying Modules
Running InstallService Microsoft Store Install Service
Running iphlpsvc     IP Helper
Running IpOverUsbSvc Windows Phone IP over USB Transport...
Stopped IxlatCfgSvc   IP Translation Configuration Service

PS W:\>

```

Bild 2.10
Eine gefilterte Liste der
Windows-Dienste

Geben Sie „Get-“ ein und drücken Sie dann mehrmals die **Tab**-Taste. Die PowerShell zeigt nacheinander alle Commandlets an, die mit dem Verb „get“ beginnen. Microsoft bezeichnet diese Funktionalität als „Tabulatorvervollständigung“. Halten Sie bei „Get-Eventlog“ an. Wenn Sie **Enter** drücken, fordert die PowerShell einen Parameter namens „LogName“ an. Bei „LogName“ handelt es sich um einen erforderlichen Parameter (Pflichtparameter). Nachdem Sie „Application“ eingetippt und die Enter-Taste gedrückt haben, erscheint eine lange Liste der aktuellen Einträge in Ihrem Anwendungsereignisprotokoll.



```

Windows PowerShell
PS W:\> Get-EventLog

cmdlet Get-EventLog at command pipeline position 1
Supply values for the following parameters:
LogName:

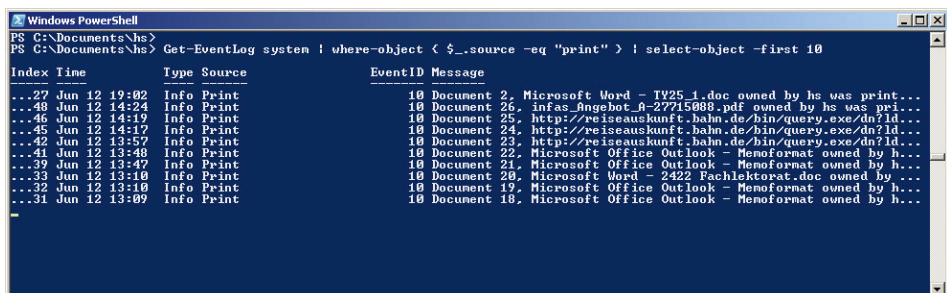
```

Bild 2.11
PowerShell fragt einen
erforderlichen Parameter ab.

Der letzte Test bezieht sich auf die Pipeline-Funktionalität der PowerShell. Auch geht es darum, die Listeneinträge aus dem Windows-Ereignisprotokoll aufzulisten, doch dieses Mal sind nur bestimmte Einträge interessant. Die Aufgabe besteht darin, die letzten zehn Ereignisse abzurufen, die sich auf das Drucken beziehen. Geben Sie den folgenden Befehl ein, der aus drei Commandlets besteht, die über Pipes miteinander verbunden sind:

```
Get-EventLog system | Where-Object { $_.source -eq "print" } | Select-Object -first 10
```

Die PowerShell scheint einige Sekunden zu hängen, nachdem die ersten zehn Einträge ausgegeben wurden. Dieses Verhalten ist korrekt, da das erste Commandlet (Get-EventLog) alle Einträge empfängt. Dieses Filtern geschieht durch aufeinanderfolgende Commandlets (Where-Object und Select-Object). Leider besitzt Get-EventLog keinen integrierten Filtermechanismus.



```

Windows PowerShell
PS C:\Documents\hs>
PS C:\Documents\hs> Get-EventLog system | where-object { $_.source -eq "print" } | select-object -first 10

Index Time           Type Source                EventID Message
-----
...27 Jun 12 19:02    Info Print                10 Document 2, Microsoft Word - T25_1.doc owned by hs was print...
...48 Jun 12 14:24    Info Print                10 Document 26, infas_angebot_0-27715089.pdf owned by hs was pri...
...46 Jun 12 14:19    Info Print                10 Document 25, http://reiseauskunft.bahn.de/bin/query.exe/dn?ld...
...45 Jun 12 14:17    Info Print                10 Document 24, http://reiseauskunft.bahn.de/bin/query.exe/dn?ld...
...42 Jun 12 13:57    Info Print                10 Document 23, http://reiseauskunft.bahn.de/bin/query.exe/dn?ld...
...41 Jun 12 13:48    Info Print                10 Document 22, Microsoft Office Outlook - Memoformat owned by h...
...39 Jun 12 13:47    Info Print                10 Document 21, Microsoft Office Outlook - Memoformat owned by h...
...33 Jun 12 13:10    Info Print                10 Document 20, Microsoft Word - 2422 Fachlektorat.doc owned by ...
...32 Jun 12 13:10    Info Print                10 Document 19, Microsoft Office Outlook - Memoformat owned by h...
...31 Jun 12 13:09    Info Print                10 Document 18, Microsoft Office Outlook - Memoformat owned by h...

```

Bild 2.12 Die Einträge des Ereignisprotokolls filtern

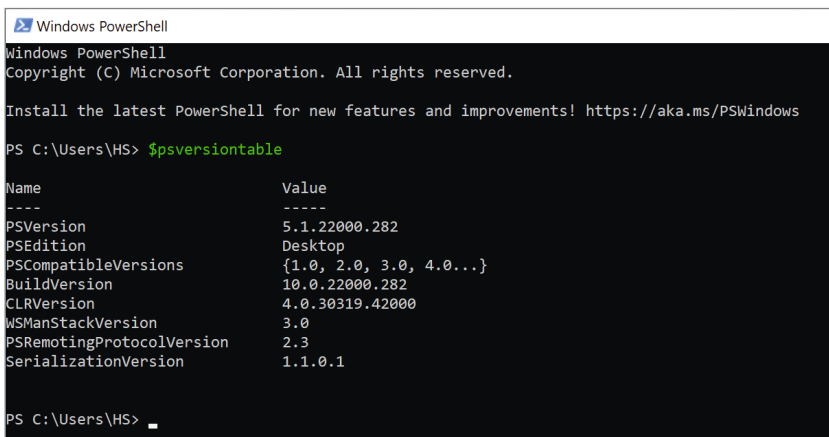
2.2.2 Installierte Version ermitteln

Die Windows PowerShell gibt bei ihrem Start ihre Versionsnummer nicht direkt preis.

Die Versionsinformation ermittelt man durch den Abruf der eingebauten Variablen `$PSVersionTable`. Neben der PowerShell-Version erhält man auch Informationen über die Frameworks und Protokolle, auf denen die PowerShell aufsetzt.

Die „CLRVersion“ steht dabei für die Version der „Common Language Runtime“ (CLR), die Laufzeitumgebung des Microsoft .NET Framework. Es fehlt in der Versionstabelle leider die Information, dass die PowerShell 5.1 zwar mit der CLR-Version 4.0 zufrieden ist, aber die .NET-Klassenbibliothek in der Version 4.5.2 oder höher benötigt, was eine Installation des .NET Frameworks 4.5.2 oder höher voraussetzt.

PowerShell Core 6.0 erfordert .NET Core 2.0. PowerShell Core 6.1 erfordert .NET Core 2.1. Allerdings braucht man .NET Core nicht separat zu installieren: Es wird beim Installationspaket von PowerShell Core mitgeliefert.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

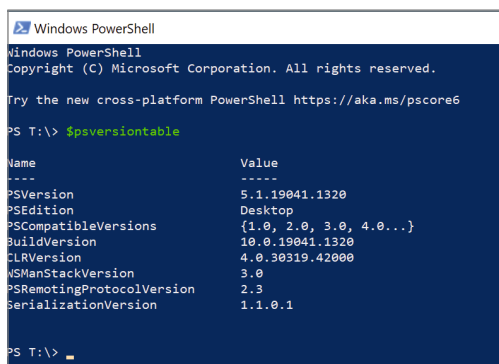
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\HS> $psversiontable

Name                           Value
----                           -
PSVersion                      5.1.22000.282
PSEdition                      Desktop
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
BuildVersion                   10.0.22000.282
CLRVersion                     4.0.30319.42000
WSManStackVersion              3.0
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1

PS C:\Users\HS> _
```

Bild 2.13 Abruf der Versionsinformationen zur Windows PowerShell 5.1 (hier unter Windows 11, Update-Stand 17.03.2022)



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS T:\> $psversiontable

Name                           Value
----                           -
PSVersion                      5.1.19041.1320
PSEdition                      Desktop
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
BuildVersion                   10.0.19041.1320
CLRVersion                     4.0.30319.42000
WSManStackVersion              3.0
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1

PS T:\> _
```

Bild 2.14

Abruf der Versionsinformationen zur Windows PowerShell 5.1 (hier unter Windows 10, Update-Stand 17.03.2022)

2.2.3 PowerShell im Skriptmodus

Bei einem PowerShell-Skript handelt es sich um eine Textdatei, die Commandlets und/oder Elemente der PowerShell-Skriptsprache (PSL) umfasst. Das zu erstellende Skript legt ein neues Benutzerkonto auf Ihrem lokalen Computer an.

2.2.4 Skript eingeben

Öffnen Sie den Windows-Editor „Notepad“ (oder einen anderen Texteditor) und geben Sie die folgenden Skriptcodezeilen ein, die aus Kommentaren, Variablendeklarationen, COM-Bibliotheksaufrufen und Shell-Ausgabe bestehen:

Listing 2.1 Ein Benutzerkonto erstellen

```
[1_Basiswissen/ErsteSchritte/LocalUser_Create.ps1]

### PowerShell-Skript
### Lokales Benutzerkonto anlegen
### (C) Holger Schwichtenberg

# Eingabewerte
$Name = "Dr. Holger Schwichtenberg"
$Accountname = "HolgerSchwichtenberg"
$Description = "Autor dieses Buchs / Website: www.powershell-doktor.de"
$Password = "secret+123"
$Computer = "localhost"

"Anlegen des Benutzerkontos $Name auf $Computer"

# Zugriff auf Container mit der COM-Bibliothek "Active Directory Service Interface"
$Container = [ADSI] "WinNT://$Computer"

# Benutzer anlegen
$objUser = $Container.Create("user", $Accountname)
$objUser.Put("Fullname", $Name)
$objUser.Put("Description", $Description)
# Kennwort setzen
$objUser.SetPassword($Password)
# Änderungen speichern
$objUser.SetInfo()

"Benutzer angelegt: $Name auf $Computer"
```

Speichern Sie die Textdatei unter dem Namen „createuser.ps1“ in einem Ordner auf der Festplatte, z. B. *w:\Skripte*. Beachten Sie, dass die Dateinamenserweiterung „.ps1“ lauten muss.



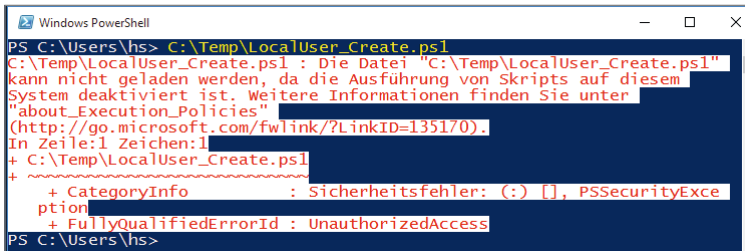
HINWEIS: Im Kapitel „Lokale Benutzer und Gruppen“ werden Sie lernen, dass es seit PowerShell 5.1 auch einen eleganteren Weg zum Anlegen lokaler Benutzer per Commandlet `New-LocalUser` gibt.

2.2.5 Skript starten

Starten Sie die PowerShell-Konsole. Versuchen Sie dort nun, das Skript zu starten. Geben Sie dazu

```
w:\Skripte\createuser.ps1
```

ein. Für die Ordner- und Dateinamen können Sie die Tabulatorvervollständigung verwenden! Der Versuch scheitert zunächst wahrscheinlich, da die Skriptausführung in der PowerShell auf den meisten Windows-Betriebssystemversionen standardmäßig nicht zulässig ist. Dies ist kein Fehler, sondern eine Sicherheitsfunktionalität. Denken Sie an den „Love Letter“-Wurm für den Windows Script Host!



```
Windows PowerShell
PS C:\Users\hs> C:\Temp\LocalUser_Create.ps1
C:\Temp\LocalUser_Create.ps1 : Die Datei "C:\Temp\LocalUser_Create.ps1"
kann nicht geladen werden, da die Ausführung von Skripten auf diesem
System deaktiviert ist. Weitere Informationen finden Sie unter
"about_Execution_Policies"
(http://go.microsoft.com/fwlink/?LinkID=135170).
In Zeile:1 Zeichen:1
+ C:\Temp\LocalUser_Create.ps1
+ ~~~~~
+ CategoryInfo          : Sicherheitsfehler: (:) [], PSSecurityExce
ption
+ FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\hs>
```

Bild 2.15
Die Skriptausführung ist standardmäßig verboten.



HINWEIS: Bisher war die PowerShell-Skriptausführung auf allen Betriebssystemen im Standard verboten. Erstmals in Windows Server 2012 R2 hat Microsoft sie im Standard erlaubt, sofern das Skript auf der lokalen Festplatte liegt. Entfernte Skripte können nur mit digitaler Signatur gestartet werden. Diese Einstellung nennt sich „RemoteSigned“. In anderen Betriebssystemen gibt es jedoch keine Änderung des Standards, der „Restricted“ lautet.

2.2.6 Skriptausführungsrichtlinie ändern

Um ein PowerShell-Skript auf Betriebssystemen wie Windows 7, Windows 8.x und Windows 10/11, wo dies im Standard nicht erlaubt ist, überhaupt starten zu können, müssen Sie die Skript-Ausführungsrichtlinie ändern. Später in diesem Buch lernen Sie, welche Optionen es dafür gibt. Für den ersten Test wird die Sicherheit ein wenig abgeschwächt, aber wirklich nur ein wenig. Mit dem folgenden Befehl lässt man die Ausführung von Skripten zu, die sich auf dem lokalen System befinden, verbietet aber Skripten von Netzwerkressourcen (das Internet eingeschlossen) die Ausführung, wenn diese keine digitale Signatur besitzen.

```
Set-ExecutionPolicy RemoteSigned
```

Später in diesem Buch lernen Sie, wie Sie PowerShell-Skripte digital signieren. Außerdem erfahren Sie, wie Sie Ihr System auf Skripte beschränken, die Sie oder Ihre Kollegen signiert haben.

Überprüfen Sie die vorgenommenen Änderungen mit dem Commandlet `Get-ExecutionPolicy`.

Es kann nun sein, dass Sie `Set-ExecutionPolicy RemoteSigned` gar nicht ausführen können und eine Fehlermeldung wie die nachstehende sehen, dass die Änderung in der Registrierungsdatenbank mangels Rechten nicht ausgeführt werden konnte.

Bild 2.16

Die Benutzerkontensteuerung verbietet die Änderung der Skriptausführungsrichtlinie.

Dies ist die Benutzerkontensteuerung, die Microsoft seit Windows Vista in Windows mitliefert. Benutzerkontensteuerung (User Account Control, UAC) bedeutet, dass alle Anwendungen seit Windows Vista immer unter normalen Benutzerrechten laufen, auch wenn ein Administrator angemeldet ist. Wenn eine Anwendung höhere Rechte benötigt (z. B. administrative Aktionen, die zu Veränderungen am System führen), fragt Windows explizit in Form eines sogenannten Consent Interface beim Benutzer nach, ob der Anwendung diese Rechte gewährt werden sollen.



HINWEIS: Nur mit Windows Server ab Version 2012 startet der eingebaute Administrator (Konto „Administrator“) alle Skripte, die Konsole und andere `.exe`-Anwendungen unter vollen Rechten. Alle anderen Administratoren unterliegen der Benutzerkontensteuerung.

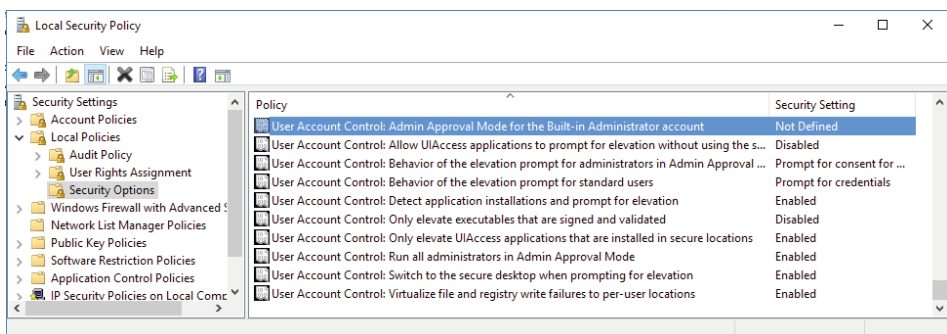


Bild 2.17 Die besondere Einstellung für den eingebauten Administrator in den Sicherheitsrichtlinien von Windows Server

Um die PowerShell mit vollen Rechten zu starten, wählen Sie aus dem Startmenü (oder einer Verknüpfung, z. B. in der Taskleiste) die PowerShell mit der rechten Maustaste aus und klicken auf „Als Administrator ausführen“.

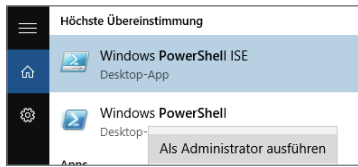


Bild 2.18
PowerShell „Als Administrator ausführen“

Dass die PowerShell als Administrator gestartet ist, sehen Sie an dem Zusatz „Administrator:“ in der Fenstertitelzeile der Konsole.

Geben Sie in diesem Fenster erneut ein:

```
Set-ExecutionPolicy RemoteSigned
```

Dies sollte nun funktionieren, wie in der nachstehenden Abbildung gezeigt.

Starten Sie nun das Skript erneut mit:

```
w:\skripte\createuser.ps1
```

Jetzt sollte die Nachricht erscheinen, dass das Benutzerkonto erstellt worden ist.

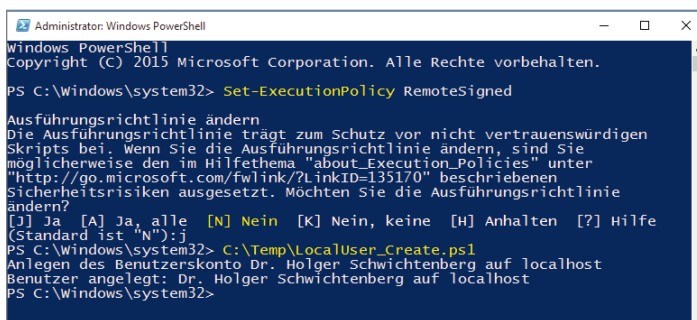


Bild 2.19 Erfolgreiches Ändern der Skriptausführungsrichtlinien und Start des Skripts „LocalUser_Create.ps1“

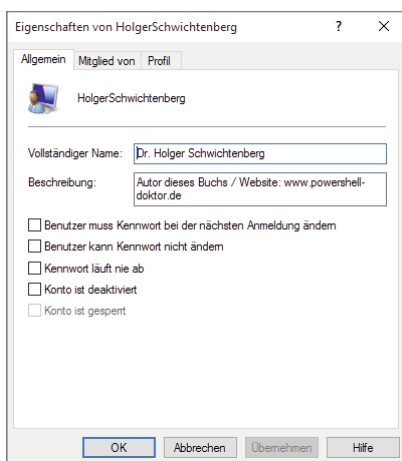


Bild 2.20
Das neu erstellte lokale Benutzerkonto

2.2.7 Einrichtungsskript ausführen

In den Downloads zu diesem Buch finden Sie eine Skriptdatei

\Einrichten der Beispiele\Einrichten Laufwerk Beispiele und Arbeitsdateien.ps1

Diese erstellt ein Laufwerk x: zu dem Standort der Beispiele auf Ihrer Festplatte. Hierzu ermittelt das Skript selbst den Standort der Beispiele.

Das Skript erstellt zudem ein Laufwerk w: als Arbeitslaufwerk für die Skripte in diesem Buch.



HINWEIS: Wenn Sie dieses Einrichtungsskript nicht ausführen wollen, dann passen Sie bitte in allen Beispielen in diesem Buch die Pfade auf Ihre Umgebung an.

Listing 2.2 Das Einrichtungsskript

```
[\Einrichten der Beispiele\Einrichten Laufwerk Beispiele und Arbeitsdateien.ps1]
# PowerShell-Buch www.IT-Visions.de/PowerShellBuch
# (C) Dr. Holger Schwichtenberg
# Einrichtungsskript

# In diesem Buch wird für den Zugriff auf die Beispieldateien das X:-Laufwerk
verwendet.
# Bitte legen Sie entweder ein Laufwerk X: an, das zu den Beispielen bei Ihnen führt,
# oder ändern Sie den Laufwerksbuchstaben in den Skripten.

(Get-Host).PrivateData.ErrorBackgroundColor = "white"
Write-host "Einrichten von Laufwerk x: für die Skripte im Buch" -ForegroundColor
Yellow
$root = (get-item $PSScriptRoot).Parent.FullName
$root
New-PSDrive x -PSProvider FileSystem -root $root | ft
subst x: $root

# Das w:-Laufwerk wird für Arbeitsdateien verwendet. Leiten Sie es zu einem leeren
Ordner
Write-host "Einrichten von Laufwerk w: für die Arbeitsdateien im Buch"
-ForegroundColor Yellow
$Arbeitsverzeichnis = "c:\PSBuch_Work" # Pfad ggf. anpassen!!!
if (-not (test-path $Arbeitsverzeichnis)) { md $Arbeitsverzeichnis -ea
SilentlyContinue }
New-PSDrive w -PSProvider FileSystem -root $Arbeitsverzeichnis | ft
subst w: $Arbeitsverzeichnis

# Auflisten
Write-host "Eingerichtete Laufwerke:" -ForegroundColor Yellow
subst

# ggf. später aufräumen:
# subst w: /d
# subst x: /d
# rd $Arbeitsverzeichnis -force -Recurse
```



```

C:\Program Files\WindowsApps\Microsoft.PowerShell_7.2.2.0_x64_8wekyb3d8bbwe\pwsh.exe
PS C:\> & "H:\TFS\Demos\PowerShell\0_Einrichten der Beispiele\Einrichten Laufwerk Beispiele und Arbeitsdateien.ps1"
Einrichten von Laufwerk x: f@r die Skripte im Buch
H:\TFS\Demos\PowerShell

Name                Used (GB)  Free (GB) Provider      Root                CurrentLocation
-----                -
x                    241,42    58,76  FileSystem    H:\TFS\Demos\PowerShell

Einrichten von Laufwerk w: f@r die Arbeitsdateien im Buch

Name                Used (GB)  Free (GB) Provider      Root                CurrentLocation
-----                -
w                    374,86    25,88  FileSystem    C:\PSBuch_work

Eingerichtete Laufwerke:
W:\: => C:\PSBuch_Work
X:\: => H:\TFS\Demos\PowerShell
PS C:\>

```

Bild 2.21 Ausführen des Einrichtungsskripts

■ 2.3 Woher kommen die PowerShell-Befehle?

In der PowerShell muss man drei Quellen für Befehle (Commandlets einschließlich Funktionen – die Unterscheidung zwischen diesen beiden Befehlsarten wird später im Buch noch erklärt) unterscheiden:

- Befehle, die zum Kern der PowerShell gehören und mit der PowerShell-Installation ausgeliefert werden
- Befehle, die zum jeweiligen Betriebssystem gehören und nicht portabel auf andere Betriebssysteme sind
- Befehle aus Zusatzmodulen, die man in der PowerShell Gallery [www.powershellgallery.com] oder aus anderen Quellen bekommt

Wenn man einmal nur die erste Gruppe zählt, dann findet man in Windows 11 mit Windows PowerShell 5.1 325 Befehle aus zehn Modulen. Zum Vergleich: Die Windows PowerShell umfasste in der Version 1.0 nur 129 Commandlets (und Funktionen). In PowerShell 2.0 waren es 236, in PowerShell 3.0 waren es 322 und in PowerShell 4.0 waren es 328. Gezählt werden hier alle Commandlets und Funktionen aus Modulen, die das Wort „PowerShell“ im Modulnamen tragen und in der Dokumentation „Core Modules“ genannt werden.

Die letzten beiden Zahlen in der folgenden Abbildung stellen die Gesamtzahl der Module (79) und der darin enthaltenen Befehle (1670) auf Windows 11 dar. Hierin ist dann die zweite oben genannte Befehlsgruppe (Befehle, die zum jeweiligen Betriebssystem gehören und nicht portabel auf andere Betriebssysteme sind) enthalten. Mit Windows 7 bzw. Windows Server 2008 R2 hatte Microsoft begonnen, Zusatzmodule direkt mit dem Betriebssystem auszuliefern. Diese Zusatzmodule bringen in Windows 8.1 die Anzahl der Commandlets auf über 1000. In Windows 10 (Stand v20H1) sind es dann 1586.

Diese Zahl bezieht sich jeweils auf eine Grundinstallation von Windows. Mehr PowerShell-Module mit weiteren Befehlen erhält man, wenn man optionale Features wie Hyper-V, Internet Information Services (IIS) oder die Remote Server Administration Tools (RSAT) installiert.

```

Windows PowerShell
PS C:\> (Get-Module *powershell* -ListAvailable).Count
10
PS C:\> (Get-Command -module *powershell* | Sort-Object name | Get-Unique).Count
325
PS C:\> (Get-Module -ListAvailable).Count
81
PS C:\> (Get-Command | Sort-Object name | Get-Unique).Count
1670
PS C:\> |

```

Bild 2.22 Zählen der Module und Befehle (Commandlets und Funktionen) in Windows PowerShell 5.1 unter Windows 11 v21H2 (Grundinstallation)

```

Administrator: Windows PowerShell
PS C:\> (Get-Module *powershell* -ListAvailable).Count
10
PS C:\> (Get-Command -module *powershell* | Sort-Object name | Get-Unique).Count
325
PS C:\> (Get-Module -ListAvailable).Count
89
PS C:\> (Get-Command | Sort-Object name | Get-Unique).Count
1799
PS C:\> _

```

Bild 2.23 Zählen der Module und Befehle (Commandlets und Funktionen) in Windows PowerShell 5.1 unter Windows Server 2022 (Grundinstallation)

```

Windows PowerShell
PS C:\scripts> (Get-Module *powershell* -ListAvailable).Count
10
PS C:\scripts> (get-command -module *powershell* | sort-object name | Get-Unique).Count
325
PS C:\scripts> (Get-Module -ListAvailable).Count
79
PS C:\scripts> (get-command | sort-object name | Get-Unique).Count
1586
PS C:\scripts> _

```

Bild 2.24 Zählen der Module und Befehle (Commandlets und Funktionen) in Windows PowerShell 5.1 unter Windows 10 v20H1 (Grundinstallation)



ACHTUNG: Anders als die Erweiterungsmodule, die es oft für mehrere (auch ältere) PowerShell-Versionen gibt, kann man die zum Betriebssystem gehörenden Module nicht in einem älteren Betriebssystem verwenden. In dem zum Redaktionschluss dieses Buchs aktuellen Stand der PowerShell 7.2 kann man viele, aber noch nicht alle zum Windows-Betriebssystem gehörenden PowerShell-Module auch in PowerShell 7.2 unter Windows verwenden.

■ 2.4 PowerShell Community Extensions (PSCX) herunterladen und installieren

Bei den „PowerShell Community Extensions“ (kurz PSCX) handelt es sich um ein Open Source-Projekt (ursprünglich auf Codeplex.com, mittlerweile auf [Github.com](https://github.com), siehe github.com/Pscx/Pscx), das zusätzliche Funktionalität mit Commandlets für die Windows PowerShell realisiert, wie zum Beispiel `Get-DHCPserver`, `Get-DomainController`, `Get-MountPoint`, `Get-TerminalSession`, `Set-VolumeLabel`, `Write-Tar` und viele weitere.

Früher wurden in regelmäßigen Abständen neue Versionen veröffentlicht. Die aktuelle Version zum Reaktionsschluss dieses Buchs ist die Version 4.0.0-beta4 vom 8.1.2022. Diese Version läuft sowohl unter Windows PowerShell als auch unter PowerShell 7.x.



TIPP: In diesem Buch werden an einigen Stellen Commandlets aus den PSCX verwendet, da diese zusätzliche Funktionen bieten, die nicht im Kern der PowerShell enthalten sind. Daher sollten Sie die PSCX bei sich installieren.

Die PSCX sind ein sehr beliebtes Modul (siehe Abrufstatistik links in der folgenden Abbildung).

The screenshot shows the PowerShell Gallery page for the PSCX 4.0.0-beta4 package. The page layout includes a search bar at the top, a navigation menu, and a main content area with the following details:

- Downloads:** 674,395 total, 238 for 4.0.0-beta4. Last published: 1/8/2022.
- Description:** PowerShell Community Extensions (PSCX) base module which implements a general purpose set of Cmdlets.
- Minimum PowerShell version:** 5.0
- Installation Options:** Install Module, Azure Automation, Manual Download.
- Command:** `PS> Install-Module -Name Pscx -AllowPrerelease`
- Author(s):** Keith Hill, Oisín Grehan and contributors.
- Copyright:** (c) 2006 - 2020 Keith Hill, Oisín Grehan, and contributors.
- Version History Table:**

Version	Downloads	Last updated
4.0.0-beta4 (current version)	238	2 months ago
4.0.0-beta3	22	3 months ago
4.0.0-beta2	2,435	10/22/2020
4.0.0-beta1	20	10/18/2020
3.3.2	607,708	1/17/2018

Bild 2.25 PowerShell-Gallery-Webseite zu den PSCX. Es gab 2019 und 2021 keine neuen Versionen. [www.powershellgallery.com/packages/Pscx]

Die Installation der PSCX führt man heutzutage am einfachsten über das Commandlet `Install-Module` aus. Dieses Commandlet lädt das Modul aus der PowerShell Gallery [www.powershellgallery.com], einem von Microsoft betriebenen Online-Portal mit PowerShell-Erweiterungen, und installiert das Modul. Alternativ dazu können Sie auf Github ein ZIP-Paket laden [github.com/Pscx/Pscx/releases] und die Installation manuell vornehmen.

Allerdings ist die Installation nicht ganz so trivial, wie man es sich wünscht. Beim Aufruf von

```
Install-Module PSCX
```

kommt zuerst die Warnmeldung, dass der NuGet-Provider in Windows nicht aktuell ist (das passiert sogar in einem topaktuellen Windows 11 und Windows Server 2022). Die PowerShell Gallery basiert auf dem Paketmanager NuGet (siehe www.NuGet.org).

Wenn man dies bestätigt, kommt die Nachfrage, ob man der PowerShell Gallery trauen möchte. Danach erscheint die Fehlermeldung, dass in den PSCX einige Commandlet-Namen implementiert sind, die es in der aktuellen PowerShell schon gibt. (Dies hat historische Gründe: PSCX hat diese zuerst implementiert, dann hat Microsoft später gleichnamige Commandlets implementiert, die aber im Detail manchmal etwas anders funktionieren.)

Man muss mit `-AllowClobber` bestätigen, dass man diesen Zustand tolerieren will:

```
Install-Module PSCX -AllowClobber
```

Danach sieht man in der Modulliste, die man via

```
Get-Module PSCX -ListAvailable
```

bekommt, dass nur Version 3.3.2 aus dem Jahr 2018 installiert wurde.

Wenn man nun aber die Version 4.0.0-Beta4 aus dem Jahr 2022 haben möchte, soll man laut der Webseite in der PowerShell Gallery den Parameter `-AllowPrerelease` ergänzen. Allerdings merkt man schon bei der fehlenden Eingabehilfe, dass es den nicht gibt. Es gibt zwar den Parameter `-RequiredVersion`, mit dem man dies versuchen kann:

```
Install-Module -Name PSCX -RequiredVersion "4.0.0-beta4"
```

Das wiederum scheitert daran, dass man früher solche Versionsnummern noch nicht bilden durfte.

Grundproblem ist, dass auch im aktuellen Windows-Client und Windows Server immer noch die total **veraltete Version 1.0.0.1 des Moduls PowerShellGet** mitgeliefert wird. Die aktuelle Version laut <https://www.powershellgallery.com/packages/PowerShellGet> ist aber 2.2.5 (stabil) bzw. 3.0.12-beta (Prerelease).

```

Administrator: Windows Powe x + v
PS C:\> Install-Module PSCX

NuGet provider is required to continue
PowerShellGet requires NuGet provider version '2.8.5.201' or newer to interact with NuGet-based repositories. The NuGet
provider must be available in 'C:\Program Files\PackageManagement\ProviderAssemblies' or
'C:\Users\Administrator\AppData\Local\PackageManagement\ProviderAssemblies'. You can also install the NuGet provider by
running 'Install-PackageProvider -Name NuGet -MinimumVersion 2.8.5.201 -Force'. Do you want PowerShellGet to install
and import the NuGet provider now?
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"): Y
running 'Install-PackageProvider -Name NuGet -MinimumVersion 2.8.5.201 -Force'. Do you want PowerShellGet to install
and import the NuGet provider now?
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"): Y

Untrusted repository
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "N"): A
PackageManagement\Install-Package : The following commands are already available on this
system: 'gcb, Expand-Archive, Format-Hex, Get-Hash, help, prompt, Get-Clipboard, Get-Help, Set-Clipboard'. This module 'Pscx'
may override the existing commands. If you still want to install this module 'Pscx', use -AllowClobber parameter.
At C:\Program Files\WindowsPowerShell\Modules\PowerShellGet\1.0.0.1\PSModule.psm1:1809 char:21
+ ...          $null = PackageManagement\Install-Package @PSBoundParameters
+ ~~~~~
+ CategoryInfo          : InvalidOperation: (Microsoft.Power...InstallPackage:InstallPackage),
Exception
+ FullyQualifiedErrorId : CommandAlreadyAvailable,Validate-ModuleCommandAlreadyAvailable,Microsoft.PowerShell.Pack
ageManagement.Cmdlets.InstallPackage

PS C:\> Install-Module PSCX -AllowClobber

Untrusted repository
You are installing the modules from an untrusted repository. If you trust this repository, change its
InstallationPolicy value by running the Set-PSRepository cmdlet. Are you sure you want to install the modules from
'PSGallery'?
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "N"):
A

PS C:\> Get-Module PSCX -ListAvailable

Directory: C:\Program Files\WindowsPowerShell\Modules

ModuleType Version Name ExportedCommands
-----
Script 3.3.2 Pscx {Add-PathVariable, Clear-MSMQueue, ConvertFrom-Base64, Con...

PS C:\> |

```

Bild 2.26 Installation der PSCX in der aktuellen stabilen Version, die aber aus dem Jahr 2018 stammt

```

Administrator: Windows Powe x + v
PS C:\> Install-Module -Name PSCX -AllowPreRelease
Install-Module : A parameter cannot be found that matches parameter name 'AllowPreRelease'.
At line:1 char:27
+ Install-Module -Name PSCX -AllowPreRelease
+ ~~~~~
+ CategoryInfo          : InvalidArgument: (:) [Install-Module], ParameterBindingException
+ FullyQualifiedErrorId : NamedParameterNotFound,Install-Module

PS C:\> Install-Module -Name PSCX -RequiredVersion "4.0.0-beta4"
Install-Module : Cannot process argument transformation on parameter 'RequiredVersion'. Cannot convert value
"4.0.0-beta4" to type "System.Version". Error: "Input string was not in a correct format."
At line:1 char:44
+ Install-Module -Name PSCX -RequiredVersion "4.0.0-beta4"
+ ~~~~~
+ CategoryInfo          : InvalidData: (:) [Install-Module], ParameterBindingArgumentTransformationException
+ FullyQualifiedErrorId : ParameterArgumentTransformationError,Install-Module

PS C:\> Get-Module PowerShellGet -ListAvailable

Directory: C:\Program Files\WindowsPowerShell\Modules

ModuleType Version Name ExportedCommands
-----
Script 1.0.0.1 PowerShellGet {Install-Module, Find-Module, Save-Module, Update-Module...}

PS C:\>

```

Bild 2.27 PowerShellGet ist veraltet und kennt kein -AllowPreRelease und keine Versionsnummern mit Buchstaben.

The screenshot shows the PowerShell Gallery website for the PowerShellGet 2.2.5 package. The page features a search bar at the top with the text "PowerShellGet, Get-AzVM, etc...". Below the search bar, the package name "PowerShellGet 2.2.5" is displayed. To the left of the package name, there is a logo and statistics: 62,093,555 Downloads, 7,804 Downloads of 3.0.12-beta, and a last published date of 9/22/2020. The main content area includes a description of the PowerShell module, a minimum PowerShell version of 3.0, and a notification about a newer prerelease version. Below this, there are installation options: "Install Module", "Azure Automation", and "Manual Download". A code block shows the command: `PS> Install-Module -Name PowerShellGet -RequiredVersion 2.2.5 -Force`. The page also lists the author as Microsoft Corporation and includes a copyright notice. At the bottom, there is a version history table.

Version	Downloads	Last updated
3.0.12-beta	7,804	3 months ago
3.0.11-beta	11,346	7 months ago
3.0.0-beta10	50,288	9/5/2020
3.0.0-beta1	8,674	3/30/2020
2.2.5 (current version)	33,813,967	9/22/2020
2.2.4.1	11,945,073	4/22/2020

Bild 2.28 PowerShellGet in der PowerShell Gallery [<https://www.powershellgallery.com/packages/PowerShellGet>]

Nun also gilt es, zuerst einmal das Modul PowerShellGet auf den aktuellen Stand zu bringen. Dazu aktualisiert man zur Sicherheit erst einmal den NuGet-Treiber:

```
Install-PackageProvider -Name NuGet -Force
```

Danach führt man aus:

```
Install-Module -Name PowerShellGet -force
```

Der Zusatz `-force` ist notwendig, weil es das Modul ja schon gibt. (Es gibt einen Befehl `Update-Module`, der funktioniert hier aber nicht, weil PowerShellGet ein Systemmodul ist. `Update-Module` lässt sich nur für von der PowerShell Gallery installierte Module verwenden).

Danach hat man dann zwei Versionen von PowerShellGet.

```

Administrator: Windows PowerShell
PS C:\> Install-PackageProvider -Name NuGet -Force

Name                Version      Source          Summary
-----                -
nuget                2.8.5.208   https://onege... NuGet provider for the OneGet meta-package manager

PS C:\> Install-Module -Name PowerShellGet -force
PS C:\> Get-Module PowerShellGet -ListAvailable

Directory: C:\Program Files\WindowsPowerShell\Modules

ModuleType Version      Name                ExportedCommands
-----
Script     2.2.5       PowerShellGet       {Find-Command, Find-DSCResource, Find-Module, Find-RoleCap...}
Script     1.0.0.1     PowerShellGet       {Install-Module, Find-Module, Save-Module, Update-Module...}

PS C:\>

```

Bild 2.29 Aktualisieren von PowerShellGet



TIPP: Nun gehen Sie ins Verzeichnis `C:\Program Files\WindowsPowerShell\Modules\PowerShellGet` und löschen das Unterverzeichnis `1.0.0.1`, um das alte Modul loszuwerden! Tun Sie Gleiches, wenn es dieses Unterverzeichnis in `C:\Windows\system32\WindowsPowerShell\v1.0\Modules\PowerShellGet` gibt. Prüfen Sie gegebenenfalls mit `get-module powershellget -ListAvailable`, ob Sie noch irgendwo eine alte Version haben!

Das Deinstallieren ganz alter Versionen geht nicht via `Uninstall-Module`, weil es ein Systemmodul ist.

Nun endlich, wo Sie nur noch PowerShellGet in Version 2.2.5 haben, können Sie die PSCX-Betaversion installieren mit:

```
Install-Module PSCX -AllowPrerelease
```

```

Administrator: Windows PowerShell
PS C:\> Get-Module PowerShellGet -ListAvailable

Directory: C:\Program Files\WindowsPowerShell\Modules

ModuleType Version      Name                ExportedCommands
-----
Script     2.2.5       PowerShellGet       {Find-Command, Find-DSCResource, Find-Module, Find-RoleCap...}

PS C:\> Install-Module PSCX -AllowPrerelease

Untrusted repository
You are installing the modules from an untrusted repository. If you trust this repository, change its
InstallationPolicy value by running the Set-PSRepository cmdlet. Are you sure you want to install the modules from
'PSGallery'?
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "N"): A
PS C:\> Get-Module PSCX -ListAvailable

Directory: C:\Program Files\WindowsPowerShell\Modules

ModuleType Version      Name                ExportedCommands
-----
Script     4.0.0       Pscx                {Add-PathVariable, ConvertFrom-Base64, ConvertTo-Base64, C...}

PS C:\>

```

Bild 2.30 Installation der PSCX in der aktuellen Beta-Version aus dem Jahr 2022



TIPP: Wenn Sie kein Freund von Beta-Versionen sind, könnten Sie sich mit der stabilen Version 3.3.2 zufriedengeben und diese ganzen Modulupdates vermeiden wollen. Bitte beachten Sie aber: Die Version 3.3.2 ist aus dem Jahr 2018. Die Version 4.0, die es bisher nur als Beta gibt, löst insbesondere das Problem der doppelten Commandlet-Namen. Daher brauchen Sie kein `-AllowClobber` mehr bei der Installation der PSCX 4.0. PSCX hat in Version 4.0 das Problem der historisch bedingten doppelten Commandlet-Namen beseitigt, indem man die PSCX-Commandlets umbenannt hat.

```

12 4.0.0-beta2 - October 22, 2020
13
14 * Renamed less function to PscxLess.
15 * Renamed help function to PscxHelp.
16 * Renamed prompt function to PscxPrompt.
17 * Renamed Get-ADObject to Get-PscxADObject.
18 * Renamed Get-Help to Get-PscxHelp.
19 * Renamed Mount/Dismount-VHD to Mount/Dismount-PscxVHD.
20
21 * Changed Pscx to only override the built-in help function if PageHelpUsingLess Pscx.UserPreference is $true
22 * Changed default value of Pscx.UserPreference to be $true only on PowerShell v5.
23
24 4.0.0-beta1 - October 17, 2020
25
26 BREAKING CHANGES - PLEASE READ
27 * Minimum version of PowerShell is 5.0.
28 * Migrate to .NET 4.61.
29 * Renamed Expand-Archive to Expand-PscxArchive and Read-Archive to Read-PscxArchive.
30 * Renamed Set-LocationEx to Set-PscxLocation.
31 * Renamed all *-Clipboard commands to *-PscxClipboard.
32 * Renamed Format-Hex command to Format-PscxHex.
33 * Renamed Get-Uptime to Get-PscxUptime.
34 * Renamed Join-String to Join-PscxString.
35
36 * Removed redefinition of the cd alias
37 * Removed the gcb alias that now conflicts with the built-in gcb alias
38 * Removed ?? alias to avoid conflict with ?? operator in PS 7.
39 * Removed ?: alias since PS 7 now implements a true ternary operator.
40
41 * Fixed Expand-PscxArchive help topic to remove references to the Format parameter - this parameter does not exist.
42 * Changed help function to default to displaying Full help details.
43

```

Bild 2.31 Auszug aus den Release Notes der PSCX 4.0 [<https://github.com/Pscx/Pscx/blob/master/ReleaseNotes.txt>]

Bevor Sie nun einen Befehl der PSCX ausführen können, müssen Sie in jeder neuen Instanz der PowerShell den `Import-Befehl` für die PSCX einmalig ausführen.

```
Import-Module PSCX
```

Sie lernen später in diesem Buch, wie man solche Befehle in der PowerShell-Profildatei hinterlegt und dafür sorgt, dass diese automatisch in jeder neuen Instanz ausgeführt werden.

Geben Sie nun `Get-DomainController` ein (wenn Ihr Computer Mitglied einer Active Directory-Domäne ist) oder testen Sie die PSCX mit dem Befehl `Ping-Host`, der auf jedem Computer im Netzwerk funktioniert.

Wie Sie in der Abbildung an der Ausgabe zu Ping-Host lesen können: Es ist ein Commandlet, für das es mittlerweile in der PowerShell einen Ersatz (hier: Test-Connection) gibt. Daher erhalten Sie eine Warnung, dass der Befehl obsolet sei.

```

Administrator: Windows PowerShell
PS C:\> Import-Module PSCX
PS C:\> Get-DomainController

Forest                : IT-Visions.local
CurrentTime           : 19.03.2022 08:50:04
HighestCommittedUsn  : 40354373
OSVersion             : Windows Server 2016 Standard
Roles                 : {SchemaRole, NamingRole, PdcRole, RidRole...}
Domain               : IT-Visions.local
IPAddress             : 192.168.1.31
SiteName              : Default-First-Site
SyncFromAllServersCallback :
InboundConnections   : {182a0349-7e20-4a73-a8e6-7a155182c66a, c1686568-e259-4b3b-901d-f3a2fafbe258,
                        f93a5fa1-56cc-4785-b262-8be041f3cf07}
OutboundConnections  : {66a5ff30-aa64-46b3-af51-c2a0e08e019c, e1ea8652-0b68-4761-b6d9-4b54a7e25e79}
Name                  : IT-Visions.local
Partitions            : {DC=IT-Visions,DC=local, CN=Configuration,DC=IT-Visions,DC=local,
                        CN=Schema,CN=Configuration,DC=IT-Visions,DC=local,
                        DC=DomainDnsZones,DC=IT-Visions,DC=local...}

PS C:\> Get-Command Get-DomainController

CommandType  Name                               Version  Source
-----
Cmdlet       Get-DomainController              4.0.0   PSCX

PS C:\> Ping-Host www.IT-Visions.de
WARNING: The command 'Ping-Host' is obsolete. The PSCX\Ping-Host cmdlet is obsolete and will be removed in the next
version of PSCX. Use the built-in Microsoft.PowerShell.Management\Test-Connection cmdlet instead.
Pinging www.IT-Visions.de [81.20.82.74] with 32 bytes of data:
    Reply from 81.20.82.74 bytes=32 time=11ms TTL=120
    Reply from 81.20.82.74 bytes=32 time=10ms TTL=120
    Reply from 81.20.82.74 bytes=32 time=11ms TTL=120
    Reply from 81.20.82.74 bytes=32 time=9ms TTL=120
  
```

Bild 2.32 PSCX-Befehle Get-DomainController und Ping-Host testen

■ 2.5 Den Windows PowerShell-Editor „ISE“ verwenden

Integrated Scripting Environment (ISE) ist der Name des Skripteditors, den Microsoft seit der Windows PowerShell 2.0 mitliefert und der in Windows PowerShell 3.0 nochmals erheblich verbessert wurde. Die ISE startet man mit dem Symbol „PowerShell ISE“ oder indem man in der PowerShell den Befehl „ise“ ausführt.

Die ISE verfügt über zwei Fenster: ein Skriptfenster (im Standard oben, alternativ über „View“-Menü einstellbar rechts) und ein interaktives Befehlseingabefenster (unten bzw. links). Optional kann man ein drittes Fenster einblenden, das „Command Add-On“, in dem man Befehle suchen kann und eine Eingabehilfe für Befehlsparameter erhält.

Geben Sie unten im interaktiven Befehlseingabefenster in der ISE ein:

```
Get-Process
```

Nachdem Sie mindestens einen Buchstaben eingegeben haben, können Sie die Eingabe mit der Tabulatortaste vervollständigen. Alternativ können Sie **STRG + Leertaste** für eine Eingabehilfe mit Auswahlfenster (IntelliSense) drücken. Die Ausgaben des interaktiven Bereichs erscheinen dann direkt unter den Befehlen, wie bei der PowerShell-Konsole. Einen dedizierten Ausgabebereich wie in der ISE in PowerShell 2.0 gibt es nicht mehr.

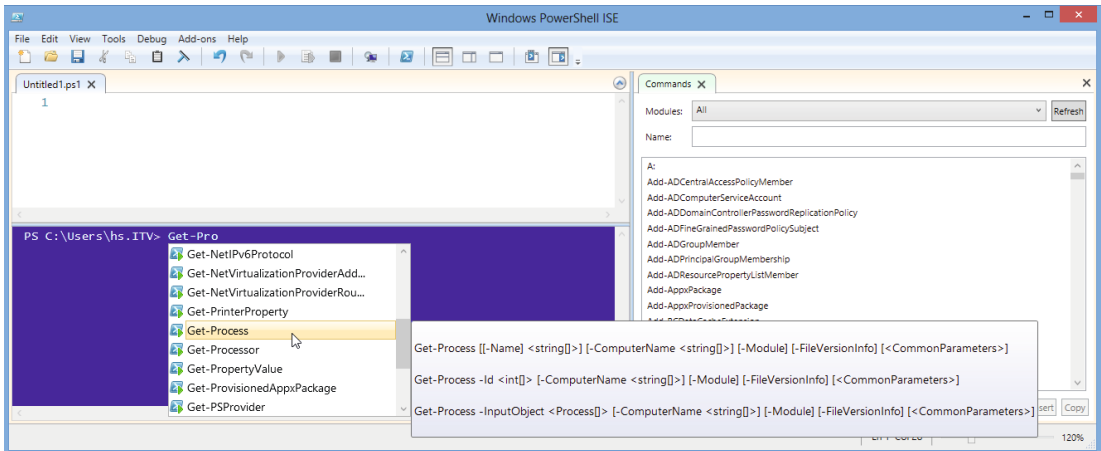


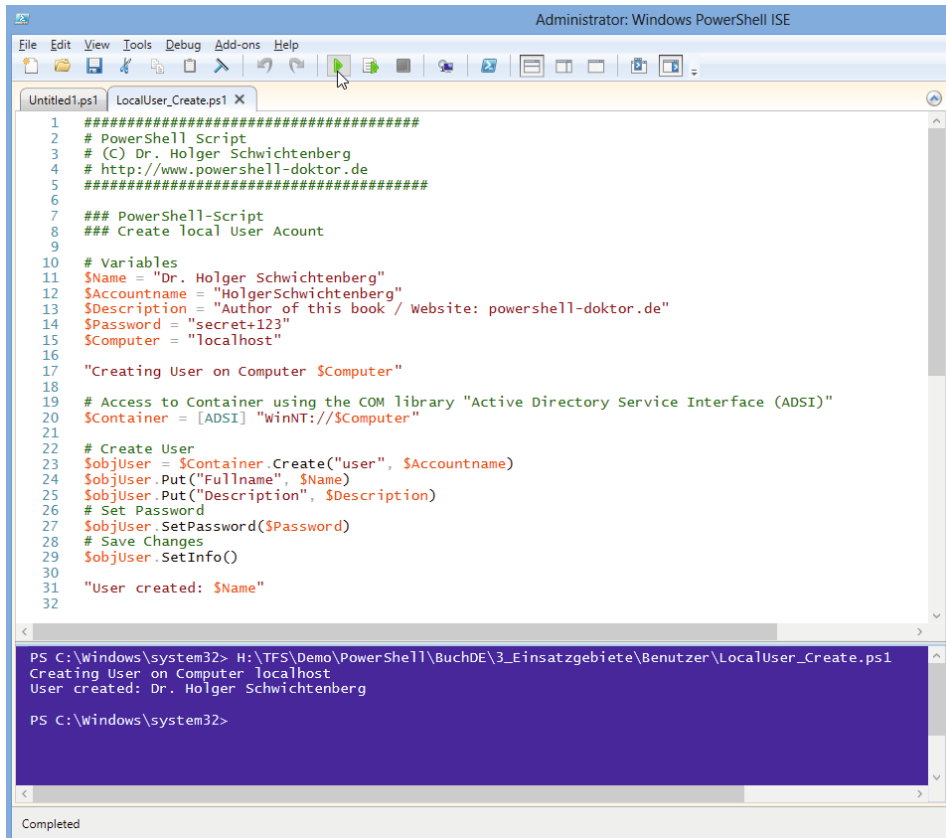
Bild 2.33 IntelliSense-Eingabehilfe

Um die ISE im Skriptmodus zu verwenden, erstellen Sie eine neue Skriptdatei (Menü „File/New“) oder öffnen Sie eine vorhandene *.ps1*-Datei (Menü „File/Open“). Öffnen Sie als Beispiel die Skriptdatei *CreateUser.ps1*, die Sie zuvor erstellt haben. Es sind Zeilennummern zu sehen. Die verschiedenen Bestandteile des Skripts sind in unterschiedlichen Farben dargestellt. Auch hier funktioniert die Eingabeunterstützung mit der Tabulatortaste und IntelliSense.

Um das Skript auszuführen, klicken Sie auf das Start-Symbol in der Symbolleiste (siehe folgende Abbildung) oder drücken Sie **F5**. Auch hier wird das Ergebnis im interaktiven Bereich angezeigt.



TIPP: Stellen Sie sicher, dass Sie die ISE als Administrator ausführen und dass das Benutzerkonto noch nicht existiert, bevor Sie das Skript ausführen.



```

1 #####
2 # PowerShell Script
3 # (C) Dr. Holger Schwichtenberg
4 # http://www.powershell-doktor.de
5 #####
6
7 ### PowerShell-Script
8 ### Create local User Account
9
10 # Variables
11 $Name = "Dr. Holger Schwichtenberg"
12 $Accountname = "HolgerSchwichtenberg"
13 $Description = "Author of this book / Website: powershell-doktor.de"
14 $Password = "secret+123"
15 $Computer = "localhost"
16
17 "Creating User on Computer $Computer"
18
19 # Access to Container using the COM library "Active Directory Service Interface (ADSI)"
20 $Container = [ADSI] "WinNT://$Computer"
21
22 # Create User
23 $objUser = $Container.Create("user", $Accountname)
24 $objUser.Put("FullName", $Name)
25 $objUser.Put("Description", $Description)
26 # Set Password
27 $objUser.SetPassword($Password)
28 # Save Changes
29 $objUser.SetInfo()
30
31 "User created: $Name"
32

```

```

PS C:\Windows\system32> H:\TFS\Demo\PowerShell\BuchDE\3_Einsatzgebiete\Benutzer\LocalUser_Create.ps1
Creating User on Computer localhost
User created: Dr. Holger Schwichtenberg

PS C:\Windows\system32>

```

Bild 2.34 Die ISE im Skriptmodus

Eine hilfreiche Funktion der ISE ist das Debugging, mit dem Sie ein Skript Zeile für Zeile durchlaufen und währenddessen den Zustand der Variablen betrachten können.

Setzen Sie dafür den Cursor auf eine beliebige Zeile in Ihrem Skript und tippen Sie dann auf F9 (oder wählen Sie „Toggle Breakpoint“ im Kontextmenü oder im Menü „Debug“). Daraufhin erscheint die Zeile in Rot – ein sogenannter „Haltepunkt“.

Starten Sie das Skript nun mit F5. Die ISE stoppt in der Zeile mit dem Haltepunkt, und diese wird orange. Mit der Taste F10 springen Sie zum nächsten Befehl in der folgenden auszuführenden Zeile. Diese wird dann gelb, und die Zeile mit dem Haltepunkt wird wieder rot.



HINWEIS: Die gelbe Zeile ist immer die nächste Zeile, die ausgeführt wird.

```

10 # Variables
11 $Name = "Dr. Holger Schwichtenberg"
12 $Accountname = "HolgerSchwichtenberg"
13 $Description = "Author of this book / Website: powershell-doktor.de"
14 $Password = "set+123"
15 $Computer = "localhost"
16
17 "Creating User on Computer $Computer"
18
19 # Access to Container using the COM library "Active Directory Service Interface (ADSI)"
20 $Container = [ADSI] "WinNT://$Computer"
21
22 # Create User
23 $ObjUser = $Container.Create("user", $Accountname)
24 $ObjUser.Put("FullName", $Name)
25 $ObjUser.Put("Description", $Description)
26 # Set Password
27 $ObjUser.SetPassword($Password)
28 # Save Changes
29 $ObjUser.SetInfo()
30
31 "User created: $Name"
32

```

```

[DBG]: PS C:\Windows\system32>> $Computer
localhost

[DBG]: PS C:\Windows\system32>> $Container

distinguishedName :
Path              : WinNT://localhost

[DBG]: PS C:\Windows\system32>> |

```

Bild 2.35 Skript-Debugging mit der ISE

Im interaktiven Bereich können Sie im Haltemodus den aktuellen Zustand der Variablen abfragen, indem Sie dort z. B. eingeben

```
$Computer
```

oder

```
$Container
```

Man kann auch Werte interaktiv ändern. Um das Skript fortzusetzen, drücken Sie wieder **F5**. Über das Menü „Debug“ sind weitere Steuerbefehle möglich.



HINWEIS: Sie müssen den Debugger vorher beenden (Menüpunkt „Debug/Stop Debugger“), wenn Sie das Skript erneut ändern möchten.

■ 2.6 PowerShell 7 installieren und testen

Dieses Unterkapitel behandelt die Installation der plattformneutralen PowerShell 7 auf Windows, Linux und macOS. Die PowerShell 7 wird bislang mit keinem Betriebssystem direkt ausgeliefert. Auch in Windows 11 ist nur die Windows PowerShell 5.1 im Standard enthalten.

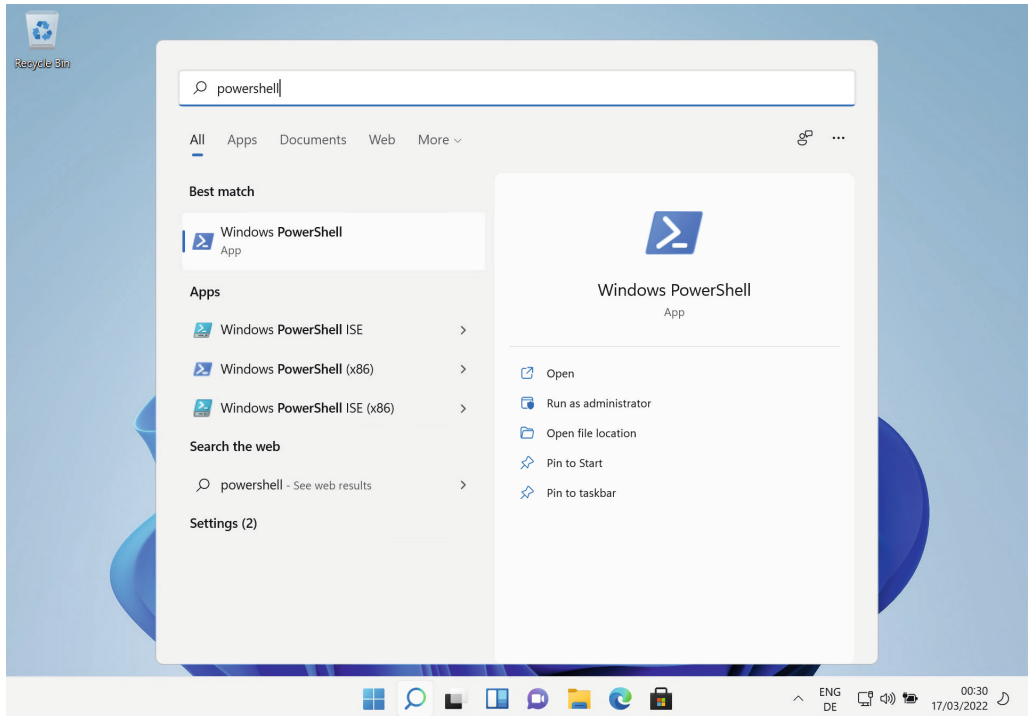


Bild 2.36 Nur Windows PowerShell 5.1 in Windows 11



TIPP: Sie können dieses Unterkapitel überspringen, wenn Sie nur die klassische Windows PowerShell einsetzen wollen. Details zur PowerShell 7 lesen Sie in Kapitel 13 „PowerShell 7 für Windows, Linux und macOS“.

2.6.1 Basis für PowerShell 7

Die verschiedenen Versionen der PowerShell 7 basieren auf verschiedenen Versionen des modernen .NET:

- PowerShell 7.0 basiert auf .NET Core 3.1.
- PowerShell 7.1 basiert auf .NET 5.0.
- PowerShell 7.2 basiert auf .NET 6.0.
- PowerShell 7.3 basiert auf .NET 7.0.



HINWEIS: Die jeweilige .NET-Version wird bei der PowerShell bereits mitgeliefert. Die passende .NET-Version braucht vorher also nicht installiert zu werden. Man nennt dies eine „Self-Contained App“ (SCA).

2.6.2 Installation und Test auf Windows

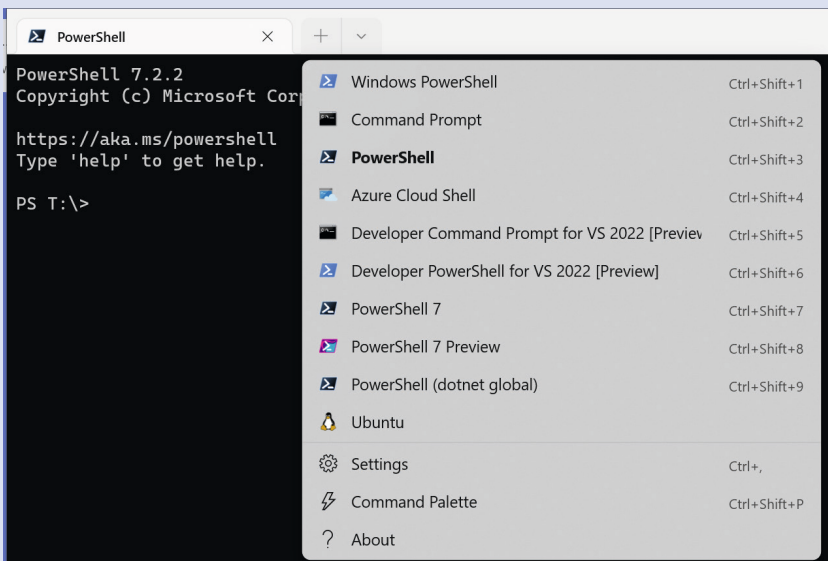
Für Windows wird die PowerShell 7 auf folgenden Wegen geliefert:

- Windows Store
- Installationsprogramm (MSI)
- ZIP-Datei
- .NET SDK CLI-Erweiterung



TIPP: Die PowerShell 7 kann man parallel zu den bisherigen Windows PowerShell-Installationen auf einem Windows-Rechner betreiben. Mit allen bisherigen PowerShell-Aktualisierungen war so ein Parallelbetrieb nicht möglich.

Zudem kann man beliebig viele Versionen und Varianten (Release und Preview) parallel installieren. Und man kann aus den verschiedenen oben genannten Quellen parallel installieren. Dies führt dann dazu, dass Sie im Startmenü und im Menü des „Windows Terminal“ viele verschiedene Einträge sehen (siehe Abbildung). **Bitte prüfen Sie genau, ob Sie damit klarkommen, bevor Sie so was machen.**



▼ Assets 26	
hashes.sha256	2.28 KB
powershell-7.2.1-1.rh.x86_64.rpm	65.9 MB
powershell-7.2.1-linux-alpine-x64.tar.gz	65.5 MB
powershell-7.2.1-linux-arm32.tar.gz	63.4 MB
powershell-7.2.1-linux-arm64.tar.gz	62 MB
powershell-7.2.1-linux-x64-fxdependent.tar.gz	21.9 MB
powershell-7.2.1-linux-x64.tar.gz	66.4 MB
powershell-7.2.1-osx-arm64.pkg	60.4 MB
powershell-7.2.1-osx-arm64.tar.gz	60.1 MB
powershell-7.2.1-osx-x64.pkg	64 MB
powershell-7.2.1-osx-x64.tar.gz	63.7 MB
PowerShell-7.2.1-win-arm32.zip	65.5 MB
PowerShell-7.2.1-win-arm64.zip	65.7 MB
PowerShell-7.2.1-win-fxdependent.zip	23.9 MB
PowerShell-7.2.1-win-fxdependentWinDesktop.zip	22.7 MB
PowerShell-7.2.1-win-x64.msi	101 MB
PowerShell-7.2.1-win-x64.zip	103 MB
PowerShell-7.2.1-win-x86.msi	92.3 MB
PowerShell-7.2.1-win-x86.zip	93.9 MB
powershell-lts-7.2.1-1.rh.x86_64.rpm	65.9 MB
powershell-lts-7.2.1-osx-arm64.pkg	60.4 MB
powershell-lts-7.2.1-osx-x64.pkg	64 MB
powershell-lts-7.2.1-1.deb_amd64.deb	66.5 MB
powershell-7.2.1-1.deb_amd64.deb	66.5 MB
Source code (zip)	
Source code (tar.gz)	

Bild 2.37 Download-Übersicht von PowerShell 7.2.1 auf GitHub [<https://github.com/PowerShell/powershell/releases>]

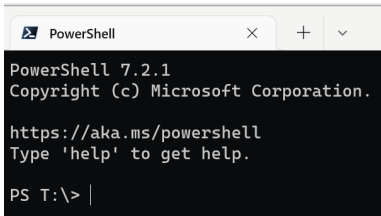
2.6.2.1 Nutzung der Archivdatei

Die niedrigste Installationshürde bietet die Nutzung der PowerShell 7 aus einem ZIP-Archiv, das Sie unter <https://github.com/PowerShell/powershell/releases> bekommen. Auch das ZIP-Archiv enthält alle benötigten Dateien von .NET 6.0, das die Basis von PowerShell 7.2 ist (man nennt dies in .NET eine „Self-Contained App“ – SCA).

Man entpackt das Archiv und startet dort dann einfach vom Windows Explorer oder von der klassischen Kommandozeile oder der klassischen PowerShell aus die ausführbare Datei *pwsh.exe*.



ACHTUNG: In der Windows PowerShell war der Name der Programmdatei `powershell.exe`. Microsoft hat den Namen gegenüber Windows PowerShell 5.1 bewusst geändert, um den Parallelbetrieb einfacher zu machen. Auf einem Windows mit Windows PowerShell 5.1 und PowerShell 6/7 startet man also per Eingabe von `powershell.exe` immer die Windows PowerShell und durch Eingabe von `pwsh.exe` immer die PowerShell 6 oder 7. Es können zudem mehrere Versionen der PowerShell 6 und 7 auf einem System koexistieren. So eine Versionskoexistenz war mit der Windows PowerShell nicht möglich.



```

PowerShell
PowerShell 7.2.1
Copyright (c) Microsoft Corporation.

https://aka.ms/powershell
Type 'help' to get help.

PS T:\> |

```

Bild 2.38

So meldet sich die PowerShell 7.2 auf Windows.

2.6.2.2 MSI-basierte Installation

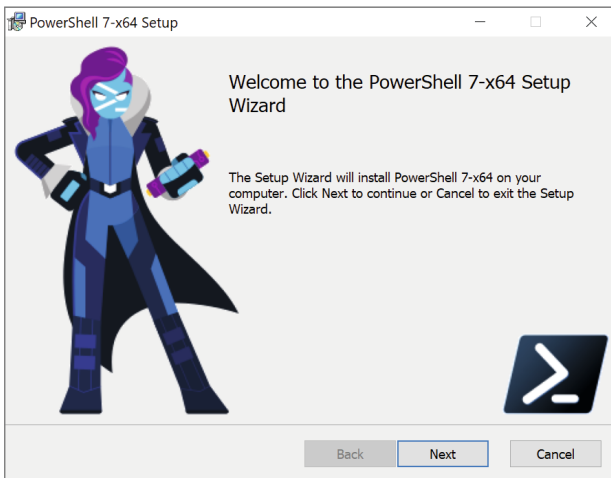
Auch das MSI-Paket für die PowerShell 7 bekommen Sie auf GitHub [<https://github.com/PowerShell/powershell/releases>].

Für die jeweils aktuelle Release-Version des MSI-Paketes gibt es eine Portalseite: <https://docs.microsoft.com/en-us/powershell/scripting/install/installing-powershell-on-windows> in der Dokumentation,

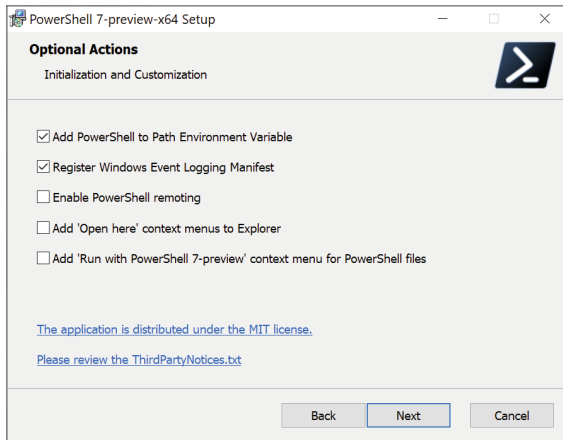
Das MSI-basierte Installationsprogramm installiert die PowerShell im Standard im Pfad `c:\Program Files\PowerShell\`. Dieser Pfad lässt sich bei der Installation ändern. In diesem Pfad wird ein Unterordner für die Version erstellt, z. B. „/7“ zwischen „/7-Preview“ für Preview-Versionen. In diesem Ordner befinden sich dann alle Dateien der PowerShell inklusive der benötigten Dateien von .NET.



TIPP: Die MSI-basierte Installation hat den Vorteil, dass es einige Installationsoptionen gibt (siehe folgende Abbildungen), um z. B. den Zielpfad der Installation automatisch zur Umgebungsvariablen `%Path%` zu ergänzen, sodass man nun `pwsh.exe` ohne Voranstellen eines Pfadnamens starten kann. Außerdem entsteht ein Eintrag im Startmenü.

**Bild 2.39**

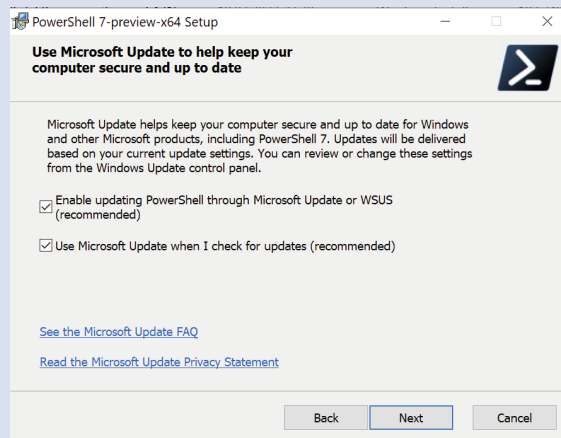
Installation der PowerShell 7 mit dem MSI-Paket

**Bild 2.40**

Optionen bei der Installation der PowerShell 7 mit dem MSI-Paket



HINWEIS: Per MSI-Paket installierte Versionen der PowerShell 7 werden seit Version 7.2 über „Microsoft Update“ (nicht „Windows Update“, denn PowerShell 7 ist kein Teil von Windows!) von Microsoft aktualisiert. Dies ist die Standardeinstellung im Setup, sie kann aber deaktiviert werden.



2.6.2.3 Installation per Microsoft Store

Die moderne PowerShell 7 lässt sich auch aus dem Microsoft Store installieren. Man sucht in der Store-App nach „PowerShell“.



HINWEIS: Im Windows Store gibt es die PowerShell 7 zweimal: einmal als aktuelle Release-Version und einmal als „Preview“ mit den neusten, noch nicht stabilen Features. Die beiden Varianten unterscheiden sich auch durch das Symbol (siehe folgende Abbildung).

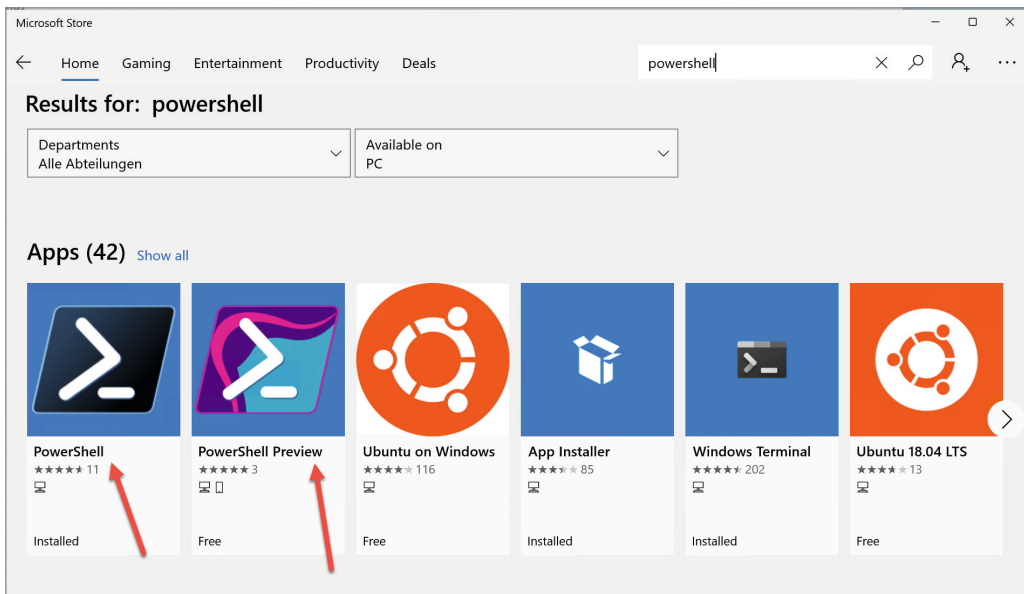


Bild 2.41 Die moderne PowerShell 7 im Microsoft Store (hier unter Windows 11)

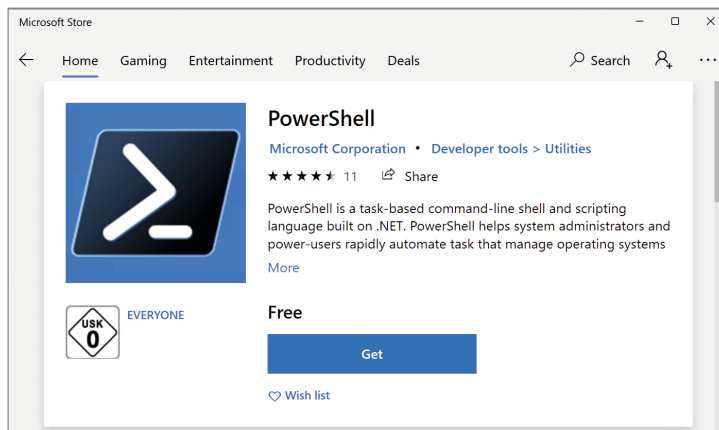


Bild 2.42 Installieren der Release-Version der PowerShell 7 aus dem Microsoft Store (hier unter Windows 11)

Durch die Installation landet die PowerShell (je nach Rechten) in diesem Verzeichnis:

```
C:\Users\xy\AppData\Local\Microsoft\WindowsApps\Microsoft.PowerShell_8wekyb3d8bbwe\pwsh.exe
```

oder

```
C:\Program Files\WindowsApps\Microsoft.PowerShell_8wekyb3d8bbwe\pwsh.exe
```

Aktualisierungen werden automatisch vom Microsoft Store angeboten.

2.6.2.4 Installation der PowerShell 7 als .NET SDK Global Tool

Wenn Sie das .NET Software Development Kit (.NET SDK) auf Ihrem System installiert haben, können Sie die PowerShell (ab Version 6.2) auch als ein sogenanntes .NET Core Global Tool über einen kurzen Kommandozeilenbefehl mithilfe der .NET CLI (dotnet.exe) installieren:

```
dotnet tool install --global powershell
```

Dies installiert die aktuelle RTM-Version der PowerShell. Zum Redaktionsschluss dieses Buchs ist dies die Version 7.2.1. Um die bestimmte Version der PowerShell zu installieren, müssen Sie Folgendes schreiben:

```
dotnet tool install --global PowerShell --version 7.2.1
```

Sofern Sie schon eine frühere Version installiert haben, müssen Sie so aktualisieren:

```
dotnet tool update --global PowerShell
```



HINWEIS: Da es .NET Core SDK für Linux, macOS und Windows gibt, funktioniert diese Installation auf allen drei Betriebssystemen. Durch die Installation als .NET Core Global Tool steht dann die PowerShell in Form des Kommandozeilenbefehls pwsh in allen Konsolenfenstern zur Verfügung. Es wird aber kein Menüeintrag (Symbol) erzeugt.

Ein .NET Global Tool wird von NuGet.org heruntergeladen. Dort finden Sie die PowerShell unter der Webadresse [www.nuget.org/packages/PowerShell/].

```
PS T:\> dotnet tool install --global PowerShell --version 7.1.0
You can invoke the tool using the following command: pwsh
Tool 'powershell' (version '7.1.0') was successfully installed.
PS T:\> dotnet tool install --global PowerShell
Tool 'powershell' is already installed.
PS T:\> dotnet tool update --global PowerShell
Tool 'powershell' was successfully updated from version '7.1.0' to version '7.2.1'.
PS T:\>
```

Bild 2.43 Installation der PowerShell als .NET Core Global Tool: erst Version 7.1.0, dann Update auf 7.2.1



HINWEIS: In den Docker Images des .NET SDK (ab Version 3.0), die Microsoft vorbereitet, ist die PowerShell bereits enthalten [https://hub.docker.com/_/microsoft-dotnet-core-sdk/].

2.6.2.5 Docker-Images

Es gibt auch vorgefertigte Docker-Images von Microsoft, die PowerShell 7 enthalten, siehe [https://hub.docker.com/_/microsoft-powershell/].

Wie Sie ein Docker-Image mit der PowerShell starten, erfahren Sie im Kapitel „Docker-Container“.

2.6.3 Test der PowerShell 7 auf Windows

Starten Sie die PowerShell 7 per Startmenüeintrag oder per Ausführung von *pwsh.exe*, z. B. innerhalb der *cmd.exe* oder *powershell.exe*.



HINWEIS: Man kann die PowerShell 7 aber nicht innerhalb der Windows PowerShell ISE starten.

Wie bei der klassischen Windows PowerShell gibt es auch unter PowerShell 7 Versions- und Systeminformationen über die eingebaute Variable `$psversiontable`.

Unter PowerShell Core hat Microsoft einige Anzeigen der Versionstabelle geändert. Am auffälligsten sind der Wert „Core“ statt „Desktop“ bei „PSEdition“ sowie die hinzugefügten Einträge „Platform“ und „OS“ für das aktuelle Betriebssystem. Platform hat die Werte Win32NT, macOSX und Unix. Die „CLRVersion“ wird hier nicht mehr angezeigt. Microsoft verbirgt hier leider, welche Version von .NET Core bei PowerShell Core mitgeliefert wird.

```

C:\Program Files\WindowsApps\Microsoft.PowerShell_7.2.2.0_x64__8wekyb3d8bbwe\pwsh.exe
PowerShell 7.2.2
Copyright (c) Microsoft Corporation.

https://aka.ms/powershell
Type 'help' to get help.

PS C:\Windows\System32> $psversiontable

Name                           Value
----                           -
PSVersion                       7.2.2
PSEdition                       Core
GitCommitId                     7.2.2
OS                               Microsoft Windows 10.0.22000
Platform                       Win32NT
PSCompatibleVersions            {1.0, 2.0, 3.0, 4.0...}
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1
WSManStackVersion              3.0

PS C:\Windows\System32>

```

Bild 2.44

Abruf der Versionsinformationen zur modernen PowerShell (hier installiert per Windows Store, unter Windows 11)

```

Z:\PowerShell-7.3.0-preview.2-win-x64\pwsh.exe
PS T:\> $psversiontable

Name                           Value
----                           -
PSVersion                       7.3.0-preview.2
PSEdition                       Core
GitCommitId                     7.3.0-preview.2
OS                               Microsoft Windows 10.0.19044
Platform                       Win32NT
PSCompatibleVersions            {1.0, 2.0, 3.0, 4.0...}
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1
WSManStackVersion              3.0

PS T:\>

```

Bild 2.45

Abruf der Versionsinformationen zur PowerShell Core 7.3.0 (unter Windows 10)

2.6.4 Installation und Test auf Ubuntu Linux

PowerShell 7 wird für Ubuntu als `.deb`-Datei ausgeliefert [github.com/PowerShell/PowerShell/releases], die sich über „Ubuntu Software“ (Ubuntu ab Version 16.04) installieren lässt.

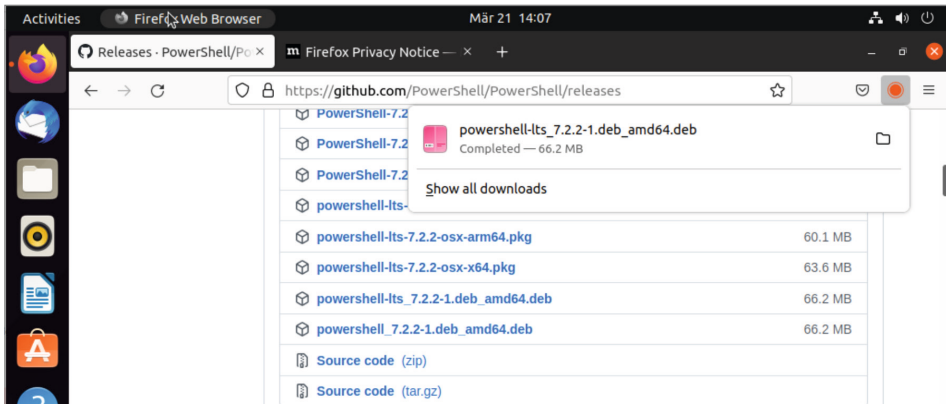


Bild 2.46 Download der Linux-Version für Ubuntu

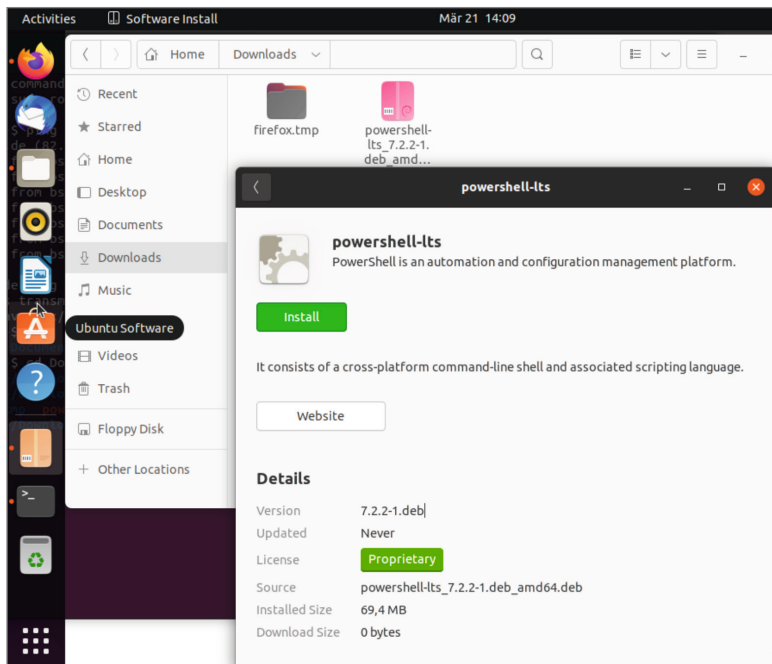


Bild 2.47 Installation der PowerShell 7 auf Ubuntu Linux (hier Version 21.04)

Alternativ geht dies nach einem manuellen Download per Kommandozeile:

```
sudo apt install /home/hs/Downloads/v7.2.1/powershell-lts_7.2.2-1.deb_amd64.deb
sudo apt-get install -f
```



HINWEIS: Lassen Sie sich nicht verunsichern, wenn Warnungen bezüglich Abhängigkeiten beim ersten Befehl erscheinen. Diese Probleme werden durch den zweiten Befehl geheilt.

Zum Start der PowerShell 7 gibt man im Terminal-Fenster `pwsh` (nicht `powershell` oder `powershell.exe!`) ein.

```

hs@D210: ~/Documents
hs@D210:~/Documents$ pwsh
PowerShell 7.2.2
Copyright (c) Microsoft Corporation.

https://aka.ms/powershell
Type 'help' to get help.

PS /home/hs/Documents> $PSVersionTable

Name                Value
----                -
PSVersion           7.2.2
PSEdition            Core
GitCommitId         7.2.2
OS                  Linux 5.13.0-35-generic #40-Ubuntu SMP Mon Mar 7 08:03:10 UTC 2022
Platform            Unix
PSCompatibleVersions {1.0, 2.0, 3.0, 4.0...}
PSRemotingProtocolVersion 2.3
SerializationVersion 1.1.0.1
WSManStackVersion   3.0

PS /home/hs/Documents>

```

Bild 2.48 Start der PowerShell 7 auf Ubuntu-Linux



TIPP: Sofern `pwsh` nicht gefunden wird, müssen Sie noch den Installationspfad in der Umgebungsvariablen `$PATH` in der Datei `/etc/environment` ergänzen und dann das System neu starten.

2.6.5 Installation der PowerShell 7 auf anderen Linux-Varianten

Für Debian und Kali gibt es ebenfalls eine `.deb`-Datei. Red Hat Enterprise Linux, OpenSUSE und CentOS werden durch `.rpm`-Dateien unterstützt. Für andere Linux-Distributionen gibt es eine Archiv-Datei (`.gz`).

2.6.6 Installation und Test auf macOS

Für die Installation auf macOS stellt Microsoft unter [github.com/PowerShell/PowerShell/releases] eine `.pkg`-Datei (Apple Software Package) oder alternativ ein Archiv (`.gz`) bereit.



HINWEIS: Seit PowerShell-Version 7.2 wird auch macOS mit Apple M1-Prozessor unterstützt! Die Datei für x64 heißt `powershell-7.x.y-osx-x64.pkg` und für M1 dann `powershell-7.x.y-osx-arm64.pkg`.

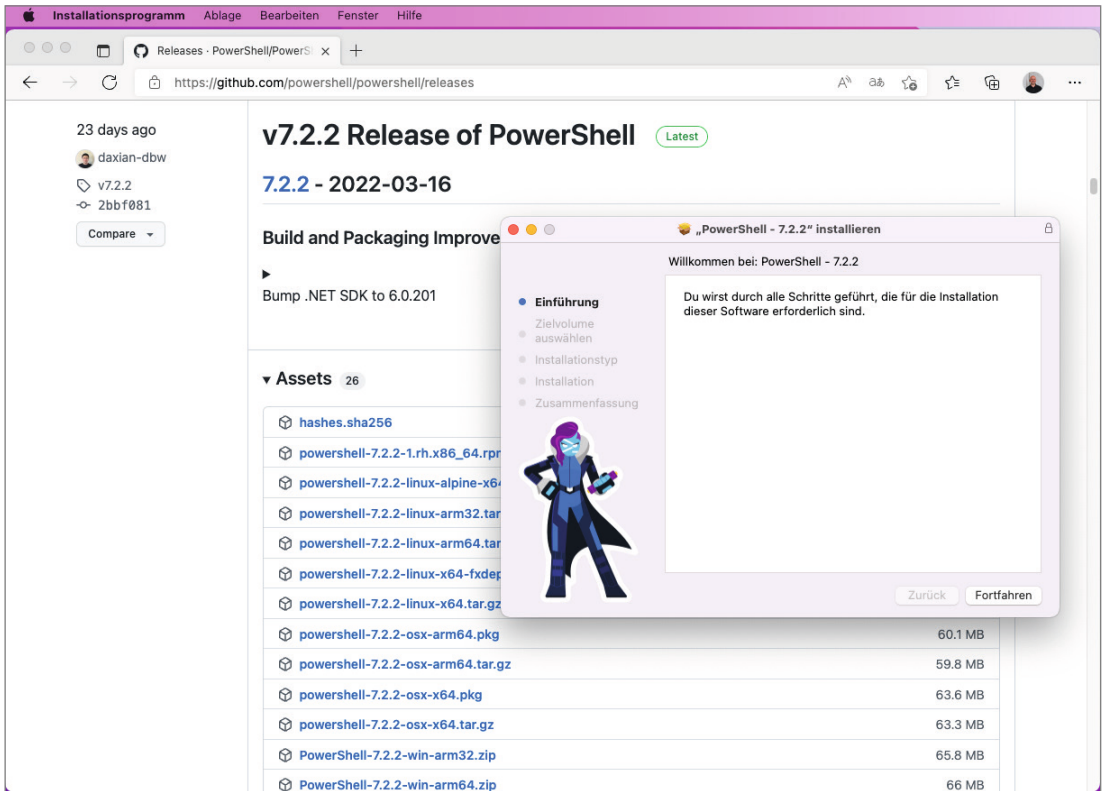


Bild 2.49 Download und Start der .pkg-Datei für macOS

Zum Start der PowerShell 7 auf OS/X gibt man im bash-basierten Terminal-Fenster pwsh (nicht „powershell“) ein.

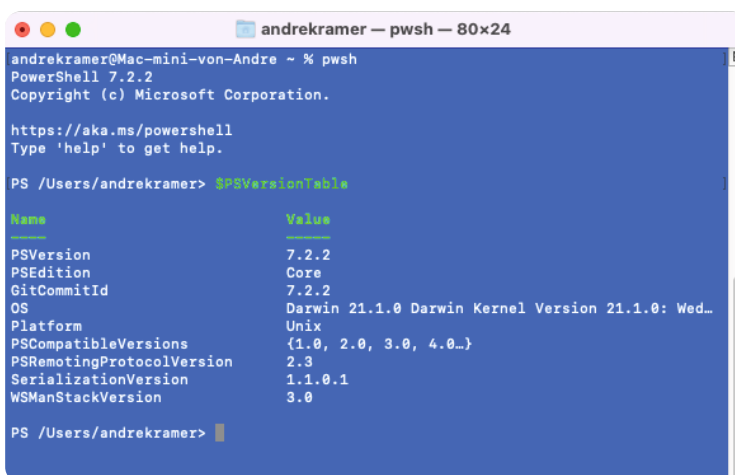


Bild 2.50 Start der PowerShell 7 auf macOS



TIPP: Microsoft verwendet auch unter macOS verschiedene Farben an der Konsole, die aber in einigen Fällen (z. B. Commandlet-Namen und Klassenmitgliedernamen) hell sind und auf einem weißen Hintergrund nicht genug Kontrast bieten. Stellen Sie daher für das macOS-Terminal-Fenster ein Farbschema mit einem dunkleren Hintergrund ein. Gut eignet sich das Farbschema „Ocean“. Sie ändern das Farbschema in dem Terminal-Fenster im Menü „Terminal/Einstellungen“ in der Registerkarte „Profile“.

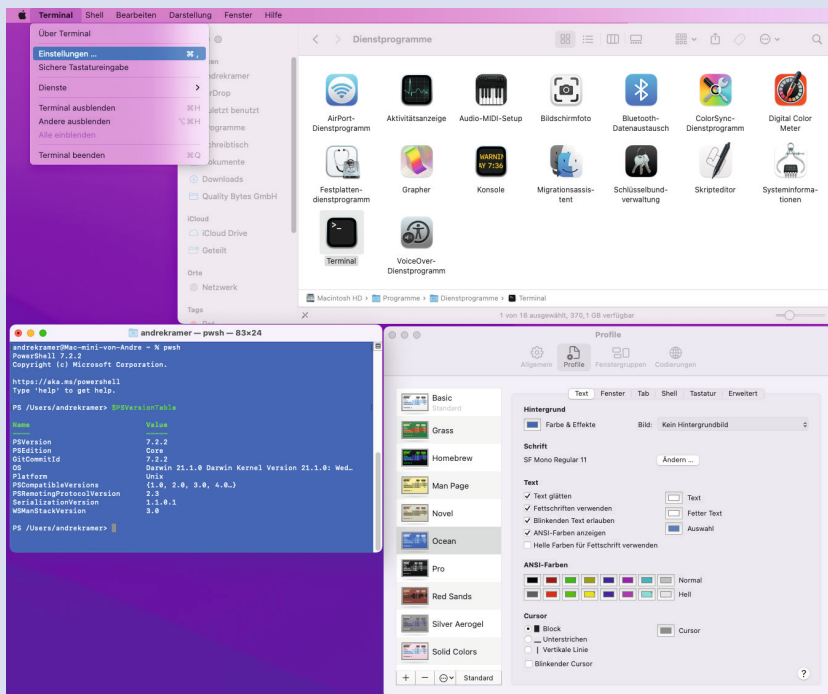


Bild 2.51 Einstellung des Farbschemas „Ocean“ für das macOS-Terminal-Fenster

2.6.7 Editor für PowerShell 7

PowerShell ISE ist nur mit Windows PowerShell verwendbar. Für PowerShell 6/7 müssten Sie den kostenfreien plattformneutralen Editor Visual Studio Code [<https://code.visualstudio.com>] zusammen mit der Erweiterung „PowerShell“ [<https://marketplace.visualstudio.com/items?itemName=ms-vscode.PowerShell>] einsetzen, der ebenfalls getrennt herunterzuladen und zu installieren ist. Er ist auf Windows, Linux und macOS installierbar.

Visual Studio Code ist zunächst auf die PowerShell 7 zu konfigurieren. Dazu sind einige Schritte notwendig, die Sie in Kapitel 13 „PowerShell 7 für Windows, Linux und macOS“ kennenlernen.

3

Einzelbefehle der PowerShell

Die PowerShell kennt folgende Arten von Einzelbefehlen:

- Commandlets (inkl. Funktionen)
- Aliase
- Ausdrücke
- Externe Befehle
- Dateinamen

■ 3.1 Commandlets

Ein „normaler“ PowerShell-Befehl heißt *Commandlet* (kurz: *Cmdlet*) oder *Funktion* (*Function*). Eine Funktion ist eine Möglichkeit, in der PowerShell selbst wieder einen Befehl zu erstellen, der funktioniert wie ein Commandlet. Da die Unterscheidung zwischen Commandlets und Funktionen aus Nutzersicht zum Teil akademischer Art ist, erfolgt hier zunächst keine Differenzierung: Das Kapitel spricht allgemein von Commandlets und meint damit auch Funktionen.

3.1.1 Aufbau eines Commandlets

Ein Commandlet besteht typischerweise aus drei Teilen:

- einem Verb,
- einem Substantiv und
- einer (optionalen) Parameterliste.

Verb und Substantiv werden durch einen Bindestrich „-“ voneinander getrennt, die optionalen Parameter durch Leerzeichen. Daraus ergibt sich der folgende Aufbau:

```
Verb-Substantiv [-Parameterliste]
```

Die Groß- und Kleinschreibung ist bei den Commandlet-Namen nicht relevant.

3.1.2 Aufruf von Commandlets

Ein einfaches Beispiel ohne Parameter lautet:

```
Get-Process
```

Dieser Befehl liefert eine Liste aller laufenden Prozesse im System.

Ein zweites Beispiel ist:

```
Get-ChildItem
```

Dieser Befehl liefert Unterelemente des aktuellen Standorts. Meist ist der aktuelle Standort ein Dateisystempfad. In der PowerShell kann der aktuelle Standort aber auch in der Registrierungsdatenbank, dem Active Directory und vielen anderen (persistenten) Speichern liegen.

Ein drittes Beispiel ist:

```
Get-Service
```

Dieser Befehl liefert alle Windows-Systemdienste. Dies ist ein Commandlet, das es nur unter Windows gibt, nicht in Linux und macOS.

Das waren alle Commandlets, die Informationen liefern. Commandlets, die Aktionen ausführen (z. B. Prozesse beenden, Dateien löschen, Dienste anhalten), kommen in der Regel nicht ohne Parameter aus, da sie sonst ja global auf alle Ressourcen eines Typs angewendet würden, z. B. alle Prozesse beenden, alle Dateien löschen und alle Dienste anhalten. Das ist absichtlich nicht so implementiert. Solche Befehle, die Parameter erfordern, kommen werden im nächsten Unterkapitel behandelt.



TIPP: Die Tabulatorvervollständigung in der PowerShell-Konsole funktioniert bei Commandlets, wenn man das Verb und den Strich bereits eingegeben hat, z. B. `Export-Tab`. Auch Platzhalter kann man dabei verwenden. Die Eingabe `Get-?e*` liefert `Get-Help` `Get-Member` `Get-Service`. Andere Editoren wie das ISE bieten auch IntelliSense-Eingabeunterstützung für Commandlet-Namen an.



TIPP: Commandlets, die mit dem Wort `Get-` beginnen, kann man abkürzen, indem man das `Get-` weglässt; also z. B. einfach `Service` statt `Get-Service` schreibt. Ob man dies so erlauben möchte, sollte das Unternehmen als Richtlinie festlegen.

3.1.3 Commandlet-Parameter

Durch Angabe eines Parameters können die Commandlets Informationen für die Befehlsausführung erhalten, z. B. ist bei `Get-Process` ein Filtern über den Prozessnamen möglich.

Durch

```
Get-Process i*
```

werden nur diejenigen Prozesse angezeigt, deren Name auf das angegebene Muster (Name beginnt mit dem Buchstaben „i“) zutrifft:

Ein weiteres Beispiel für einen Befehl mit Parameter ist:

```
Get-ChildItem w:\daten
```

Get-ChildItem listet alle Unterobjekte des angegebenen Dateisystempfads (*w:\daten*) auf, also alle Dateien und Ordner unterhalb dieses Dateordners.

Ein drittes Beispiel ist:

```
Stop-Service BITS
```

Dieser Befehl führt eine Aktion aus: Der Windows-Hintergrundübertragungsdienst (Background Intelligent Transfer Service – BITS) wird angehalten.

Ein viertes Beispiel ist:

```
Remove-Item w:\temp\*.log
```

Dieser Befehl löscht alle Dateien mit der Dateinamenserweiterung „log“ aus dem Ordner *w:\temp*.

Parameter werden als Zeichenkette aufgefasst – auch wenn sie nicht explizit in Anführungszeichen stehen. Die Anführungszeichen sind optional. Man muss Anführungszeichen um den Parameterwert nur dann verwenden, wenn Leerzeichen vorkommen, denn das Leerzeichen dient als Trennzeichen zwischen Parametern:

```
Get-ChildItem "c:\Program Files"
```

Einige Commandlets erlauben für einen Parameter nicht nur einen einzelnen Wert, sondern auch eine Menge von Werten. Die Einzelwerte sind dann durch ein Komma zu trennen.

Beispiel: Prozesse, die mit dem Buchstaben a beginnen oder enden oder mit x beginnen oder enden

```
Get-Process "a*", "*a", "x*", "*x"
```

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
629	29	21672	26484	1,27	1200	4	ApplicationFrameHost
140	9	1420	1928	0,05	4276	0	armsvc
137	8	1484	1976	0,11	3192	0	atiesrxx
1259	68	75644	11056	32.389,55	4356	0	AVKProxy
896	38	155380	155808	4.404,88	3096	0	AVKwCt1x64
993	90	210684	244996	40,86	13128	4	firefox

Bild 3.1 Get-Process mit einer Liste von Namen

Commandlets haben aber in der Regel nicht nur einen, sondern zahlreiche Parameter, die durch Position oder einen Parameternamen voneinander unterschieden werden. Ohne die Verwendung von Parameternamen werden vordefinierte Standardattribute belegt, d.h., die Reihenfolge ist entscheidend.

Beispiel: Auflisten von Dateien in einem Dateisystempfad, die eine bestimmte Datennamenserweiterung besitzen. Dies erfüllt der Befehl:

```
Get-ChildItem w:\temp *.doc
```

Wenn ein Commandlet mehrere Parameter besitzt, ist die Reihenfolge der Parameter entscheidend oder der Nutzer muss die Namen der Parameter mit angeben. Bei der Angabe von Parameternamen kann man die Reihenfolge der Parameter ändern:

```
Get-ChildItem -Filter *.doc -Path w:\temp
```

Alle folgenden Befehle sind daher gleichbedeutend:

```
Get-ChildItem w:\temp *.doc
Get-ChildItem -Path w:\temp -Filter *.doc
Get-ChildItem -Filter *.doc -Path w:\temp
```

Hingegen ist Folgendes falsch und funktioniert nicht wie gewünscht, weil die Parameter nicht benannt sind und die Reihenfolge falsch ist:

```
Get-ChildItem *.doc w:\temp
```

Diesen Versuch beantwortet die PowerShell mit einer Fehlermeldung („Das zweite Pfadfragment darf kein Laufwerk oder UNC-Name sein.“) in roter Schrift (siehe nächste Bildschirmabbildung).

Bild 3.2 Fehlermeldung bei falscher Parameterreihenfolge. Die Fehlermeldungen in der modernen PowerShell sind oft prägnanter

Schalter-Parameter (engl. Switch) sind Parameter, die keinen Wert haben. Durch die Verwendung des Parameternamens wird die Funktion aktiviert, z.B. das rekursive Durchlaufen durch einen Dateisystembaum mit `-recurse`:

```
Get-ChildItem w:\demo\powershell -recurse
```



TIPP: Wenn man einen Schalter deaktivieren möchte, weil er im Standard aktiv ist oder weil man sehr explizit darauf hinweisen möchte, dass er nicht aktiv sein soll, kann man `$false` mit Doppelpunkt getrennt angeben, z. B.

```
Get-ChildItem w:\demo\powershell -recurse:$false
```

Parameter können berechnet, d.h. aus Teilzeichenketten zusammengesetzt sein, die mit einem Pluszeichen verbunden werden. (Dies macht insbesondere Sinn in Zusammenhang mit Variablen, die aber erst später in diesem Buch eingeführt werden.)

Der folgende Ausdruck führt jedoch nicht zum gewünschten Ergebnis, da auch hier das Trennzeichen vor und nach dem + ein Parametertrenner ist.

```
Get-ChildItem "c:\" + "Windows" *.dll -Recurse
```

Auch ohne die beiden Leerzeichen vor und nach dem + geht es nicht. In diesem Fall muss man durch eine runde Klammer dafür sorgen, dass die Berechnung erst ausgeführt wird:

```
Get-ChildItem ("c:\" + "Windows") *.dll -Recurse
```

Es folgt dazu noch ein Beispiel, bei dem Zahlen berechnet werden. Der folgende Befehl liefert den Prozess mit der ID 2900:

```
Get-Process -id (2800+100)  
Get-Service -exclude "[k-z]*"
```

zeigt nur diejenigen Systemdienste an, deren Name nicht mit den Buchstaben „k“ bis „z“ beginnt.

Auch mehrere Parameter können der Einschränkung dienen. Der folgende Befehl liefert nur die Benutzereinträge aus einem bestimmten Active-Directory-Pfad. (Das Beispiel setzt die Installation der PSCX voraus.)

```
Get-ADObject -dis "LDAP://Server123/ou=agents,DC=FBI,DC=net" -class user
```



TIPP: Tabulatorvervollständigung klappt auch bei Parametern. Versuchen Sie einmal folgende Eingabe an der PowerShell-Konsole: `Get-ChildItem -Tab`

3.1.4 Platzhalter bei den Parameterwerten

An vielen Stellen sind Platzhalter bei den Parameterwerten erlaubt. Ein Stern steht für beliebig viele Zeichen. Eine Liste aller Prozesse, die mit einem „i“ anfangen, erhält man so:

```
Get-Process i*
```

Eine Liste aller Prozesse, die mit einem „i“ anfangen und auf „ore“ enden, erhält man so:

```
Get-Process i*ore
```

Ein Fragezeichen steht für genau ein beliebiges Zeichen. Eine Liste aller Prozesse, die mit einem „v“ anfangen, gefolgt von einem einzigen beliebigen Zeichen und auf „mms“ enden, erhält man so:

```
Get-Process v?mms
```

Eine eckige Klammer steht für genau ein Zeichen aus einer Auswahl. Alle Prozesse, die mit s oder t anfangen, erhält man so:

```
Get-Process [st]*
```

Alle Prozesse, die mit s oder t anfangen und bei denen dann ein v oder f folgt, erhält man so:

```
Get-Process [st][vf]*
```

```
PS C:\> get-process [st][vf]*
```

Handles	NPM(K)	PM(K)	WS(K)	UM(M)	CPU(s)	Id	ProcessName
1439	162	29196	44776	581		484	svchost
284	14	5448	5772	308		568	svchost
2958	107	315244	66196	608		576	svchost
417	16	5728	10464	53		892	svchost
748	19	14500	17492	65		948	svchost
747	45	23100	27612	152		1028	svchost
819	35	42244	40380	123		1084	svchost
1403	77	83400	82220	1788		1324	svchost
462	68	36844	32792	399		1508	svchost
155	13	5144	5536	74		2224	svchost
261	14	6056	5832	50		2356	svchost
152	12	2560	8480	48		3340	svchost
177	16	7560	8372	54		3640	svchost
181	8	1656	2068	23		5196	svchost
468	28	7308	11324	87		5628	svchost
614	45	42160	27224	397	2,73	7428	TfsCommandRunnerSvc
608	32	26312	25776	232	82,68	1496	TfsComProviderSvc
530	30	22540	21616	218	1,14	10700	TfsComProviderSvc

Bild 3.3 Beispiele für das Ergebnis des obigen Befehls mit Platzhaltern

3.1.5 Abkürzungen für Parameter

Parameternamen dürfen abgekürzt werden, solange sie noch eindeutig sind.

Statt

```
Get-ChildItem -Filter *.txt -Path w:\temp
```

darf man schreiben

```
Get-ChildItem -Fi *.txt -Pa w:\temp
```

Nicht möglich ist in diesem Fall die Reduzierung auf einen Buchstaben:

```
Get-ChildItem -F *.txt -P w:\temp
```

Darauf reagiert die PowerShell mit der Fehlermeldung

```
Der Parameter kann nicht verarbeitet werden, da der Parametername "F" nicht eindeutig ist. Mögliche Übereinstimmungen: -Filter -Force
```

und

```
Der Parameter kann nicht verarbeitet werden, da der Parametername "P" nicht eindeutig ist. Mögliche Übereinstimmungen: -Path -PipelineVariable
```



ACHTUNG: Bitte beachten Sie aber, dass abgekürzte Parameter auch eine Gefahr bedeuten: Was heute eine eindeutige Abkürzung ist, könnte in einer zukünftigen Version doppeldeutig sein, wenn Microsoft weitere Parameter zu einem Commandlet ergänzt. Tatsächlich gab es in der Vergangenheit auch schon kuriose Fälle, dass die Abkürzungen in verschiedenen Windows-Installationen verschieden interpretiert wurden, wie die nachstehenden Bildschirmabbildungen beweisen. Zudem sind abgekürzte Parameter nicht so „sprechend“ wie die Langparameter. Für abgekürzte Parameter spricht aber, dass Befehle dadurch kürzer und übersichtlicher werden. Trotz allem werden Sie auch abgekürzte Parameter in diesem Buch finden, da der Autor dieses Buchs eben auch ein Mensch ist, der sich im Alltag manche Tipparbeit gerne erspart.

```
PS C:\> get-process | ft -p id,name,workingset
```

Id	Name	WorkingSet
2936	avp	19230720
6336	avp	25407488
4180	conhost	4145152
9626	conhost	5565824

Bild 3.4 Verhalten auf Windows Server 2008 R2 und Windows 8 mit PowerShell 3.0

```
PS C:\> Get-Process | ft -p id,name,WorkingSet
Format-Table : Parameter cannot be processed because the parameter name 'p' is ambiguous. Possible matches include:
-Property -PipelineVariable.
At line:1 char:18
+ Get-Process | ft -p id,name,WorkingSet
+ ~~~~~
+ CategoryInfo          : InvalidArgument: (:) [Format-Table], ParameterBindingException
+ FullyQualifiedErrorId : AmbiguousParameter,Microsoft.PowerShell.Commands.FormatTableCommand
```

Bild 3.5 Verhalten auf Windows 7 und Windows 8.1 mit PowerShell 4.0

```
PS C:\> Get-Process | FT -p id,processn*,working*
```

Id	ProcessName	WorkingSet	WorkingSet64
9148	ApplicationFrameHost	24567808	24567808
2712	armsvc	6201344	6201344
1624	atioclx	0703536	0703536

Bild 3.6 Verhalten auf allen Windows 10/11 und Windows Server 2019/2022 mit PowerShell 5.1

3.1.6 Allgemeine Parameter (Common Parameters)

Es gibt einige Parameter, die in vielen (aber nicht allen) Commandlets vorkommen. Es folgt eine vollständige Liste dieser Parameter. Eine genauere Beschreibung folgt aber aus didaktischen Gründen an geeigneter Stelle im Buch, da viele allgemeine Parameter mit dem Pipelining und der Fehlerbehandlung zu tun haben, die erst in späteren Kapiteln besprochen wird.

- **-Force:** Eine Aktion wird erzwungen, z.B. eine Datei wird mit Remove-Item gelöscht, obwohl die Datei einen Schreibschutz gesetzt hat. Ein weiteres Beispiel: Remove-SmbShare fragt immer vor dem Löschen nach, wenn -force nicht gesetzt ist.
- **-What if („Was wäre wenn“):** Die Aktion wird nicht ausgeführt, es wird nur ausgegeben, was passieren würde, wenn man die Aktion ausführt. Das ist z.B. in einem Befehl mit Platzhaltern wie dem Folgenden sinnvoll, damit man weiß, welche Dienste nun gestoppt würden:

```
Get-Service | Where {$_.servicename -like "*SQL*"}
| Foreach { stop-service $_.servicename -whatif }
```

oder

```
Stop-Service -name "*sql*" -whatif
```

```
PS T:\> Stop-Service -name "*sql*" -whatif
What if: Performing the operation "Stop-Service" on target "SQL Server (MSSQLSERVER) (MSSQLSERVER)".
What if: Performing the operation "Stop-Service" on target "SQL Server Browser (SQLBrowser)".
What if: Performing the operation "Stop-Service" on target "SQL Server Agent (MSSQLSERVER) (SQLSERVERAGENT)".
What if: Performing the operation "Stop-Service" on target "SQL Server CEIP service (MSSQLSERVER) (SQLELEMETRY)".
What if: Performing the operation "Stop-Service" on target "SQL Server VSS Writer (SQLWriter)".
PS T:\> |
```

Bild 3.7 Operationen mit Platzhaltern können schlimme Konsequenzen haben – whatif zeigt, welche Dienste betroffen wären.

- **-Confirm:** Der Benutzer erhält eine Nachfrage für jede Änderungsaktion (siehe Bildschirmabbildung), z. B.

```
get-service | where {$_.servicename -like "A*"}
| foreach { stop-service $_.servicename -confirm }.
```

Innerhalb der Nachfrage kann der Benutzer in einen Suspend-Modus gehen, in dem er andere Befehle eingeben kann, z. B. um zu prüfen, ob er nun ja oder nein antworten will. Der Suspend-Modus wird mit drei Pfeilen >>> angezeigt und ist durch exit zu verlassen (siehe Bildschirmabbildung).

```
PS T:\> Remove-Item T:\daten\webserver.txt -Confirm

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove File" on target "T:\daten\webserver.txt".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"): S
PS T:\>>> Get-Content T:\daten\webserver.txt
www.dotnetframework.de;192.168.1.52;91;c:\daten\Websites\www.dotnetframework.de
www.Windows-Scripting.com;192.168.1.52;90;c:\daten\Websites\www.windows-scripting.de
www.powershell-doktor.de;192.168.1.52;98;c:\daten\Websites\www.powershell-doktor.de
www.aspnetdev.de;192.168.1.52;98;c:\daten\Websites\www.aspnetdev.de
www.dotnet-lexikon.de;192.168.1.52;98;c:\daten\Websites\www.dotnet-lexikon.de
www.windows-scripting.com;192.168.1.52;90;c:\daten\Websites\www.windows-scripting.com
www.powershell-Schulungen.de;192.168.1.52;98;c:\daten\Websites\www.powershell.de
PS T:\>>> exit

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove File" on target "T:\daten\webserver.txt".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"): Y
Remove-Item: You do not have sufficient access rights to perform this operation or the item is hidden, system, or read only.
PS T:\>
```

Bild 3.8 Confirm und Suspend

- `-ErrorAction` (abgekürzt `-ea`) und `-WarningAction` (`-wa`): Festlegung, wie ein Skript sich verhalten soll, wenn es auf einen Fehler trifft. Dieser Parameter wird im Abschnitt 7.19 „Fehlerbehandlung“ näher erklärt.
- `-Verbose`: Das Commandlet liefert eine detaillierte Bildschirmausgabe.
- `-Debug`: Das Commandlet liefert eine sehr detaillierte Bildschirmausgabe.
- `-OutVariable`: Das Commandlet liefert alle Objekte nicht nur in die Pipeline, sondern legt sie zusätzlich auch in einer Variablen ab.
- `-PipelineVariable`: Das Commandlet liefert das aktuelle Objekt nicht nur in die Pipeline, sondern legt es zusätzlich auch in einer Variablen ab.
- `-ErrorAction`: Festlegung, wie sich das Commandlet bei Fehlern verhält
- `-ErrorVariable`: speichert eine Fehlermeldung des Commandlets zusätzlich in einer Variablen
- `-WarningAction`: Festlegung, wie sich das Commandlet bei Warnungen verhält. Der Standard ist „continue“, was bedeutet, dass die Meldung ausgegeben wird. Mit „silently-continue“ kann die Ausgabe unterdrückt werden. Mit „stop“ wird ein Befehl nach der Warnung abgebrochen. Mit „inquire“ fragt die PowerShell nach, wie fortzufahren ist.
- `-WarningVariable`: speichert eine Warnung des Commandlets zusätzlich in einer Variablen
- `-OutBuffer`: stellt ein, dass die angegebene Anzahl von Objekten in der Pipeline gepuffert werden sollen, bevor sie in der Pipeline weitergegeben werden. Normalerweise werden alle Objekte sofort in der Pipeline weitergegeben.



ACHTUNG: Leider beachten nicht alle Commandlets alle allgemeinen Parameter. Erschwerend kommt hinzu, dass sie keine Fehlermeldung liefern, sondern den Parameter einfach ignorieren. Ein Beispiel ist `New-SmbShare` zum Anlegen einer Dateisystemfreigabe. Die folgenden Befehle werden trotz `-whatif` bzw. `-confirm` sofort und ohne Nachfrage ausgeführt.

```
New-SmbShare -Name Temp -Path w:\temp -WhatIf
New-SmbShare -Name Temp -Path w:\temp -confirm
```

Sie werden sich fragen, warum dies so ist. Das Fehlverhalten liegt hier bei dem Entwickler des Commandlets. Jeder Commandlet-Entwickler muss daran denken, die allgemeinen Parameter zu behandeln. Denkt er nicht daran, sind die Nutzer seines Commandlets die Leidtragenden. Es wäre natürlich besser, wenn Microsoft mit seiner Programmierschnittstelle für Commandlets die Commandlet-Entwickler zwingen würde, die Parameter zu behandeln oder zumindest eine Fehlermeldung zu liefern, wenn man die Parameter einsetzt. Leider hat Microsoft diesen Vorschlag bisher nicht aufgegriffen – auch wenn Microsoft ja sehr offensichtlich nicht mal seine eigenen Commandlet-Entwickler im Griff hat.



ACHTUNG: Leider gibt es bei den PowerShell-Commandlets, die gravierende Aktionen ausführen, einige Unterschiede im Grundverhalten und in der Verwendung der obigen Commandlets. Einige Commandlets führen im Standard die Aktion aus (z. B. Remove-Item). Andere Commandlets (z. B. Remove-ADUser und Remove-SmbShare) fragen immer nach vor dem Löschen. Das ist bei automatisierten Skripten natürlich unsinnig und daher gibt es auch eine Möglichkeit, diesen Commandlets das abzugewöhnen. Diese sieht jedoch oftmals verschieden aus. Bei Remove-ADUser muss man `-confirm:$false` als Parameter angeben; bei Remove-SmbShare ist es hingegen ein `-force`. Schade, dass Microsoft hier nicht einheitlich sein konnte.

Standardvorgaben für allgemeine Parameter

In den eingebauten Variablen `$WhatIfPreference`, `$VerbosePreference`, `$DebugPreference`, `$ConfirmPreference` und `$ErrorActionPreference` ist festgelegt, wie sich die PowerShell im Standard in Bezug auf `-WhatIf`, `-Verbose`, `-Debug`, `-Confirm` und `-ErrorAction` verhält. Dort ist hinterlegt:

- `WhatIfPreference: False`
- `VerbosePreference: SilentlyContinue`
- `DebugPreference: SilentlyContinue`
- `ErrorActionPreference: Continue`
- `ConfirmPreference: High`

Variablen werden erst später in diesem Buch (Kapitel 7 „PowerShell-Skriptsprache“) behandelt. An dieser Stelle soll aber schon mit einem Beispiel gezeigt werden, wie man `$WhatIfPreference` auf `$true` setzt und damit erreicht, dass alle Commandlets, die `-whatif` unterstützen, nun nur noch sagen, was sie machen würden – zumindest solange man nicht explizit `-whatif:$false` als Parameter angibt.

Ausgabe der aktuellen Einstellung von `$WhatIfPreference`. Sollte `$false` sein

```
Write-host "WhatIfPreference = $WhatIfPreference" -ForegroundColor Yellow
```

Neustart des Dienstes wird tatsächlich ausgeführt

```
Restart-Service BITS -Verbose
```

Nun `$WhatIfPreference` aktivieren

```
$WhatIfPreference = $true
```

Ausgabe der aktuellen Einstellung von `$WhatIfPreference`. Sollte `$true` sein

```
Write-host "WhatIfPreference = $WhatIfPreference" -ForegroundColor Yellow
```

Neustart des Dienstes wird NICHT ausgeführt

```
Restart-Service BITS -Verbose
```

Neustart des Dienstes wird tatsächlich ausgeführt

```
Restart-Service BITS -WhatIf:$false -Verbose
```

Nun \$WhatIfPreference zurücksetzen

```
$WhatIfPreference = $false
```

```
Administrator: Windows PowerShell
PS T:\> $WhatIfPreference = $false
PS T:\> Restart-Service BITS -Verbose
VERBOSE: Performing the operation "Restart-Service" on target "Background Intelligent Transfer Service (BITS)".
PS T:\> $WhatIfPreference = $true
PS T:\> Restart-Service BITS -Verbose
What if: Performing the operation "Restart-Service" on target "Background Intelligent Transfer Service (BITS)".
PS T:\> Restart-Service BITS -WhatIf:$false -Verbose
VERBOSE: Performing the operation "Restart-Service" on target "Background Intelligent Transfer Service (BITS)".
PS T:\>
```

Bild 3.9 Verwendung von \$WhatIfPreference

3.17 Dynamische Parameter

Einige Commandlets besitzen die Fähigkeit, verschiedene Parameter abhängig von bereits eingegebenen Parametern anzubieten.

```
PS T:\> dir C:\windows -
```

- Attributes
- Directory [switch] Directory
- File
- Hidden
- ReadOnly
- System
- Verbose
- Debug
- ErrorAction

Bild 3.10
Get-ChildItem (alias dir) in Verbindung mit einem Dateisystempfad (hier mit IntelliSense in PowerShell ISE)

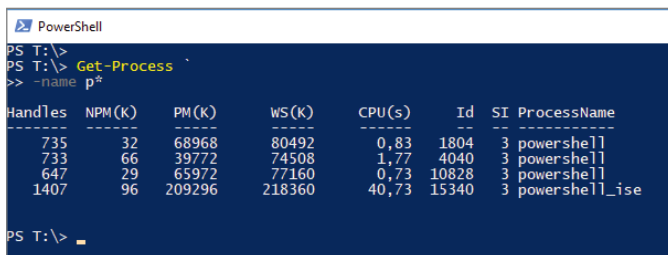
```
PS T:\> dir Cert:\CurrentUser\my -
```

- CodeSigningCert [string] Filter
- DocumentEncryptionCert
- SSLServerAuthentication
- DnsName
- Eku
- ExpiringInDays
- Verbose
- Debug
- ErrorAction

Bild 3.11
Get-ChildItem (alias dir) in Verbindung mit einem Pfad im Zertifikatsspeicher (hier mit IntelliSense in PowerShell ISE)

3.1.8 Zeilenumbrüche

Wenn man die Eingabe-Taste drückt, wird ein PowerShell-Befehl direkt ausgeführt. Möchte man einen Befehl über mehrere Zeilen erstrecken, muss man die unvollständige Zeile mit dem Gravis (Accent Grave) [`] beenden.



```

PowerShell
PS T:\>
PS T:\> Get-Process
>> -name p`
-----
Handles  NPM(K)  PM(K)  WS(K)  CPU(s)  Id  SI  ProcessName
-----
       735     32  68968   80492    0,83  1804  3  powershell
       733     66  39772   74508    1,77  4040  3  powershell
       647     29  65972   77160    0,73  10828  3  powershell
       1407    96  209296  218360   40,73  15340  3  powershell_ise
PS T:\>

```

Bild 3.12
Einsatz des Gravis für
Zeilenumbrüche im Befehl



TIPP: In der PowerShell-Konsole kann man SHIFT + EINGABE drücken. Auch dann zeigt die Standardkonsole mit >>> an, dass weitere Eingaben erwartet werden. Allerdings wird dann ein eigenständiger Befehl erwartet und nicht der vorherige fortgesetzt!

3.1.9 PowerShell-Module

Schon seit PowerShell 2.0 sind die Commandlets und Funktionen in Modulen organisiert. Während der Benutzer in PowerShell 2.0 ein Modul noch explizit mit Import-Module aktivieren musste, bevor man die Befehle aus dem Modul nutzen konnte, erledigt dies die PowerShell seit Version 3.0 bei Bedarf automatisch (Module Auto-Loading). Sowohl Konsole als auch ISE zeigen alle verfügbaren Commandlets und Funktionen aller vorhandenen Module in der Vorschlagsliste und beim Aufruf von Get-Command bereits an. Der eigentliche Import des Moduls erfolgt dann beim ersten Aufruf eines Befehls aus einem Modul.

In der PowerShell sind auch alle Kernbefehle der PowerShell in Modulen organisiert, diese zeigt die folgende Tabelle.

Tabelle 3.1 Die vier wichtigsten Module der PowerShell mit Beispielen für Commandlets in diesem Modul

Modul	Beispiele für Commandlets in diesem Modul
Microsoft.PowerShell.Diagnostics	Get-WinEvent, Get-Counter, Import-Counter, Export-Counter ...
Microsoft.PowerShell.Management	Add-Content, Clear-Content, Clear-ItemProperty, Join-Path, Get-Process, Get-Service ...
Microsoft.PowerShell.Security	Get-Acl, Set-Acl, Get-PfxCertificate, Get-Credential ...
Microsoft.PowerShell.Utility	Format-List, Format-Custom, Format-Table, Format-Wide, Where-Object ...

3.1.10 Prozessmodell

Die PowerShell erzeugt beim Start einen einzigen Prozess. In diesem Prozess laufen alle ausgeführten Commandlets. Dies ist ein Unterschied zum DOS-ähnlichen Windows-Kommandozeilenfenster, bei dem die ausführbaren Dateien (.exe) in eigenen Prozessen laufen. Es ist in der PowerShell aber auch möglich, Hintergrundaufgaben auszuführen (siehe Kapitel 25 „Hintergrundaufträge“).



TIPP: Mit **STRG+C** kann man einen laufenden Befehl in der PowerShell abbrechen.

3.1.11 Aufruf von Commandlets aus anderen Prozessen heraus

PowerShell-Commandlets kann man aus einem beliebigen Prozess heraus aufrufen, indem man powershell.exe aufruft und das Commandlet als Parameter übergibt.

Beispiel: powershell.exe "get-service a*"

Damit die Parameter des Commandlets dem Commandlet und nicht powershell.exe zugeordnet werden, muss man das Commandlet und seine Parameter in Anführungszeichen setzen. Falls der PowerShell-Commandlet-Parameter seinerseits Anführungszeichen erfordert, muss man dafür einfache Anführungszeichen verwenden:

```
powershell.exe "get-service 'a*'"
```

```

Command Prompt

C:\Users\hs>powershell.exe "get-service 'A*'"

Status  Name                DisplayName
-----  -
Stopped AJRouter            AllJoyn Router Service
Stopped ALG              Application Layer Gateway Service
Stopped AppIDSvc       Application Identity
Running  Appinfo             Application Information
Stopped AppMgmt        Application Management
Stopped AppReadiness   App Readiness
Stopped AppVClient     Microsoft App-V Client
Stopped AppXSvc        AppX Deployment Service (AppXSVC)
Running  AudioEndpointBu...  Windows Audio Endpoint Builder
Running  Audiosrv            Windows Audio
Stopped AxInstSV       ActiveX Installer (AxInstSV)

C:\Users\hs>

```

Bild 3.13

Aufruf eines PowerShell-Commandlets aus einer klassischen Windows-Konsole (CMD) heraus mit powershell.exe (im Bild in Windows 10)

3.1.12 Namenskonventionen

Man beachte, dass bei den Commandlets das Substantiv im Singular steht, auch wenn eine Menge von Objekten abgerufen wird. Das Ergebnis muss nicht immer eine Objektmenge sein. Beispielsweise liefert

```
Get-Location
```

nur ein Objekt mit dem aktuellen Pfad. Mit

```
Set-Location c:\windows
```

wechselt man den aktuellen Pfad. Diese Operation liefert gar kein Ergebnis.



HINWEIS: Die Groß- und Kleinschreibung der Commandlet-Namen und der Parameternamen ist irrelevant.

Gemäß der PowerShell-Konventionen soll es nur eine begrenzte Menge wiederkehrender Verben geben: `Get`, `Set`, `Add`, `New`, `Remove`, `Clear`, `Push`, `Pop`, `Write`, `Export`, `Select`, `Sort`, `Update`, `Start`, `Stop`, `Invoke` usw. Außer diesen Basisoperationen gibt es auch Ausgabekommandos mit Verben wie `Out` und `Format`. Auch Bedingungen werden durch diese Syntax abgebildet (`Where-Object`).

■ 3.2 Aliase

Durch sogenannte Aliase kann die Eingabe von Commandlets verkürzt werden. So ist `ps` als Alias für `Get-Process` oder `help` für `Get-Help` vordefiniert. Statt `Get-Process i*` kann also auch geschrieben werden: `ps i*`.



HINWEIS: Manche PowerShell-Experten betrachten den Einsatz von Aliasen als schlechten Stil, der die Lesbarkeit von PowerShell-Skripten erschwert. Auf der anderen Seite ersparen Aliase eben Tipparbeit. Ob man vordefinierte und ggf. auch selbst definierte PowerShell-Aliase erlauben möchte, sollte man im Unternehmen als Richtlinie festlegen. Ich halte Alias insbesondere dann für kein Problem, wenn man die von Microsoft vordefinierten Aliase verwendet. Auch lokal innerhalb einer Skriptdatei selbstdefinierte Aliase sollten erlaubt sein, denn sie haben den Stellenwert von selbstdefinierten Funktionen, die sicherlich auch erlaubt sein müssen. Problematisch finde ich hingegen selbstdefinierte globale Aliase, denn diese setzen voraus, dass die Aliase auf jedem System, auf dem ein Skript ausgeführt werden soll, auch wirklich definiert sind und alle Administratoren, die die Skripte lesen und bearbeiten können sollen, diese Aliase auch kennen.

3.2.1 Aliase auflisten

Durch `Get-Alias` (oder den entsprechenden `Alias aliases`) erhält man eine Liste aller vordefinierten Abkürzungen in Form von Instanzen der Klasse `System.Management.Automation.AliasInfo`.

Durch Angabe eines Namens bei `Get-Alias` erhält man die Bedeutung eines Alias:

```
Get-Alias pgs
```

Möchte man zu einem Commandlet alle Aliase wissen, muss man allerdings schreiben:

```
Get-Alias | Where-Object { $_.definition -eq "Get-Process" }
```

Dies erfordert schon den Einsatz einer Pipeline, die erst im nächsten Kapitel besprochen wird.

Tabelle 3.2 Vordefinierte Aliase in der PowerShell

Alias	Commandlet
%	ForEach-Object
?	Where-Object
ac	Add-Content
asnp	Add-PSSnapIn
cat	Get-Content
cd	Set-Location
chdir	Set-Location
clc	Clear-Content
clear	Clear-Host
clhy	Clear-History
cli	Clear-Item
clp	Clear-ItemProperty
cls	Clear-Host
clv	Clear-Variable
cnsn	Connect-PSSession
compare	Compare-Object
copy	Copy-Item
cp	Copy-Item
cpj	Copy-Item
cpp	Copy-ItemProperty
cvpa	Convert-Path
dbp	Disable-PSBreakpoint
del	Remove-Item
diff	Compare-Object
dir	Get-ChildItem
dnsn	Disconnect-PSSession
ebp	Enable-PSBreakpoint
echo	Write-Output
epal	Export-Alias
epcsv	Export-CSV
epsn	Export-PSSession
erase	Remove-Item
etsn	Enter-PSSession

(Fortsetzung nächste Seite)

Tabelle 3.2 Vordefinierte Aliase in der PowerShell (Fortsetzung)

Alias	Commandlet
exsn	Exit-PSSession
fc	Format-Custom
fl	Format-List
foreach	ForEach-Object
ft	Format-Table
fw	Format-Wide
gal	Get-Alias
gbp	Get-PSBreakpoint
gc	Get-Content
gci	Get-ChildItem
gcm	Get-Command
gcs	Get-PSCallStack
gdr	Get-PSDrive
ghy	Get-History
gi	Get-Item
gjb	Get-Job
gl	Get-Location
gm	Get-Member
gmo	Get-Module
gp	Get-ItemProperty
gps	Get-Process
group	Group-Object
gsn	Get-PSSession
gsnp	Get-PSSnapIn
gsv	Get-Service
gu	Get-Unique
gv	Get-Variable
gwmi	Get-WmiObject
h	Get-History
history	Get-History
icm	Invoke-Command
iex	Invoke-Expression
ihy	Invoke-History
ii	Invoke-Item
ipal	Import-Alias
ipcsv	Import-CSV
ipmo	Import-Module
ipsn	Import-PSSession

Alias	Commandlet
irm	Invoke-RestMethod
ise	powershell_ise.exe
iwmi	Invoke-WMIMethod
iwr	Invoke-WebRequest
kill	Stop-Process
lp	Out-Printer
ls	Get-ChildItem
man	help
md	mkdir
measure	Measure-Object
mi	Move-Item
mount	New-PSDrive
move	Move-Item
mp	Move-ItemProperty
mv	Move-Item
nal	New-Alias
ndr	New-PSDrive
ni	New-Item
nmo	New-Module
npssc	New-PSSessionConfigurationFile
nsn	New-PSSession
nv	New-Variable
ogv	Out-GridView
oh	Out-Host
popd	Pop-Location
ps	Get-Process
pushd	Push-Location
pwd	Get-Location
r	Invoke-History
rbp	Remove-PSBreakpoint
rcjb	Receive-Job
rdsn	Receive-PSSession
rd	Remove-Item
rdr	Remove-PSDrive
ren	Rename-Item
ri	Remove-Item
rjb	Remove-Job
rm	Remove-Item
rmdir	Remove-Item

(Fortsetzung nächste Seite)

Tabelle 3.2 Vordefinierte Aliase in der PowerShell (*Fortsetzung*)

Alias	Commandlet
rmo	Remove-Module
rni	Rename-Item
rnp	Rename-ItemProperty
rp	Remove-ItemProperty
rsn	Remove-PSSession
rsnp	Remove-PSSnapin
rujb	Resume-Job
rv	Remove-Variable
rvpa	Resolve-Path
rwmi	Remove-WMIObject
sajb	Start-Job
sal	Set-Alias
saps	Start-Process
sasv	Start-Service
sbp	Set-PSBreakpoint
sc	Set-Content
select	Select-Object
set	Set-Variable
shcm	Show-Command
si	Set-Item
sl	Set-Location
sleep	Start-Sleep
sls	Select-String
sort	Sort-Object
sp	Set-ItemProperty
spjb	Stop-Job
spps	Stop-Process
spsv	Stop-Service
start	Start-Process
sujb	Suspend-Job
sv	Set-Variable
swmi	Set-WMIInstance
tee	Tee-Object
trcm	Trace-Command
type	Get-Content
where	Where-Object
wjb	Wait-Job
write	Write-Output

3.2.2 Neue Aliase anlegen

Einen neuen Alias definiert der Nutzer mit `Set-Alias` oder `New-Alias`, z. B.:

```
Set-Alias procs Get-Process  
New-Alias procs Get-Process
```

Der Unterschied zwischen `Set-Alias` und `New-Alias` ist marginal: `New-Alias` erstellt einen neuen Alias und liefert einen Fehler, wenn der zu vergebende Alias schon existiert. `Set-Alias` erstellt einen neuen Alias oder überschreibt einen Alias, wenn der zu vergebende Alias schon existiert. Mit dem Parameter `-description` kann man jeweils auch einen Beschreibungstext setzen.

Man kann einen Alias nicht nur für Commandlets, sondern auch für klassische Anwendungen vergeben, z. B.:

```
Set-Alias np notepad.exe
```



ACHTUNG: Beim Anlegen eines Alias wird nicht geprüft, ob das zugehörige Commandlet bzw. die Anwendung überhaupt existiert. Der Fehler würde erst beim Aufruf des Alias auftreten.

Beim Anlegen eines Alias muss man zudem aufpassen, dass man keine bestehenden Namen überschreibt, denn Aliase haben Priorität. Wenn man `Set-Alias notepad dir` eingibt, führt ab dann die Eingabe von `notepad` nicht mehr zu `notepad.exe`, sondern zum Commandlet `Get-ChildItem` (für das `dir` ein Alias ist). `notepad` ist dann also ein Alias für einen Alias.

Man kann in Aliasdefinitionen keinen Parameter mit Werten vorbelegen. Möchten Sie zum Beispiel definieren, dass die Eingabe von „Temp“ die Aktion „`Get-ChildItem c:\Temp`“ ausführt, brauchen Sie dafür eine Funktion. Mit einem Alias geht das nicht.

```
Function Temp { Get-ChildItem w:\temp }
```

Funktionen werden später (siehe Kapitel 6 „*PowerShell-Skripte*“) noch ausführlich besprochen. Die PowerShell enthält zahlreiche vordefinierte Funktionen, z. B. `c:`, `d:`, `e:` sowie `mkdir` und `help`.

Die neu definierten Aliase gelten jeweils nur für die aktuelle Instanz der PowerShell-Konsole. Man kann die eigenen Alias-Definitionen exportieren mit `Export-Alias` und später wieder importieren mit `Import-Alias`. Als Speicherformate stehen das CSV-Format und das PowerShell-Skriptdateiformat (`.ps1` – siehe spätere Kapitel) zur Verfügung. Bei dem `ps1`-Format ist zum späteren Reimport der Datei das Skript mit dem Punktoperator (engl. „Dot Sourcing“) aufzurufen.

	Dateiformat CSV	Dateiformat .ps1
Speichern	Export-Alias x:\meinealias.csv	Export-Alias x:\meinealias.ps1 -as script
Laden	Import-Alias x:\meinealias.csv	. x:\meinealias.ps1

Die Anzahl der Aliase ist im Standard auf 4096 beschränkt. Dies kann durch die Variable `$MaximumAliasCount` geändert werden.

3.2.3 Aliase entfernen

Aliase entfernen aus der aktuellen PowerShell kann man mit `Remove-Item alias:\AliasName`, also z. B.

```
Remove-Item alias:\np
```

Um den zuvor angelegten Alias zum Start von Notepad wieder zu löschen.

Seit PowerShell Core 6.0 gibt es auch das Commandlet `Remove-Alias` zu diesem Zweck:

```
Remove-Alias np
```

3.2.4 Aliase für Eigenschaften

Aliase sind auch auf Ebene von Eigenschaften definiert. So kann man statt

```
Get-Process processname, workingset
```

auch schreiben:

```
Get-Process name, ws
```

Diese Aliase der Attribute sind definiert in der Datei `types.ps1xml` im Installationsordner der PowerShell.

```

520 </Type>
521 <Type>
522   <Name>System.Diagnostics.Process</Name>
523   <Members>
524     <MemberSet>
525       <Name>PSStandardMembers</Name>
526       <Members>
527         <PropertySet>
528           <Name>DefaultDisplayPropertySet</Name>
529           <ReferencedProperties>
530             <Name>Id</Name>
531             <Name>Handles</Name>
532             <Name>CPU</Name>
533             <Name>SI</Name>
534             <Name>Name</Name>
535           </ReferencedProperties>
536         </PropertySet>
537       </Members>
538     </MemberSet>
539     <PropertySet>
540       <Name>PSConfiguration</Name>
541       <ReferencedProperties>
542         <Name>Name</Name>
543         <Name>Id</Name>
544         <Name>PriorityClass</Name>
545         <Name>FileVersion</Name>
546       </ReferencedProperties>
547     </PropertySet>
548     <PropertySet>...
549   </Members>
550 </Type>
551 <AliasProperty>
552   <Name>Name</Name>
553   <ReferencedMemberName>ProcessName</ReferencedMemberName>
554 </AliasProperty>
555 <AliasProperty>
556   <Name>SI</Name>
557   <ReferencedMemberName>SessionId</ReferencedMemberName>
558 </AliasProperty>
559 <AliasProperty>
560   <Name>Handles</Name>
561   <ReferencedMemberName>Handlecount</ReferencedMemberName>
562 </AliasProperty>
563 </Type>

```

Bild 3.14 types.ps1xml



ACHTUNG: Die types.ps1xml-Datei wird ab PowerShell 5.1 nicht mehr von der PowerShell verwendet, da das Einlesen der Datei die Startgeschwindigkeit der PowerShell-Konsolen negativ beeinflusst hat. Die Informationen liegen nun im C#-Code der Commandlets vor. Die types.ps1xml ist noch für den PowerShell 2.0-Kompatibilitätsmodus vorhanden.

■ 3.3 Ausdrücke

Ebenfalls als Befehl direkt in die PowerShell eingeben kann man Ausdrücke, z. B. mathematische Ausdrücke wie

```
10* (8 + 6)
```

oder Zeichenkettenausdrücke wie

```
"www." + "IT-Visions" + ".de"
```

Microsoft spricht hier vom Expression Mode der PowerShell im Kontrast zum Command Mode, der verwendet wird, wenn man

```
Write-Output (10* (8 + 6))
```

aufruft.

```

PowerShell
PS T:\> 10*(8+6)
140
PS T:\> "www." + "IT-Visions" + ".de"
www.IT-Visions.de
PS T:\> Write-Output 10*(8+6)
10*
14
PS T:\> Write-Output (10*(8+6))
140
PS T:\> |
  
```

Bild 3.15

Bei Write-Output (alias: echo) ist die Klammerung wichtig, sonst versteht die PowerShell $10*(8+6)$ als zwei getrennte Ausdrücke

Die PowerShell kennt zwei Bearbeitungsmodi für Befehle: einen Befehlsmodus (Command Mode) und einen Ausdrucksmodus (Expression Mode). Im Befehlsmodus werden alle Eingaben als Zeichenketten behandelt. Im Ausdrucksmodus werden Zahlen und Operationen verarbeitet. Als Faustregel gilt: Wenn eine Zeile mit einem Buchstaben oder den Sonderzeichen kaufmännisches Und [&], Punkt [.] oder Schrägstrich [/] beginnt, dann ist die Zeile im Befehlsmodus. Wenn die Zeile mit einer Zahl, einem Anführungszeichen (["] oder [']), einer runden Klammer [(] oder dem [@]-Zeichen („Klammeraffe“) beginnt, dann ist die Zeile im Ausdrucksmodus.

Befehls- und Ausdrucksmodus können gemischt werden. Dabei muss man in der Regel runde Klammern zur Abgrenzung verwenden. In einen Befehl kann ein Ausdruck durch Klammern eingebaut werden. Außerdem kann eine Pipeline mit einem Ausdruck beginnen. Die folgende Tabelle zeigt verschiedene Beispiele zur Erläuterung. Echo ist der Alias für Write-Output.

Tabelle 3.3 Ausdrücke in der PowerShell

Beispiel	Bedeutung
2+3	Ein Ausdruck – die PowerShell führt die Berechnung aus und liefert 5.
echo 2+3	Ein reiner Befehl. „2+3“ wird als Zeichenkette angesehen und ohne Auswertung auf dem Bildschirm ausgegeben.
echo (2+3)	Ein Befehl mit integriertem Ausdruck. Auf dem Bildschirm erscheint 5.
2+3 echo	Eine Pipeline, die mit einem Ausdruck beginnt. Auf dem Bildschirm erscheint 5.
echo 2+3 7+6	Eine unerlaubte Eingabe. Ausdrücke dürfen in der Pipeline nur als erstes Element auftauchen.
\$a = Get-Process	Ein Ausdruck mit integriertem Befehl. Das Ergebnis wird einer Variablen zugewiesen.
\$a Get-Process	Eine Pipeline, die mit einem Ausdruck beginnt. Der Inhalt von \$a wird als Parameter an Get-Process übergeben.
Get-Process \$a	Eine unerlaubte Eingabe. Ausdrücke dürfen in der Pipeline nur als erstes Element auftauchen.
"Anzahl der laufenden Prozesse: (Get-Process).Count"	Es ist wohl nicht das, was gewünscht ist, denn die Ausgabe ist: Anzahl der laufenden Prozesse: (Get-Process).Count
"Anzahl der laufenden Prozesse: \$(Get-Process).Count"	Jetzt ist die Ausgabe „Anzahl der laufenden Prozesse: 95“, weil \$(...) einen Unterausdruck (Subexpression) einleitet und dafür sorgt, dass Get-Process ausgeführt wird.

■ 3.4 Externe Befehle (klassische Kommandozeilenbefehle)

Viele moderne Software besitzt leider noch keine PowerShell-Commandlets, sondern nur sehr altertümliche Kommandozeilenwerkzeuge, die lediglich Textausgaben statt strukturierter Objekte liefern und die – verglichen mit der PowerShell – sehr inkonsistent in ihrem Verhalten sind. Besonders schlimme Beispiele sind die Kommandozeilenwerkzeuge von Docker (Docker-CLI: `docker.exe`) und Git (Git-CLI: `git.exe`). Aber für auch die Verwaltung der Azure-Cloud stellt Microsoft nicht für alle Aufgaben PowerShell-Commandlets bereit, sondern manchmal nur klassische Kommandozeilenbefehle (Azure-CLI: `az.cmd`).

Alle Eingaben in der PowerShell, die nicht als Commandlets, Funktionsname, Laufwerksname oder mathematische Formeln erkannt werden, werden als externe Anwendungen behandelt. Es können sowohl klassische Kommandozeilenbefehle (wie `ping.exe`, `docker.exe`, `az.exe`, `git.exe`, `ipconfig.exe` und `netstat.exe`) als auch Windows-Anwendungen ausgeführt werden.

Die Eingabe `c:\Windows\notepad.exe` ist daher möglich, um den „beliebten“ Windows-Editor zu starten. Auf gleiche Weise können auch WSH-Skripte aus der PowerShell heraus gestartet werden.

Die folgende Bildschirmabbildung zeigt den Aufruf von *netstat.exe*. Im Beispiel kommt zusätzlich das Commandlet *Select-String* zum Einsatz, das nur die Zeilen filtert, die das Wort „*mapi*“ enthalten. Sonst wird die Ausgabeliste sehr lang.

```
PS T:\> netstat | select-string "mapi"

TCP    192.168.1.60:50802    mapi:https      TIME_WAIT
TCP    192.168.1.60:53189    mapi:https      ESTABLISHED
TCP    192.168.1.60:53190    mapi:https      ESTABLISHED
TCP    192.168.1.60:53226    mapi:https      ESTABLISHED
TCP    192.168.1.60:53908    mapi:https      ESTABLISHED
TCP    192.168.1.60:54130    mapi:https      ESTABLISHED
TCP    192.168.1.60:60802    mapi:https      TIME_WAIT

PS T:\>
```

Bild 3.16 Ausführung von *netstat*

Wenn ein Leerzeichen im Pfad zu einer *.exe*-Datei vorkommt, dann kann man die Datei so nicht aufrufen (hier wird nach einem Befehl „*X:\data\software\Windows*“ gesucht):

```
X:\data\software\Windows Tools\ImageEditor.exe
```

Auch die naheliegende Lösung der Verwendung von Anführungszeichen funktioniert nicht (hier wird die Zeichenkette ausgegeben):

```
"X:\data\software\Windows Tools\ImageEditor.exe"
```

Korrekt ist die Verwendung des kaufmännischen Und (&), das dafür sorgt, dass der Inhalt der Zeichenkette als Befehl betrachtet und ausgeführt wird:

```
& "X:\data\software\Windows Tools\ImageEditor.exe"
```



ACHTUNG: Grundsätzlich könnte es passieren, dass ein interner Befehl der PowerShell (Commandlet, Alias oder Function) genauso heißt wie ein externer Befehl. Die PowerShell warnt in einem solchen Fall nicht vor der Doppeldeutigkeit, sondern die Ausführung erfolgt nach folgender Präferenzliste:

- Aliase
- Funktionen
- Commandlets
- Externe Befehle

3.5 Dateinamen

Beim direkten Aufruf von Datendateien (z. B. .doc-Dateien) wird entsprechend den Windows-Einstellungen in der Registrierungsdatenbank die Standardanwendung gestartet und damit das Dokument geladen.



HINWEIS: Dateinamen und Ordnerpfade müssen nur in Anführungszeichen (einfache oder doppelte) gesetzt werden, falls sie Leerzeichen enthalten.

```
PowerShell
PS T:\> dir 'C:\Program Files\' -filter w*

Directory: C:\Program Files

Mode                LastWriteTime         Length Name
----                -
d-----          14.01.2021    00:01           Windows Defender
d-----          11.03.2022    09:13           Windows Defender Advanced Threat Protection
d-----          20.07.2021    20:21           Windows Mail
d-----          13.10.2021    22:24           Windows Media Player
d-----          07.12.2019    10:52           Windows Multimedia Platform
d-----          07.12.2019    10:49           Windows NT
d-----          14.01.2021    00:01           Windows Photo Viewer
d-----          07.12.2019    10:52           Windows Portable Devices
d-----          07.12.2019    10:31           Windows Security
d-----          06.01.2021    13:26           WindowsPowerShell
d-----          07.01.2021    08:49           WinRAR

PS T:\> |
```

Bild 3.17 Anführungszeichen bei Pfadangaben

4

Hilfefunktionen

Dieses Kapitel beschreibt die Hilfsfunktionen der PowerShell.

■ 4.1 Auflisten der verfügbaren Befehle

Die Liste aller verfügbaren Befehle (PowerShell-Commandlets, PowerShell-Funktionen, PowerShell-Aliase und klassische ausführbare Dateien) erhält man in der PowerShell auch durch das Commandlet `Get-Command`.

Die Eingabe

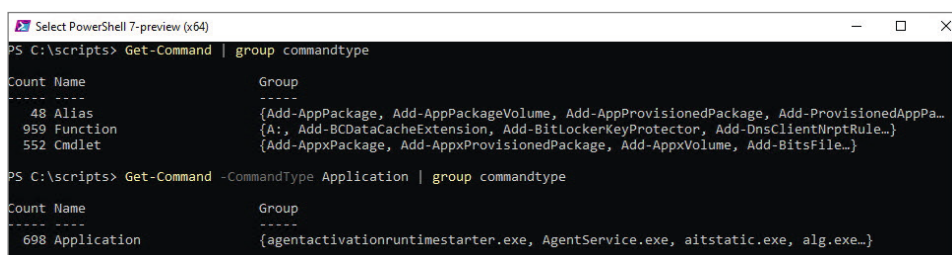
```
Get-Command
```

liefert aber im Standard nur die ersten drei o.g. Befehlsarten.

Erst mit

```
Get-Command -CommandType Application
```

bekommt man auch die klassischen ausführbaren Dateien



```
Select PowerShell 7-preview (x64)
PS C:\scripts> Get-Command | group commandtype
Count Name Group
-----
48 Alias {Add-AppPackage, Add-AppPackageVolume, Add-AppProvisionedPackage, Add-ProvisionedAppPa...
959 Function {A:, Add-BCDataCacheExtension, Add-BitLockerKeyProtector, Add-DnsClientNrptRule...}
552 Cmdlet {Add-AppxPackage, Add-AppxProvisionedPackage, Add-AppxVolume, Add-BitsFile...}

PS C:\scripts> Get-Command -CommandType Application | group commandtype
Count Name Group
-----
698 Application {agentactivationruntimestarter.exe, AgentService.exe, aitstatic.exe, alg.exe...}
```

Bild 4.1 Gruppierung nach Befehlstypen

Bei `Get-Command` sind auch Muster erlaubt.

- `Get-Command Get-*` liefert alle Befehle, die mit „get“ anfangen.
- `Get-Command [gs]et-*` liefert alle Befehle, die mit „get“ oder „set“ anfangen.
- `Get-Command *-Service` liefert alle Befehle, die das Substantiv „Service“ besitzen.
- `Get-Command -noun Service` liefert ebenfalls alle Befehle, die das Substantiv „Service“ besitzen.

- `Get-Command *wmi*` liefert alle Befehle, die die Buchstabenfolge „wmi“ enthalten (und mutmaßlich mit der Windows Management Instrumentation zu tun haben).
- `Get-Command | Where-Object { $_.name -like "*cim*" -or $_.name -like "*wmi*" }` liefert alle Befehle, die die Buchstabenfolge „wmi“ oder „cim“ enthalten. Ohne ein weiteres Commandlet `Where-Object`, das erst im nächsten Kapitel näher erläutert wird, ist diese Abfrage nicht machbar.
- Das Commandlet `Get-Command` kann auch verwendet werden, um die Information zu erhalten, was die PowerShell unter einem Befehl versteht. `Get-Command` sucht nach angegebenen Namen in Commandlets, Aliasen, Funktionen, Skriptdateien und ausführbaren Dateien (siehe nächste Abbildung).
- `Get-Command *.exe` zeigt eine Liste aller direkt aufrufbaren EXE-Dateien.

```

PS T:\> Get-Command Measure-*
CommandType Name Version Source
-----
Cmdlet Measure-Command 3.1.0.0 Microsoft.PowerShell.Utility
Cmdlet Measure-Object 3.1.0.0 Microsoft.PowerShell.Utility
Cmdlet Measure-VM 2.0.0.0 Hyper-V
Cmdlet Measure-VMReplication 2.0.0.0 Hyper-V
Cmdlet Measure-VMResourcePool 2.0.0.0 Hyper-V

PS T:\> Get-Command ps
CommandType Name Version Source
-----
Alias ps -> Get-Process

PS T:\> Get-Command notepad.exe
CommandType Name Version Source
-----
Application notepad.exe 10.0.15... C:\WINDOWS\system32\notepad.exe

PS T:\> Get-Command c:
CommandType Name Version Source
-----
Function C:

PS T:\>

```

Bild 4.2 Beispiele zum Einsatz von `Get-Command`

■ 4.2 Praxistipp: Den Standort eines Kommandozeilenbefehls suchen

Gibt man nach `Get-Command` den Namen einer `.exe`-Datei an, zeigt die PowerShell, in welchem Pfad die ausführbare Datei gefunden werden kann. Gesucht wird dabei nur in den Pfaden gemäß der Umgebungsvariablen `%Path%`.



HINWEIS: Dies entspricht der Funktion des klassischen Windows-Befehls „where.exe“ (wobei man bei `Get-Command` nicht nur den Pfad, sondern direkt ein Objekt mit mehr Informationen erhält, siehe nächste Abbildung). Man kann „where.exe“ auch in der PowerShell verwenden, muss aber unbedingt „where.exe“ mit Dateinamenserweiterung angeben. Die einfache Eingabe „where“ würde die PowerShell als Aufruf des Commandlets `Where-Object` verstehen, dessen Alias „where“ ist.

```

pwsh
PS X:\> get-command ping.exe

CommandType      Name                               Version      Source
-----
Application      PING.EXE                          10.0.1836... C:\Windows\system32\PING.EXE

PS X:\> get-command nuget.exe

CommandType      Name                               Version      Source
-----
Application      nuget.exe                          5.3.1.0      C:\Program Files\nuget\nuget.exe

PS X:\> where.exe ping.exe
C:\Windows\System32\PING.EXE
PS X:\> where.exe nuget.exe
C:\Program Files\nuget\nuget.exe
PS X:\>

```

Bild 4.3 Suche des Standorts einer EXE-Datei mit Get-Command oder Where.exe

■ 4.3 Anzahl der Befehle

Windows 11 mit PowerShell 5.1 bietet 1670 Commandlets, Windows Server 2022 bietet 1799 Commandlets. Die rasante Fortentwicklung der Funktionalität der Windows PowerShell, aber auch der Einbruch beim Umstieg auf die Core Editionen seit PowerShell 6.0 sowie die gravierende Abhängigkeit ihrer Mächtigkeit von dem jeweils installierten Betriebssystem, zeigt die folgende Tabelle.

Tabelle 4.1 Änderung der Mächtigkeit der PowerShell zwischen den verschiedenen Versionen

PowerShell-Version	Betriebssystem	Anzahl der Commandlets und Funktionen in der Grundinstallation
PowerShell 7.2	Windows	1507
PowerShell 7.2	Linux und macOS	286
PowerShell Core 6.2	Windows	1439
PowerShell Core 6.1	Windows	1436
PowerShell Core 6.0	Windows	425
PowerShell 5.1	Windows 10	1586
PowerShell 5.1	Windows 11	1670
PowerShell 5.1	Windows Server 2022	1799
PowerShell 5.0	Windows 10 (Threshold 1, Ursprungsversion vom 29.05.2016)	1404
PowerShell 4.0	Windows Server 2012 R2	1376
PowerShell 4.0	Windows 8.1	1132
PowerShell 4.0	Windows 7	573
PowerShell 3.0	Windows 8	945
PowerShell 3.0	Windows 7	561
PowerShell 2.0	Windows 7	273
PowerShell 1.0	Alle	163

Ermitteln kann man diese Zahlen mit:

```
(Get-Command | Sort-Object Name | Get-Unique).Count
```

Get-Command liefert unter PowerShell seit 2.0 sowohl Commandlets als auch eingebaute Funktionen (deren Handhabung oft der von Commandlets entspricht, nur die Art der Implementierung ist anders). Unter PowerShell 1.0 musste man die Funktionen separat zählen mit:

```
(dir function:).count
```

Wenn Sie wissen möchten,

- welche Commandlets zwischen zwei Versionen hinzugekommen sind, oder
- hinsichtlich welcher Commandlets sich zwei Systeme unterscheiden,

können Sie dies wie folgt ermitteln:

Auf dem einen System exportieren Sie eine Liste der Commandlets in eine Textdatei. Auf einem System mit PowerShell 1.0/2.0/3.0 führen Sie folgende Befehle aus, um Commandlets und Funktionen zu exportieren:

```
# PowerShell bis einschließlich 3.0, hier am Beispiel 3.0
Get-Command | ft name -hide | out-file w:\ps3_commandlets.txt
dir function: | ft Name -hide | out-file w:\ps3_commandlets.txt -Append
```

Auf einem System mit PowerShell ab Version 4.0 brauchen Sie nur einen Befehl (dieser exportiert Commandlets und Funktionen):

```
# PowerShell ab 4.0, z.B. für PowerShell 5.1
Get-Command | sort-Object name | get-unique | foreach { ([string]$_name).Trim() } |
out-file w:\ps51_commandlets.txt

# PowerShell ab 4.0, z.B. für PowerShell 7.x
Get-Command | sort-Object name | get-unique | foreach { ([string]$_name).Trim() } |
out-file w:\ps70_commandlets.txt
```



HINWEIS: Get-Unique ist hier erforderlich, weil Commandlets in verschiedenen Versionen eines Moduls doppelt vorkommen könnten. Es ist möglich, dass auf einem System ein Modul in mehreren Versionen in verschiedenen Verzeichnissen existiert.

Dann führt man beide Textdateien auf einem System zusammen und führt dort aus:

```
# Laden der Dateien für Vergleich
$wps51 = Get-content w:\ps51_Commandlets.txt | sort
$ps70 = Get-content w:\ps70_Commandlets.txt | sort

# Vergleich: Befehle nur in WPS 5.1
compare-object $wps51 $ps70 -syncwindow 2000 | where sideindicator -eq "<=" #
optional: | out-file w:\ps70_Commandlets_ fehlend.txt
```

```
# Vergleich: Befehle nur in PS 7.x
compare-object $wps51 $ps70 -syncwindow 2000 | where sideindicator -eq "=" #
optional: | out-file w:\ps70_Commandlets_neu.txt
```

4.4 Volltextsuche

Get-Command sucht nur in den Commandletnamen. Mit Get-Help kann man unter Angabe einer beliebigen Zeichenkette in den Hilfedateien suchen.

Beispiel:

```
Get-Help "Local user account"
```

```
PS T:\> Get-Help "Local user account"
Name
----
Connect-PSSession          Cmdlet      Microsoft.PowerShell.Core Reconnects to disconnected sessions.
Enter-PSSession           Cmdlet      Microsoft.PowerShell.Core Starts an interactive session with a remote co...
Get-PSSession             Cmdlet      Microsoft.PowerShell.Core Gets the Windows PowerShell sessions on local ...
Invoke-Command            Cmdlet      Microsoft.PowerShell.Core Runs commands on local and remote computers.
New-PSSession             Cmdlet      Microsoft.PowerShell.Core Creates a persistent connection to a local or ...
Receive-PSSession        Cmdlet      Microsoft.PowerShell.Core Gets results of commands in disconnected sessi...
Invoke-RestMethod         Cmdlet      Microsoft.PowerShell.U... Sends an HTTP or HTTPS request to a RESTful we...
Invoke-WebRequest        Cmdlet      Microsoft.PowerShell.U... Gets content from a web page on the Internet.
Add-LocalGroupMember     Cmdlet      Microsoft.PowerShell.L... Adds members to a local group.
Disable-LocalUser        Cmdlet      Microsoft.PowerShell.L... Disables a local user account.
Enable-LocalUser         Cmdlet      Microsoft.PowerShell.L... Enables a local user account.
Get-LocalUser            Cmdlet      Microsoft.PowerShell.L... Gets local user accounts.
New-LocalUser            Cmdlet      Microsoft.PowerShell.L... Creates a local user account.
Remove-LocalGroupMember  Cmdlet      Microsoft.PowerShell.L... Removes members from a local group.
Remove-LocalUser        Cmdlet      Microsoft.PowerShell.L... Deletes local user accounts.
Rename-LocalUser         Cmdlet      Microsoft.PowerShell.L... Renames a local user account.
Set-LocalUser            Cmdlet      Microsoft.PowerShell.L... Modifies a local user account.
Set-AssignedAccess       Function    AssignedAccess           Configures a user to launch only one app.
about_ActivityCommonParameters HelpFile    This topic describes the parameters that Windows PowerShell
about_WorkflowCommonParameters HelpFile    This topic describes the parameters that are v...
about_ActivityCommonParameters HelpFile    Describes the parameters that Windows PowerShell
about_WorkflowCommonParameters HelpFile    This topic describes the parameters that are v...
```

Bild 4.4 Volltextsuche mit Get-Help

4.5 Erläuterungen zu den Befehlen

Einen Hilfetext zu einem Commandlet bekommt man über Get-Help commandletname, z. B.:

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

Dabei kann man durch die Parameter -detailed, -example und -full mehr Hilfe erhalten. Die Hilfe erscheint abhängig von der installierten Sprachversion der PowerShell. Der Autor dieses Buchs verwendet jedoch primär englische Betriebssysteme und Anwendungen.

```

PowerShell
PS T:\> Get-Help Get-Process -full
NAME
    Get-Process

ÜBERSICHT
    Gets the processes that are running on the local computer or a remote computer.

SYNTAX
    Get-Process [[-Name <String[]>] [-ComputerName <String[]>] [-FileVersionInfo] [-Module] [<CommonParameters>]
    Get-Process [-ComputerName <String[]>] [-FileVersionInfo] -Id <Int32[]> [-Module] [<CommonParameters>]
    Get-Process [-ComputerName <String[]>] [-FileVersionInfo] -InputObject <Process[]> [-Module] [<CommonParameters>]
    Get-Process -Id <Int32[]> -IncludeUserName [<CommonParameters>]
    Get-Process [[-Name <String[]>] -IncludeUserName [<CommonParameters>]
    Get-Process -IncludeUserName -InputObject <Process[]> [<CommonParameters>]

BESCHREIBUNG
    The Get-Process cmdlet gets the processes on a local or remote computer.

    Without parameters, this cmdlet gets all of the processes on the local computer. You can also specify a particular process by process name or process ID (PID) or pass a process object through the pipeline to this cmdlet.

    By default, this cmdlet returns a process object that has detailed information about the process and supports methods that let you start and stop the process. You can also use the parameters of the Get-Process cmdlet to get file version information for the program that runs in the process and to get the modules that the process loaded.

PARAMETER
    -ComputerName <String[]>
        Specifies the computers for which this cmdlet gets active processes. The default is the local computer.

        Type the NetBIOS name, an IP address, or a fully qualified domain name (FQDN) of one or more computers. To specify the local computer, type the computer name, a dot (.), or localhost.

        This parameter does not rely on Windows PowerShell remoting. You can use the ComputerName parameter of this cmdlet even if your computer is not configured to run remote commands.

        Erforderlich?           false
        Position?              named
        Standardwert           None
        Pipelineeingaben akzeptieren? True (ByPropertyName)
        Platzhalterzeichen akzeptieren? false
  
```

Bild 4.5 Ausschnitt aus dem Hilfetext zum Commandlet Get-Process



TIPP: Alternativ zum Aufruf von `Get-Help` kann man auch den allgemeinen Parameter `-?` an das Commandlet anhängen, z. B. `Get-Process -?`. Dann erhält man die Kurzversion der Hilfe, hat aber keine Option für die ausführlicheren Versionen.

4.6 Hilfe zu Parametern

Um zu sehen, welche Parameter ein Befehl bietet, kann man `Get-Help` mit dem Parameter `-Parameter` verwenden:

```
Get-Help Get-Process -parameter "*" | ft name, type
```

Einige Commandlets (z. B. `New-Button` aus dem WPK (Windows Presentation Foundation (WPF) PowerShell Kit), siehe Kapitel 64 „*Grafische Benutzeroberflächen*“) haben sehr viele Parameter (in diesem Fall 180!). Hier kann man auch filtern:

```
Get-Help New-Button -parameter "on_*" | ft name, type
```

Genauere Hilfe zu einem einzelnen Parameter erhält man, wenn man nach `-parameter` den Namen angibt und die weitere Formatierung weglässt. Die folgende Abbildung zeigt, wie man Hilfe zu dem Parameter `-ForegroundColor` im Commandlet `Write-Host` erhält. Neben den möglichen Farbwerten sagt die Hilfe auch, dass

- die Angabe einer Farbe nicht erforderlich ist
- die Farbangabe nicht über die Position des Parameters gebunden wird, d.h., dass immer der Parametername anzugeben ist
- der Farbwert auch nicht aus der Pipeline eingelesen werden kann
- im Farbwert keine Platzhalter erlaubt sind

```

PS T:\> Get-Help Write-Host -Parameter ForegroundColor
-ForegroundColor <ConsoleColor>
  Specifies the text color. There is no default. The acceptable values for this parameter are:
  - Black
  - DarkBlue
  - DarkGreen
  - DarkCyan
  - DarkRed
  - DarkMagenta
  - DarkYellow
  - Gray
  - DarkGray
  - Blue
  - Green
  - Cyan
  - Red
  - Magenta
  - Yellow
  - White

  Erfor derlich?           false
  Position?               named
  Standardwert            None
  Pipelineeingaben akzeptieren? False
  Platzhalterzeichen akzeptieren? false
  
```

Bild 4.6 Hilfe zu dem Parameter `-ForegroundColor` beim Commandlet `Write-Host`

Schaut man sich hingegen die Hilfe zum Parameter `-Name` beim Commandlet `Get-Service` an, sieht man zwar weniger Text, aber mehr Möglichkeiten:

- Es kann nicht nur eine feste Menge von Zeichenketten, sondern eine beliebige Zeichenkette übergeben werden. Dies zeigt der Typ `<string>` an.
- Genau genommen steht da `<string[]>`. Die eckigen Klammern bedeuten „Menge“, es kann also nicht nur eine Zeichenkette, sondern auch eine Menge von Zeichenketten übergeben werden (Beispiel: Dienste, die mit dem Buchstaben `a` beginnen oder enden oder mit `x` beginnen oder enden: `Get-Service -name „a*“, „*a“, „x*“, „*x“`).
- Der Wert kann über seine Position (0 bedeutet: an erster Stelle) übergeben werden. Daher kann man `-name` weglassen, sofern man den Wert für den Parameter an erster Stelle übergibt: `Get-Service "a*", "*a", "x*", "*x"`

- Der Werte (oder die Werte) für den Parameter `-name` kann auch als Wert aus der Pipeline gelesen werden. Möglich ist also `"a*" | Get-Service` oder `"a*", "*a", "x*", "*x" | Get-Service`

```
PS T:\> Get-Help Get-Service -Parameter Name
-Name <String[]>
  Specifies the service names of services to be retrieved. Wildcards are permitted. By default, this cmdlet gets all of the services on the computer.
Erforderlich?      false
Position?          0
Standardwert       None
Pipelineeingaben akzeptieren? True (ByPropertyName, ByValue)
Platzhalterzeichen akzeptieren? false
```

Bild 4.7 Hilfe zu dem Parameter `-Name` beim Commandlet `Get-Service`



HINWEIS: Leider sind dynamische (d. h. von anderen Parametern abhängige) Parameter zu Commandlets nicht in der Hilfe verzeichnet.

```
PS T:\> Get-Help dir -Parameter Eku
Get-Help : Kein Parameter entspricht dem Kriterium "Eku".
In Zeile: 1 Zeichen:
+ ~~~~~
+ get-help dir -Parameter Eku
+ ~~~~~
+ CategoryInfo          : InvalidArgument (System.Manageme...CommandHelpInfo:ProviderCommandHelpInfo) [Get-Help], PSArgumentException
+ FullyQualifiedErrorId : NoParamsFound,Microsoft.PowerShell.Commands.GetHelpCommand

PS T:\> dir cert: -ekj
+~ Eku
+~ [string[]] Eku
```

4.7 Hilfe mit Show-Command

Die PowerShell ist kommandozeilenorientiert. Vor der PowerShell 3.0 gab es in der PowerShell nur zwei Befehle, die eine grafische Benutzeroberfläche zeigten: `Out-GridView` (zur Ausgabe von Objekten in einer filter- und sortierbaren Tabelle) und `Get-Credential` (zur Abfrage von Benutzername und Kennwort).

Seit PowerShell 3.0 kann sich der PowerShell-Nutzer mit dem Commandlet `Show-Command` für jedes PowerShell-Commandlet und jede Function eine grafische Eingabemaske zeigen lassen.



ACHTUNG: Die grafische Benutzeroberfläche, die das Commandlet zeigt, basiert auf der Windows Presentation Foundation (WPF). Da es die WPF nur unter Windows gibt, funktioniert dieses Commandlet nicht in PowerShell unter Linux und macOS.

Die nächste Abbildung zeigt `Show-Command` für das Commandlet `Stop-Service`. Ziel von `Show-Command` ist es, insbesondere Einsteigern die Erfassung der Parameter zu erleichtern. Pflichtparameter sind mit einem Stern gekennzeichnet. Ein Klick auf die „Copy“-Schaltfläche legt den erzeugten Befehl in die Zwischenablage, ohne ihn auszuführen.



TIPP: Das Fenster „Befehls-Add-On“ in dem ISE ist eine modifizierte Version von `Show-Command`.

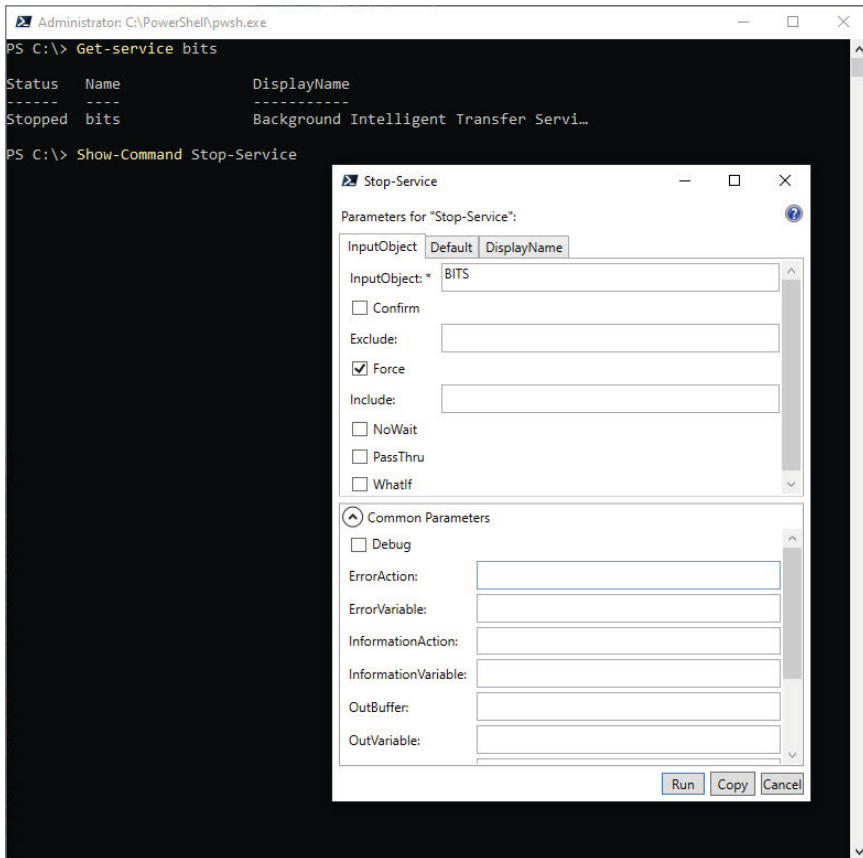


Bild 4.8 Show-Command bietet Eingabehilfen für Einsteiger.

■ 4.8 Hilfenfenster

Seit PowerShell 3.0 kann man auch aus der PowerShell-Konsole heraus ein eigenständiges Hilfenfenster starten, indem man bei `Get-Help` den Parameter `-ShowWindow` verwendet.

```
Get-Help "Set-PrintConfiguration" -ShowWindow
```

Das Hilfenfenster nutzt zur Hervorhebung fette Schrift, bietet eine Zoomfunktion und eine Volltextsuche an (vgl. die nachstehende Abbildung).



HINWEIS: `Get-Help` bietet in PowerShell unter Linux und macOS keinen Parameter `-ShowWindow`, da dieses Fenster auf der Windows Presentation Foundation (WPF) basiert, die es in .NET Core für Linux und macOS nicht gibt.

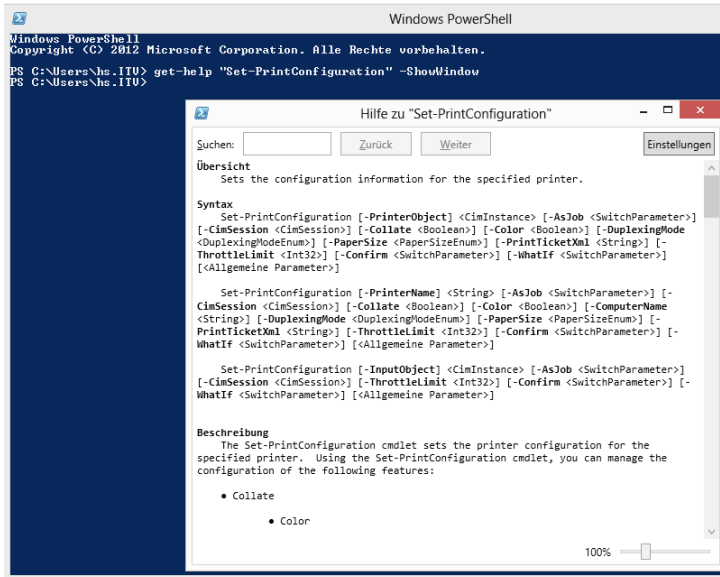


Bild 4.9 Hilfenfenster, das Get-Help durch den Parameter -ShowWindow startet

Eine grafische Hilfedatei im *.chm*-Dateiformat zur PowerShell gab es nur für die PowerShell 1.0 und 2.0 als Zusatz.

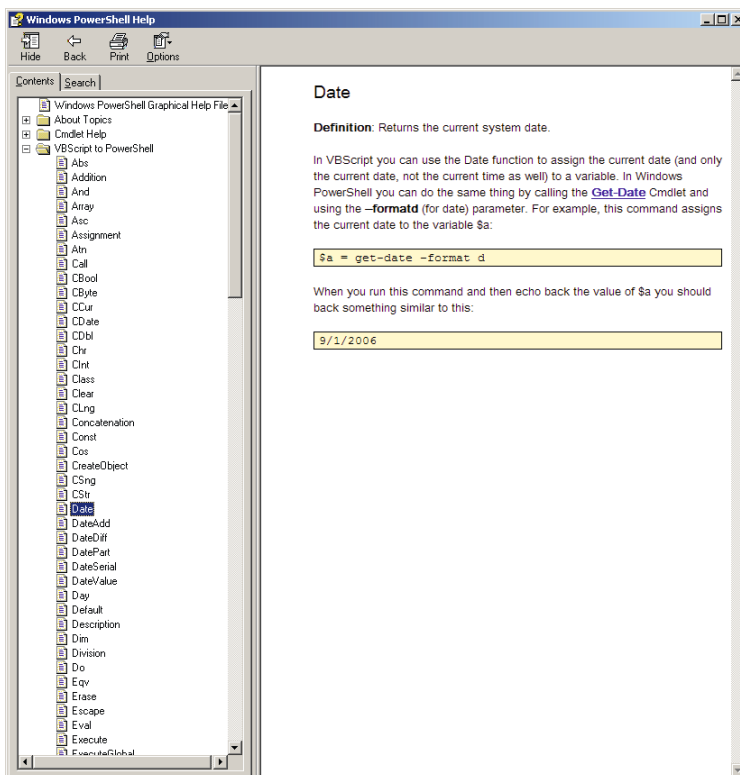


Bild 4.10 Hilfe zum Transfer von VBScript nach PowerShell

4.9 Allgemeine Hilfetexte

Die PowerShell enthält auch einige allgemeine Hilfetexte. Diese Dokumente beginnen mit „about“. Man findet sie mit `Get-Help about_`. Ein konkretes Dokument ruft man dann unter Angabe des kompletten Dokumentennamen ab: z. B. `Get-Help about_arrays`.

```
PS T:\> Get-help about_
Name                Category  Module  Synopsis
----                -
about_ActivityCommonParameters HelpFile
about_Aliases        HelpFile
about_Arithmetic_Operators HelpFile
about_Arrays         HelpFile
about_Assignment_Operators HelpFile
about_Automatic_Variables HelpFile
about_Break          HelpFile
about_Checkpoint-Workflow HelpFile
about_CimSession     HelpFile
about_Classes        HelpFile
about_Command_Precedence HelpFile
about_Command_Syntax HelpFile
about_Comment_Based_Help HelpFile
about_CommonParameters HelpFile
about_Comparison_Operators HelpFile
about_Continue       HelpFile
about_Core_Commands HelpFile
about_Data_Sections HelpFile
about_Debuggers      HelpFile
about_DesiredStateConfiguration HelpFile
about_Do              HelpFile
about_Environment_Variables HelpFile
about_Escape_Characters HelpFile
about_Eventlogs       HelpFile
about_Execution_Policies HelpFile
about_For             HelpFile
about_ForEach-Parallel HelpFile
about_Foreach         HelpFile
about_Format_psxml    HelpFile
about_Functions       HelpFile
about_Functions_Advanced HelpFile
about_Functions_Advanced_Methods HelpFile
about_Functions_Advanced_Parameters HelpFile
about_Functions_CmdletBinding HelpFile
about_Functions_OutputTypeAttributes HelpFile
```

Bild 4.11 Ausschnitt aus der Liste der „About“-Dokumente

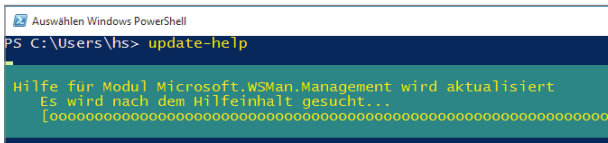
4.10 Aktualisieren der Hilfedateien

Die Hilfeinformationen, die durch `Get-Help` ausgelesen werden können, sind in XML-Dateien gespeichert. Das verwendete XML-Format heißt Microsoft Assistance Markup Language (MAML). Die Hilfe-Dateien sind den einzelnen Modulen zugeordnet.

```
Microsoft.PowerShell.Commands.Management.dll-help.xml
9897 <command:syntaxItem>
9898 <command:name>Get-Process</command:name>
9899 <command:parameter required="false" variableLength="true" globbing="true" pipelineInput="true (ByPropertyName)"
position="1" aliases="">
9900 <command:name>Name</command:name>
9901 <command:description>
9902 <command:para>Specifies one or more processes by process name. You can type multiple process names (separated by commas)
and use wildcard characters. The parameter name ("Name") is optional.</command:para>
9903 </command:description>
9904 <command:parameterValue required="true" variableLength="true">String[]</command:parameterValue>
9905 </command:parameter>
9906 <command:parameter required="false" variableLength="true" globbing="false" pipelineInput="true (ByPropertyName)"
position="named" aliases="">
9907 <command:name>ComputerName</command:name>
9908 <command:description>
9909 <command:para>Gets the processes running on the specified computers. The default is the local computer.</command:para>
9910 <command:para>Type the NetBIOS name, an IP address, or a fully qualified domain name of one or more computers. To specify
the local computer, type the computer name, a dot (.), or "localhost".</command:para>
9911 <command:para>This parameter does not rely on Windows PowerShell remoting. You can use the ComputerName parameter of Get-Process
even if your computer is not configured to run remote commands.</command:para>
9912 </command:description>
9913 <command:parameterValue required="true" variableLength="true">String[]</command:parameterValue>
9914 </command:parameter>
9915 <command:parameter required="false" variableLength="false" globbing="false" pipelineInput="false" position="named"
aliases="">
9916 <command:name>FileVersionInfo</command:name>
9917 <command:description>
9918 <command:para>Gets the file version information for the program that runs in the process.</command:para>
9919 <command:para>On windows Vista and later versions of Windows, you must open Windows PowerShell with the "Run as
administrator" option to use this parameter on processes that you do not own.</command:para>
9920 <command:para>You cannot use the FileVersionInfo and ComputerName parameters of the Get-Process cmdlet in the same
command. To get file version information for a process on a remote computer, use the Invoke-Command cmdlet.</command:para>
```

Bild 4.12 Ausschnitt aus der Hilfedatei Microsoft.PowerShell.Commands.Management.dll-help.xml

Mit PowerShell 3.0 hatte Microsoft die Möglichkeit eingeführt, die Hilfe-Dateien aus der laufenden PowerShell heraus zu aktualisieren („Updatable Help System“). Die Ausführung des Commandlets `Update-Help` kontaktiert den Microsoft-Downloadserver (`download.microsoft.com`) und aktualisiert im laufenden Betrieb die Hilfedateien. Auch wenn es sich um relativ kleine Dateien handelt (aktuell insgesamt nur rund 10 MB), dauert der Download über eine 50-MBit-Leitung zwei bis drei Minuten. Der Download besteht für jedes PowerShell-Modul aus einer sogenannten Help-Info-Datei, die als wesentliche Information die Sprache und die Versionsnummer enthält, sowie einer komprimierten Datei (ZIP-Format, Dateinamenserweiterung ist aber CAB), die nur heruntergeladen wird, wenn die lokalen Hilfeinformationen nicht auf dem aktuellen Stand sind.

**Bild 4.13**

Aktualisieren der Hilfe mit `Update-Help`



HINWEIS: Die Aktualisierung der Hilfedateien für alle Standardmodule, die sich im `c:\Windows\System32\WindowsPowerShell`-Verzeichnis befinden, ist nur mit administrativen Rechten möglich.

Listing 4.1 Beispiel für eine Help-Info-Datei

```

<?xml version="1.0" encoding="utf-8"?>
<HelpInfo xmlns="http://schemas.microsoft.com/powershell/help/2010/05">
<HelpContentURI>http://go.microsoft.com/fwlink/?linkid=210601</HelpContentURI>
  <SupportedUICultures>
    <UICulture>
      <UICultureName>en-US</UICultureName>
      <UICultureVersion>3.1.0.0</UICultureVersion>
    </UICulture>
  </SupportedUICultures>
</HelpInfo>
  
```

`Update-Help` kann durch Angabe eines Modulnamens im Parameter `-Module` die Hilfe für ein einzelnes Modul aktualisieren.

`Update-Help` kann durch Angabe eines Pfads im Parameter `-SourcePath` die Hilfedateien von einem lokalen Dateisystempfad oder Netzwerkpfad laden. Zu diesem Zweck kann man mit `Save-Help` die Help-Info-Dateien und die CAB-Dateien herunterladen. Größere Unternehmen können so die Hilfedateien zentral für alle Nutzer im Unternehmensnetzwerk bereitstellen.



TIPP: Die Aktualisierung der Hilfedateien kann auch im Editor „ISE“ im Menü „Hilfe“ ausgelöst werden.

4.11 Online-Hilfe

Die Dokumentation der PowerShell findet man hier: <https://docs.microsoft.com/de-de/powershell/>

Die zusätzlichen betriebssystemabhängigen PowerShell-Module sind hier dokumentiert: <https://docs.microsoft.com/de-de/powershell/module/>

Sie werden aber feststellen, dass dort jedes Commandlet einzeln beschrieben ist. Es gibt aber leider keine Dokumente, die das komplexere Zusammenspiel von Commandlets erklären oder die Vorgehensweise anhand von Praxisgebieten beschreiben wie in diesem Buch.



HINWEIS: Neu seit PowerShell 3.0 ist der Parameter `-Online` beim Commandlet `Get-Help`, der für ein Commandlet direkt die passende Seite in der Online-Hilfe öffnet.

Eine Online-Hilfe des Buchautors ist die Website www.dotnet-lexikon.de, wo Sie zu vielen Begriffen rund um PowerShell und .NET Erklärungstexte sowie ein Abkürzungsverzeichnis finden.

Das Bild zeigt einen Screenshot der Website www.dotnet-lexikon.de mit Suchergebnissen für den Begriff "powershell". Die Suchergebnisse sind in einer Liste dargestellt, die verschiedene PowerShell-Module und Konzepte enthält.

Oberthemen

- NET (DOTNET)
- NET Core (NETCORE)
- NET Framework (.NET FX)
- NET Framework 3.0 (.NET FX 3.0)
- NET Framework 3.5 (.NET FX 3.5)
- NET Framework 4.0 (.NET FX 4.0)
- NET Framework 4.5 (.NET 4.5)
- NET Framework 4.5.1 (.NET451)
- NET Framework 4.6 (.NET46)
- NET Framework Class Library (FCL)
- Active Server Pages .NET (ASPNET)
- ASP.NET Core
- Component Object Model (COM)
- Datenbank (DB)
- Java
- JavaScript (JS)
- Microsoft Azure
- Microsoft SQL Server (MSSQL)
- PowerShell (PS)
- Softwarekomponente
- Visual Studio (VS)
- Webframework
- Webservice
- Webtechniken
- Windows
- Windows 10
- Windows 8
- Windows PowerShell (WPS)
- Windows Runtime (WinRT)
- Windows Scripting (WS)
- Windows Server

Suchergebnisse

84 Einträge zum Thema 'powershell':

- NET Framework 3.0 (.NET FX 3.0)
- NET Framework 4.0 (.NET FX 4.0)
- AliasEigenschaft
- AppFabric
- AppX
- Aspx
- Ausführungsrichtlinie
- Biztalk Server
- Commandlet (Cmdlet)
- Commandlet Definition XML (CDXML)
- Desired State Configuration (DSC)
- Desired State Configuration Resource Kit
- Pipeline
- Powershell (PS)
- PowerShell Community Extensions (PSCX)
- PowerShell Core
- PowerShell Direct (PSDirect)
- PowerShell Gallery
- PowerShell Integrated Scripting Environment (ISE)
- PowerShell Language (PSL)
- PowerShell Package Manager
- PowerShell Provider
- PowerShell Script Analyzer
- Dublin
- edgeJS
- Eigenschaftssatz
- Exchange Management Shell (EMS)
- Extended Type System (ETS)
- Funktionsbasiertes Commandlets
- Internet Information Server 7.5
- Microsoft Assistance Markup Language (MAML)
- Microsoft Build-Konferenz 2016 (BUILD 2016)
- Microsoft Shell (MSH)
- Microsoft SQL Server 2008 (MSSQL08)
- Microsoft SQL Server 2012
- Monad
- NanoWBEM
- Navigationsprovider
- NotEigenschaft
- NuGet Library Package Manager (NuGet)
- Objekt-Pipelining
- OneGet
- Prester
- PrimalScript
- Regulärer Ausdruck (RA)
- Remote Server Management Tools (RSMT)
- SkriptEigenschaft
- Team Foundation Power Tools (TFPS)
- Team Foundation Server 2013 (TFS 2013)

IT-Visions.de

Schulung

Lassen Sie sich von Dr. Holger Schwichtenberg, Manfred Steyer, Bernd Marquardt und anderen Top-Experten weiterbilden.

Beratung

Langjährige Erfahrungen bei der Entwicklung von .NET-Anwendungen und dem Betrieb von Software auf der Microsoft-Plattform geben die Top-Experten von www.IT-Visions.de an Sie weiter.

Software-Entwicklung

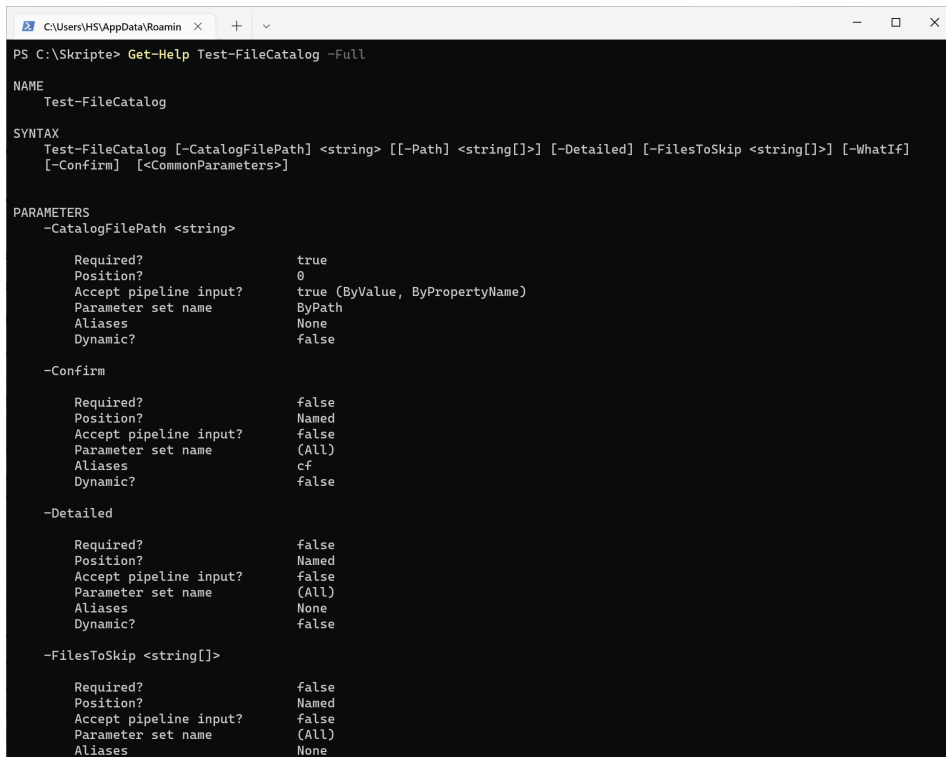
Sie brauchen Unterstützung bei der Entwicklung Ihrer Software? Die Experten-Entwickler von www.IT-Visions.de sind jederzeit online oder nach Pflichtenheft, bei Ihnen vor Ort oder zum Festpreis.

Inhalt Copyright © Dr. Holger Schwichtenberg 2002-2017 | Impressum, Datenschutz, Haftung | Kontakt

Bild 4.14 Hilfe zu den Fachbegriffen und Abkürzungen auf der Website www.dotnet-lexikon.de

4.12 Fehlende Hilfetexte

Leider gibt es nicht zu allen Commandlets eine Hilfe. Microsoft wird in seinem Softwareentwicklungsprozess immer agiler und vernachlässigt dabei leider die Dokumentation. So gibt es zum Beispiel zu einigen bereits in PowerShell 5.1 eingeführten Commandlets wie `Test-FileCatalog` einige Jahre später, zum Redaktionsschluss dieses Buchs immer noch keine adäquaten Hilfetexte, auch nicht in Windows 11. So zeigt `Get-Help` hier genau wie die Webseite nur die im Commandlet automatisch verfügbaren Metadaten über die Parameter, aber keinerlei Erläuterungstexte und keine Beispiele.



```

PS C:\Skripte> Get-Help Test-FileCatalog -Full

NAME
    Test-FileCatalog

SYNTAX
    Test-FileCatalog [-CatalogFilePath] <string> [[-Path] <string[]>] [-Detailed] [-FilesToSkip <string[]>] [-WhatIf]
    [-Confirm] [<CommonParameters>]

PARAMETERS
    -CatalogFilePath <string>

        Required?                true
        Position?                 0
        Accept pipeline input?    true (ByValue, ByPropertyName)
        Parameter set name       ByPath
        Aliases                   None
        Dynamic?                 false

    -Confirm

        Required?                false
        Position?                 Named
        Accept pipeline input?    false
        Parameter set name       (All)
        Aliases                   cf
        Dynamic?                 false

    -Detailed

        Required?                false
        Position?                 Named
        Accept pipeline input?    false
        Parameter set name       (All)
        Aliases                   None
        Dynamic?                 false

    -FilesToSkip <string[]>

        Required?                false
        Position?                 Named
        Accept pipeline input?    false
        Parameter set name       (All)
        Aliases                   None
  
```

Bild 4.15 Keine Hilfetexte zum Commandlet und zu den Parametern bei `Test-FileCatalog`

Namensraum `System.Diagnostics`. In dem Baum links erkennt man die verschiedenen Arten von Mitgliedern: *Methoden* (Methods), *Eigenschaften* (Properties) und *Ereignisse* (Events).

The screenshot shows the Microsoft documentation page for `System.Diagnostics.Process` in German. The left sidebar displays the class hierarchy, with `Process` selected. The main content area is titled 'Eigenschaften' (Properties) and lists several properties with their descriptions:

- Process()**: Initialisiert eine neue Instanz der `Process`-Klasse.
- BasePriority**: Ruft die Basispriorität des zugeordneten Prozesses ab.
- CanRaiseEvents**: Ruft einen Wert ab, der angibt, ob die Komponente ein Ereignis auslösen kann. (Geerbt von `Component`)
- Container**: Ruft den `IContainer` ab, der die `Component` enthält. (Geerbt von `Component`)
- DesignMode**: Ruft einen Wert ab, der angibt, ob sich `Component` gegenwärtig im Entwurfsmodus befindet. (Geerbt von `Component`)
- EnableRaisingEvents**: Ruft ab oder legt fest, ob beim Beenden des Prozesses das `Exited`-Ereignis ausgelöst werden soll.
- Events**: Ruft die Liste der Ereignishandler ab, die dieser `Component` angefügt sind. (Geerbt von `Component`)
- ExitCode**: Ruft den Wert ab, der vom zugeordneten Prozess beim Beenden angegeben wurde.
- ExitTime**: Ruft den Zeitpunkt ab, zu dem der zugeordnete Prozess beendet wurde.
- Handle**: Ruft das systemeigene Handle des zugeordneten Prozesses ab.
- HandleCount**: Ruft die Anzahl der vom Prozess geöffneten Handles ab.
- HasExited**: Ruft einen Wert ab, der angibt, ob der zugehörige Prozess beendet wurde.

Bild 4.17 Ausschnitt aus der Dokumentation der .NET-Klasse `System.Diagnostics.Process` (hier in der Online-Variante)



HINWEIS: Da die Dokumentation der .NET-Klassen für Softwareentwickler geschrieben wurde, ist sie häufig zu detailliert und nicht adäquat für PowerShell-Anwender. Es sind auch keine Beispiele in PowerShell-Syntax vorhanden. Leider hat Microsoft bisher noch nie eine für die Bedürfnisse von Administratoren angepasste Version der Klassenbibliotheksdokumentation veröffentlicht.



TIPP: Die englische Dokumentation ist der deutschen vorzuziehen, weil es in den deutschen Übersetzungen viele Übersetzungsfehler gibt, die das Verständnis erschweren.

5

Objektorientiertes Pipelining

Ihre Mächtigkeit entfaltet die PowerShell erst durch das objektorientierte Pipelining, also durch die Weitergabe von strukturierten Daten von einem Commandlet zum anderen.



HINWEIS: Dieses Kapitel setzt ein Grundverständnis des Konzepts der Objektorientierung voraus. Wenn Sie diese Grundkenntnisse nicht besitzen, lesen Sie bitte zuvor im Anhang den Crashkurs „Objektorientierung“ sowie den Crashkurs „.NET Framework“ oder vertiefende Literatur.

5.1 Befehlsübersicht

Die folgende Tabelle zeigt eine Übersicht der wichtigsten Commandlets, die Basisoperationen auf Pipelines ausführen. Diese Commandlets werden in den folgenden Kapiteln genau besprochen.

Tabelle 5.1 Übersicht über die wichtigsten Pipelining-Commandlets

Commandlet (mit Aliasen)	Bedeutung
Where-Object (where, ?)	Filtern mit Bedingungen
Select-Object (select)	Abschneiden der Ergebnismenge vorne/hinten bzw. Reduktion der Attribute der Objekte. Auch: Eliminieren von Duplikaten
Sort-Object (sort)	Sortieren der Objekte
Group-Object (group)	Gruppieren der Objekte
Foreach-Object { \$_... } (%)	Schleife über alle Objekte. Der Befehlsblock { ... } wird für jedes Objekt in der Pipeline einmal ausgeführt.
Get-Member (gm)	Ausgabe der Metadaten (Reflection)
Measure-Object (measure)	Berechnung: -min -max -sum -average
Compare-Object (compare, diff)	Vergleichen von zwei Objektmengen

■ 5.2 Pipeline-Operator

Für eine Pipeline wird – wie auch in Unix-Shells üblich und in der normalen Windows-Konsole möglich – der vertikale Strich „|“ (genannt „Pipe“ oder „Pipeline Operator“) verwendet.

```
Get-Process | Format-List
```

bedeutet, dass das Ergebnis des `Get-Process`-Commandlets an `Format-List` weitergegeben werden soll. Die Standardausgabeform von `Get-Process` ist eine Tabelle. Durch `Format-List` werden die einzelnen Attribute der aufzulistenden Prozesse untereinander statt in Spalten ausgegeben.

Die Pipeline kann beliebig lang sein, d. h., die Anzahl der Commandlets in einer einzigen Pipeline ist nicht begrenzt. Man muss aber jedes Mal den Pipeline-Operator nutzen, um die Commandlets zu trennen.

Ein Beispiel für eine komplexere Pipeline lautet:

```
Get-ChildItem w:\daten -r -filter *.doc  
| Where-Object { $_.Length -gt 40000 }  
| Select-Object Name, Length  
| Sort-Object Length  
| Format-List
```

`Get-ChildItem` ermittelt alle Microsoft-Word-Dateien im Ordner `w:\daten` und in seinen Unterordnern. Durch das zweite Commandlet (`Where-Object`) wird die Ergebnismenge auf diejenigen Objekte beschränkt, bei denen das Attribut `Length` größer ist als 40000. `$_` ist dabei der Zugriff auf das aktuelle Objekt in der Pipeline. Der Ausdruck `$.Length -gt 40000` ruft aus dem aktuellen Objekt die Eigenschaft `Length` ab und vergleicht, ob diese größer (`-gt`) als 40000 ist. `Select-Object` beschneidet alle Attribute aus `Name` und `Length`. Durch das vierte Commandlet in der Pipeline wird die Ausgabe nach dem Attribut `Length` sortiert. Das letzte Commandlet schließlich erzwingt eine Listendarstellung.

Nicht alle Aneinanderreihungen von Commandlets ergeben einen Sinn. Einige Aneinanderreihungen sind auch gar nicht erlaubt. Die Reihenfolge der einzelnen Befehle in der Pipeline ist nicht beliebig. Keineswegs kann man im obigen Befehl die Sortierung hinter die Formatierung setzen, weil nach dem Formatieren zwar noch ein Objekt existiert, dieses aber einen Textstrom repräsentiert. `Where-Object` und `Sort-Object` könnte man vertauschen; aus Gründen des Ressourcenverbrauchs sollte man aber erst einschränken und dann die verringerte Liste sortieren. Ein Commandlet kann aus vorgenannten Gründen erwarten, dass es bestimmte Arten von Eingabeobjekten gibt. Am besten sind aber Commandlets, die jede Art von Eingabeobjekt verarbeiten können.

Eine automatische Optimierung der Befehlsfolge wie in der Datenbankabfrage SQL gibt es bei PowerShell nicht.

Seit PowerShell-Version 3.0 hat Microsoft für den Zugriff auf das aktuelle Objekt der Pipeline zusätzlich zum Ausdruck `$_` den Ausdruck `$PSItem` eingeführt. `$_` und `$PSItem` sind synonym. Microsoft hat `$PSItem` eingeführt, weil einige Benutzer das Feedback gaben, dass `$_` zu (Zitat) „magisch“ sei.



ACHTUNG: Die PowerShell erlaubt beliebig lange Pipelines und es gibt auch Menschen, die sich einen Spaß daraus machen, möglichst viel durch eine einzige Befehlsfolge mit sehr vielen Pipes auszudrücken. Solche umfangreichen Befehlsfolgen sind aber meist für andere Menschen extrem schlecht lesbar. Bitte befolgen Sie daher den folgenden Ratschlag: Schreiben Sie nicht alles in eine einzige Befehlsfolge, nur weil es geht. Teilen Sie besser die Befehlsfolgen nach jeweils drei bis vier Pipe-Symbolen durch den Einsatz von Variablen auf (wird in diesem Kapitel auch beschrieben!) und lassen Sie diese geteilten Befehlsfolgen dann besser als PowerShell-Skripte ablaufen (siehe das Kapitel „PowerShell-Skripte“).

■ 5.3 .NET-Objekte in der Pipeline

Objektorientierung ist die herausragende Eigenschaft der PowerShell: Commandlets können durch Pipelines mit anderen Commandlets verbunden werden. Anders als Pipelines in Unix-Shells tauschen die Commandlets der PowerShell keine Zeichenketten, sondern typisierte .NET-Objekte aus. Das objektorientierte Pipelining ist im Gegensatz zum in den Unix-Shells und in der normalen Windows-Shell (*cmd.exe*) verwendeten zeichenkettenbasierten Pipelining nicht abhängig von der Position der Informationen in der Pipeline.

Ein Commandlet kann auf alle Attribute und Methoden der .NET-Objekte, die das vorhergehende Commandlet in die Pipeline gelegt hat, zugreifen. Die Mitglieder der Objekte können entweder durch Parameter der Commandlets (z. B. in `Sort-Object Length`) oder durch den expliziten Verweis auf das aktuelle Pipeline-Objekt (`$_`) in einer Schleife oder Bedingung (z. B. `Where-Object { $_.Length -gt 40000 }`) genutzt werden.

In einer Pipeline wie

```
Get-Process | Where-Object { $_.name -eq "iexplore" } | Format-Table ProcessName, WorkingSet64
```

ist das dritte Commandlet daher nicht auf eine bestimmte Anordnung und Formatierung der Ausgabe von vorherigen Commandlets angewiesen, sondern es greift über den sogenannten Reflection-Mechanismus (den eingebauten Komponentenerforschungsmechanismus des .NET Frameworks) direkt auf die Eigenschaften der Objekte in der Pipeline zu.



HINWEIS: Genau genommen bezeichnet Microsoft das Verfahren als „Extended Reflection“ bzw. „Extended Type System (ETS)“, weil die PowerShell in der Lage ist, Objekte um zusätzliche Eigenschaften anzureichern, die in der Klassendefinition gar nicht existieren.

Im obigen Beispiel legt `Get-Process` ein .NET-Objekt der Klasse `System.Diagnostics.Process` für jeden laufenden Prozess in die Pipeline. `System.Diagnostics.Process` ist eine Klasse aus der .NET-Klassenbibliothek. Commandlets können aber jedes beliebige .NET-Objekt in die Pipeline legen, also auch einfache Zahlen oder Zeichenketten, da es in .NET

keine Unterscheidung zwischen elementaren Datentypen und Klassen gibt. Eine Zeichenkette in die Pipeline zu legen, wird aber in der PowerShell die Ausnahme bleiben, denn der typisierte Zugriff auf Objekte ist wesentlich robuster gegenüber möglichen Änderungen als die Zeichenkettenauswertung mit regulären Ausdrücken.

Deutlicher wird der objektorientierte Ansatz, wenn man als Attribut keine Zeichenkette heranzieht, sondern eine Zahl. `WorkingSet64` ist ein 64 Bit langer Zahlenwert, der den aktuellen Speicherverbrauch eines Prozesses repräsentiert. Der folgende Befehl liefert alle Prozesse, die aktuell mehr als 20 Megabyte verbrauchen:

```
Get-Process | Where-Object { $_.WorkingSet64 -gt 20*1024*1024 }
```

Anstelle von `20*1024*1024` hätte man auch das Kürzel „20MB“ einsetzen können. Außerdem kann man `Where-Object` mit einem Fragezeichen abkürzen. Die kurze Variante des Befehls wäre dann also:

```
ps | ? { $_.ws -gt 20MB }
```

Wenn nur ein einziges Commandlet angegeben ist, dann wird das Ergebnis auf dem Bildschirm ausgegeben. Auch wenn mehrere Commandlets in einer Pipeline zusammengeschaltet sind, wird das Ergebnis des letzten Commandlets auf dem Bildschirm ausgegeben. Wenn das letzte Commandlet keine Daten in die Pipeline wirft, erfolgt keine Ausgabe.

5.4 Pipeline Processor

Für die Übergabe der .NET-Objekte zwischen den Commandlets sorgt der *PowerShell Pipeline Processor* (siehe folgende Grafik). Die Commandlets selbst müssen sich weder um die Objektweitergabe noch um die Parameterauswertung kümmern.

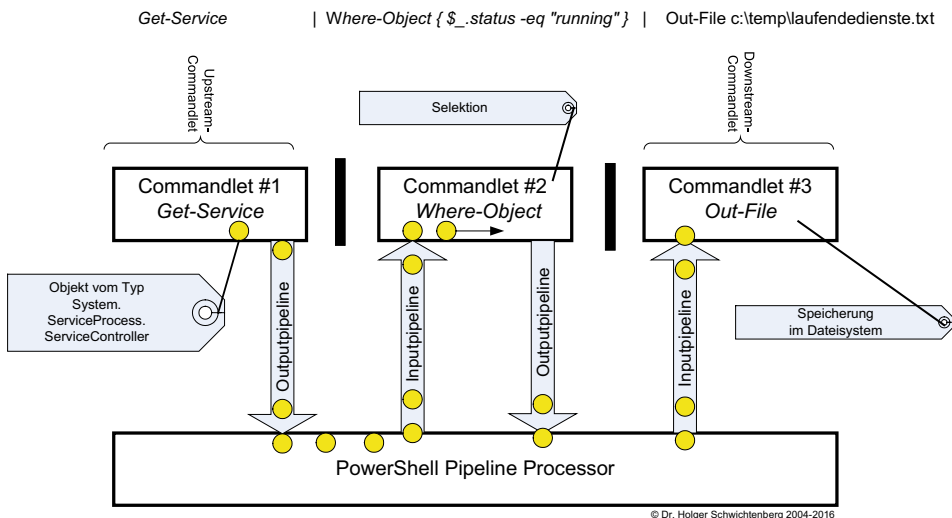


Bild 5.1 Der Pipeline Processor befördert die Objekte vom Upstream-Commandlet zum Downstream-Commandlet. Die Verarbeitung ist in der Regel asynchron.

Wie das obige Bild schon zeigt, beginnt ein nachfolgendes Commandlet mit seiner Arbeit, sobald es ein erstes Objekt aus der Pipeline erhält. Das Objekt durchläuft die komplette Pipeline. Erst dann wird das nächste Objekt vom ersten Commandlet abgeholt. Man nennt dies „Streaming-Verarbeitung“. Streaming-Verarbeitung ist schneller als die klassische sequentielle Verarbeitung, weil die folgenden Commandlets in der Pipeline nicht auf vorhergehende warten müssen.



HINWEIS: Intern arbeitet die einem Thread, d. h. es findet keine parallele Verarbeitung mehrerer Befehle statt. Erst seit PowerShell 7.0 gibt es mit dem Parameter `-parallel` bei `Foreach-Command` eine einfache Möglichkeit, jedes Objekt in einem eigenen Thread zu verarbeiten.

Aber nicht alle Commandlets beherrschen die asynchrone Streaming-Verarbeitung. Commandlets, die alle Objekte naturgemäß erst mal kennen müssen, bevor sie überhaupt ihren Zweck erfüllen können (z. B. `Sort-Object` zum Sortieren und `Group-Object` zum Gruppieren), blockieren die asynchrone Verarbeitung.



HINWEIS: Es gibt auch einige Commandlets, die zwar asynchron arbeiten könnten, aber leider nicht so programmiert wurden, um dies zu unterstützen.

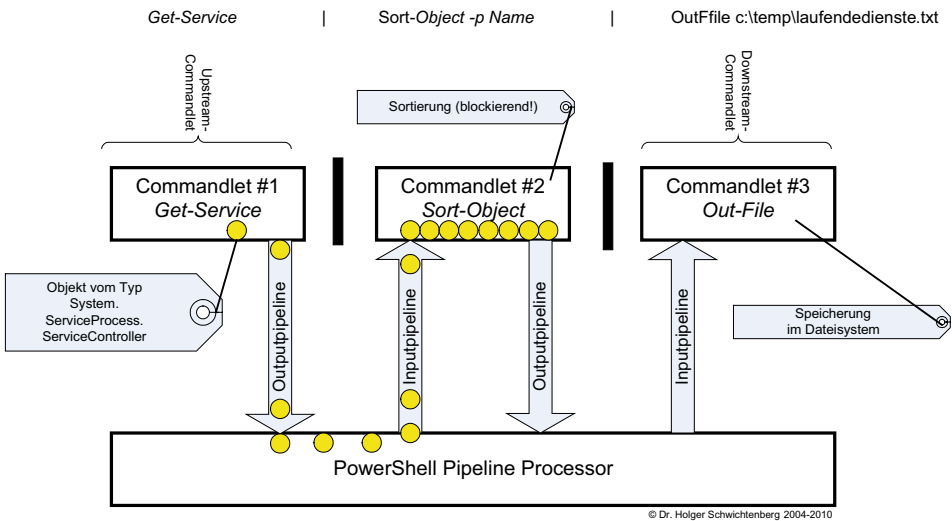


Bild 5.2 `Sort-Object` blockiert die direkte Weitergabe. Erst wenn alle Objekte angekommen sind, kann das Commandlet sortieren.

Auch bei Commandlets, die Streaming-Verarbeitung unterstützen kann der PowerShell-Nutzer mit dem allgemeinen Parameter `-OutBuffer` (abgekürzt `-ob`), das jedes Commandlet anbietet, dafür sorgen, dass eine bestimmte Anzahl von Objekten angesammelt wird bevor eine Weitergabe an das nachfolgende Commandlet erfolgt.

Im Standard beginnt die Ausgabe der Ordner- und Dateinamen sofort:

```
dir c:\ -Recurse | ft name
```

In diesem Fall passiert lange nichts, bevor die Ausgabe beginnt:

```
dir c:\ -Recurse -OutBuffer:100000 | ft name
```

■ 5.5 Pipelining von Parametern

Die Pipeline kann jegliche Art von Information befördern, auch einzelne elementare Daten. Einige Commandlets unterstützen es, dass auch die Parameter aus der Pipeline ausgelesen werden. Der folgende Pipeline-Befehl führt zu einer Auflistung aller Windows-Systemdienste, die mit dem Buchstaben „I“ beginnen.

```
"i*" | Get-Service
```

Die folgende Abbildung zeigt einige Parameter des Commandlets Get-Service. Diese Liste erhält man durch den Befehl `Get-Help Get-Service -Parameter *`.

```
-Include <string[]>
  Retrieves only the specified services. The value of this parameter qualifies the Name parameter. Enter a name element or pattern, such as "s*". Wildcards are permitted.

  Required?                false
  Position?                named
  Default value            named
  Accept pipeline input?   false
  Accept wildcard characters? false

-InputObject <ServiceController[]>
  Specifies ServiceController objects representing the services to be retrieved. Enter a variable that contains the objects, or type a command or expression that gets the objects. You can also pipe a service object to Get-Service.

  Required?                false
  Position?                named
  Default value            named
  Accept pipeline input?   true (ByValue)
  Accept wildcard characters? false

-Name <string[]>
  Specifies the service names of services to be retrieved. Wildcards are permitted. By default, Get-Service gets all of the services on the computer.

  Required?                false
  Position?                1
  Default value            1
  Accept pipeline input?   true (ByValue, ByPropertyName)
  Accept wildcard characters? true

-RequiredServices [(SwitchParameter)]
  Gets only the services that this service requires.

  This parameter gets the value of the ServicesDependedOn property of the service. By default, Get-Service gets all services.

  Required?                false
  Position?                named
  Default value            False
  Accept pipeline input?   false
  Accept wildcard characters? false
```

Bild 5.3 Hilfe zu den Parametern des Commandlets Get-Service

Interessant sind die mit Pfeil markierten Stellen. Nach „Accept pipeline Input“ kann man jeweils nachlesen, ob der Parameter des Commandlets aus den vorhergehenden Objekten in der Pipeline „befüttert“ werden kann.

Bei „-Name“ steht ByValue und ByPropertyName. Dies bedeutet, dass der Name sowohl das ganze Objekt in der Pipeline sein darf als auch Teil eines Objekts.

Im Fall von

```
"BITS" | Get-Service
```

ist der Pipeline-Inhalt eine Zeichenkette (ein Objekt vom Typ String), die als Ganzes auf Name abgebildet werden kann.

Es funktioniert aber auch folgender Befehl, der alle Dienste ermittelt, deren Name genauso lautet wie der Name eines laufenden Prozesses:

```
Get-Process | Get-Service -ea silentlycontinue | ft name
```

Dies funktioniert über die zweite Option (ByPropertyName), denn Get-Process liefert Objekte des Typs Process, die ein Attribut namens Name haben. Der Parameter Name von Get-Service wird auf dieses Name-Attribut abgebildet.

Beim Parameter -InputObject ist hingegen nur „ByValue“ angegeben. Hier erwartet Get-Service gerne Instanzen der Klasse ServiceController. Es gibt aber keine Objekte, die ein Attribut namens InputObject haben, in dem dann ServiceController-Objekte stecken.

Zahlreiche Commandlets besitzen einen Parameter -InputObject, insbesondere die allgemeinen Verarbeitungs-Commandlets wie Where-Object, Select-Object und Measure-Object, die Sie im nächsten Kapitel kennenlernen werden. Der Name -InputObject ist eine Konvention.

```
PS P:\> Get-Help Where-Object -Parameter *
-FilterScript <scriptblock>
  Specifies the script block that is used to filter the objects. Enclose the
  script block in braces < > .
  Required?                true
  Position?                1
  Default value
  Accept pipeline input?   false
  Accept wildcard characters? false

-InputObject <psobject>
  Specifies the objects to be filtered. You can also pipe the objects to Where-Object.
  Required?                false
  Position?                named
  Default value
  Accept pipeline input?   true <ByValue>
  Accept wildcard characters? false
```

Bild 5.4 Parameter des Commandlets Where-Object

Leider geht es nicht bei allen Commandlets so einfach mit der Parameterübergabe. Man nehme zum Beispiel das Commandlet Test-Connection, das prüft, ob ein Computer per Ping erreichbar ist.

Der normale Aufruf mit Parameter ist:

```
Test-Connection -computername Server123
```

oder ohne benannten Parameter

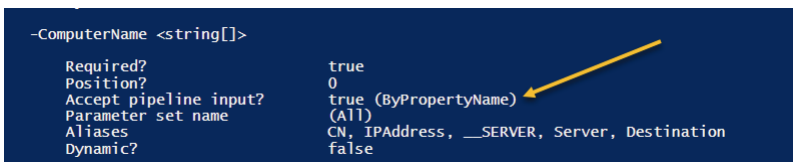
```
Test-Connection Server123
```

Nun könnte man auf die Idee kommen, hier den Computernamen genau so zu übergeben, wie den Namen bei Get-Service. Allerdings liefert "Server123" | Test-Connection den Fehler: *"The input object cannot be bound to any parameters for the command either because the command does not take pipeline input or the input and its properties do not match any of the parameters that take pipeline input."*

Warum das nicht geht, kann man in der Hilfe zum Parameter ComputerName des Commandlets Test-Connection erkennen. Dort steht, dass ComputerName nur als „ByPropertyName“ akzeptiert wird und nicht wie beim Parameter Name beim Commandlet Get-Service auch „ByValue“. Das bedeutet also, dass man erst ein Objekt mit der Eigenschaft ComputerName konstruieren und dann übergeben muss:

```
New-Object psobject -Property @{Computername="Server123"} | Test-Connection
```

Das funktioniert zwar, ist aber hässlich und umständlich. Warum Test-Connection und einige andere Commandlets die Eingaben nicht „ByValue“ unterstützen, wusste übrigens das PowerShell-Entwicklungsteam auf Nachfrage auch nicht zu beantworten. Die Schuld liegt hier vermutlich bei dem einzelnen Entwickler bei Microsoft, der die Commandlets implementiert hat.



```
-ComputerName <string[]>
Required?           true
Position?          0
Accept pipeline input? true (ByPropertyName)
Parameter set name (All)
Aliases            CN, IPAddress, __SERVER, Server, Destination
Dynamic?           false
```

Bild 5.5 Hilfe zum Parameter ComputerName des Commandlets Test-Connection

■ 5.6 Pipelining von klassischen Befehlen

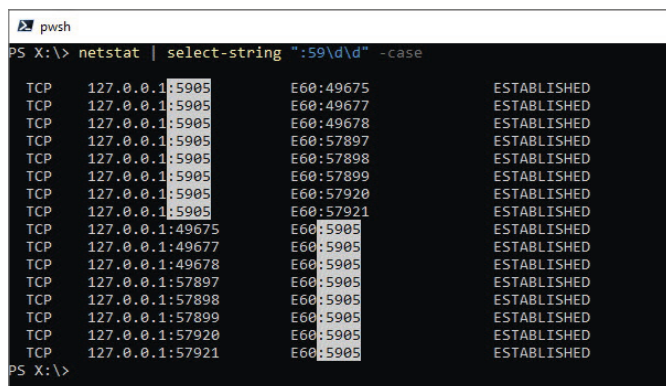
Grundsätzlich dürfen auch klassische Kommandozeilenanwendungen in der PowerShell verwendet werden. Wenn man einen Befehl wie netstat.exe oder ping.exe ausführt, dann legen diese eine Menge von Zeichenketten in die Pipeline: Jede Ausgabezeile ist eine Zeichenkette.

Diese Zeichenketten kann man sehr gut mit dem Commandlet Select-String auswerten. Select-String lässt nur diejenigen Zeilen die Pipeline passieren, die auf den angegebenen regulären Ausdruck zutreffen.



TIPP: Die Syntax der regulären Ausdrücke in .NET wird im Kapitel „PowerShell-Skriptsprache“ noch etwas näher beschrieben werden.

In dem folgenden Beispiel werden nur diejenigen Zeilen der Ausgabe von `netstat.exe` gefiltert, die einen Doppelpunkt gefolgt von den Ziffern 59 und zwei weiteren Ziffern enthalten. Die Hervorhebung der Treffer durch Negativschrift gibt es erst seit PowerShell 7.0.



```

pwsh
PS X:\> netstat | select-string ":59\d\d" -case
TCP    127.0.0.1:5905           E60:49675           ESTABLISHED
TCP    127.0.0.1:5905           E60:49677           ESTABLISHED
TCP    127.0.0.1:5905           E60:49678           ESTABLISHED
TCP    127.0.0.1:5905           E60:57897           ESTABLISHED
TCP    127.0.0.1:5905           E60:57898           ESTABLISHED
TCP    127.0.0.1:5905           E60:57899           ESTABLISHED
TCP    127.0.0.1:5905           E60:57920           ESTABLISHED
TCP    127.0.0.1:5905           E60:57921           ESTABLISHED
TCP    127.0.0.1:49675         E60:5905            ESTABLISHED
TCP    127.0.0.1:49677         E60:5905            ESTABLISHED
TCP    127.0.0.1:49678         E60:5905            ESTABLISHED
TCP    127.0.0.1:57897         E60:5905            ESTABLISHED
TCP    127.0.0.1:57898         E60:5905            ESTABLISHED
TCP    127.0.0.1:57899         E60:5905            ESTABLISHED
TCP    127.0.0.1:57920         E60:5905            ESTABLISHED
TCP    127.0.0.1:57921         E60:5905            ESTABLISHED
PS X:\>

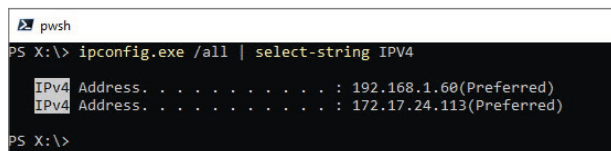
```

Bild 5.6

Einsatz von `Select-String` zur Filterung von Ausgaben klassischer Kommandozeilenwerkzeuge

Ein weiteres Beispiel ist das Filtern der Ausgaben von `ipconfig.exe`. Der nachfolgende Befehl liefert nur die Zeilen zum Thema IPv4:

```
ipconfig.exe /all | select-string IPv4
```



```

pwsh
PS X:\> ipconfig.exe /all | select-string IPv4
IPv4 Address. . . . . : 192.168.1.60(Preferred)
IPv4 Address. . . . . : 172.17.24.113(Preferred)
PS X:\>

```

Bild 5.7

Abbildung: Ausführung des obigen Befehls

Es gibt aber leider klassische Kommandozeilenbefehle, die inhaltliche Informationen über Farben statt über Texte transportieren. Ein schlechtes Beispiel ist hier:

```
git branch -a
```

Der Befehl `git branch -a` liefert eine Liste aller Git-Branches in einem lokalen Git-Repository als farblich verschieden markierte Textzeilen.



```

T:\CC2 [master =>]> git branch -a
* master
remotes/GITHUB/Feature1
remotes/GITHUB/master
remotes/GITHUB/F2
remotes/GITHUB/Feature1
remotes/GITHUB/Feature2
remotes/GITHUB/Feature3
remotes/GITHUB/HEAD -> GITHUB/master
remotes/GITHUB/master

```

Eine schwarze Ausgabe (erste beide Zeilen) bedeutet, dass es für den Remote-Branch auch einen lokalen Branch gibt. Eine rote Ausgabe (Zeile 3 bis 8, hier im Buch aufgrund des Schwarz-Weiß-Drucks leider nicht zu sehen) bedeutet dabei, dass ein Remote-Branch noch kein lokales Äquivalent besitzt.

Man kann diesen Befehl zwar in der PowerShell ausführen und sieht dort auch die Farben. Aber eine Weiterverarbeitung per Pipeline mit dem Ziel „Lege einen lokalen Branch an für alle Branches, die lokal noch nicht existieren“, ist nicht möglich.

Man kann lediglich `git branch` für alle ausführen. Hierbei muss man nicht nur filtern, sondern auch mit `Trim()` die Leerzeichen zu Beginn eliminieren:

```
git branch -a | ? { $_ -like "*remotes*" -and $_ -notlike "*HEAD*" } | % { git branch --track ${remote#origin/} $_.Trim() }
```

oder

```
git branch -a | sls -pattern "remotes" | sls -pattern "HEAD" -NotMatch | % { git branch --track ${remote#origin/} $_.Line.Trim() }
```

Man bekommt aber immer eine Fehlermeldung für die schon existierenden lokalen Branches.

```
T:\CC2 [master #]> git branch -a | ? { $_ -like "*remotes*" -and $_ -notlike "*HEAD*" } | % { git branch --track ${remote#origin/} $_.Trim() }
fatal: A branch named 'remotes/GITHUB/Feature1' already exists.
fatal: A branch named 'remotes/GITHUB/master' already exists.
Branch 'remotes/GITHUB/F2' set up to track local branch 'master'.
fatal: A branch named 'remotes/GITHUB/Feature1' already exists.
Branch 'remotes/GITHUB/Feature2' set up to track local branch 'master'.
Branch 'remotes/GITHUB/Feature3' set up to track local branch 'master'.
fatal: A branch named 'remotes/GITHUB/master' already exists.
```

■ 5.7 Zeilenumbrüche in Pipelines

Wenn sich ein Pipeline-Befehl über mehrere Zeilen erstrecken soll, kann man dies auf mehrere Weisen bewerkstelligen:

- Man beendet die Zeile mit einem Pipe-Symbol `[]` und drückt EINGABE. PowerShell-Standardkonsole und PowerShell-ISE-Konsole erkennen, dass der Befehl noch nicht abgeschlossen ist, und erwarten weitere Eingaben. Die Standardkonsole zeigt dies auch mit `>>>` an.
- Man kann am Ende einer Zeile mit einem Gravis `[`]`, ASCII-Code 96, bewirken, dass die nächste Zeile mit zum Befehl hinzugerechnet wird (Zeilenumbruch in einem Befehl). Das funktioniert in allen PowerShell-Hosts und auch in PowerShell-Skripten.

```
PS T:\> Get-Process p* | Sort-Object WorkingSet |
>> Format-Table id,name,WorkingSet

Id Name           WorkingSet
----
10828 powershell     92942336
15340 powershell_ise 220946432
1804 powershell     83664896
4040 powershell     76177408

PS T:\> _
```

Bild 5.8

Zeilenumbruch nach Pipeline-Symbol

■ 5.8 Schleifen

Ein wichtiges Commandlet ist

```
Foreach-Object { $_... }
```

Alias:

```
% { $_... }
```

Foreach-Object führt eine Schleife (Iteration) über alle Objekte in der Pipeline aus. Der Befehlsblock { ... } wird für jedes Objekt in der Pipeline einmal ausgeführt. Das jeweils aktuelle Objekt, das an der Reihe ist, erhält man über die eingebaute Variable `$_`. `$_` ist die Abkürzung für `$PSItem`. Beide Schreibweisen haben die gleiche Funktion.

5.8.1 Notwendigkeit für Foreach-Object

Der Einsatz von Foreach-Object ist in Pipelines nicht notwendig, wenn das nachfolgende Commandlet die Objekte des vorherigen Commandlets direkt verarbeiten kann.

Beispiele:

```
Get-ChildItem Bu* | Remove-Item
Get-Service BI* | Start-Service
Get-Process chrome | Stop-Process
```

Gleichwohl könnte man in diesen Fällen Foreach-Object einsetzen, was den Befehl aber verlängert:

```
Get-ChildItem Bu* | Foreach-Object { Remove-item $_.FullName }
Get-Service BI* | Foreach-Object { Start-Service $_ }
Get-Process chrome | Foreach-Object { Stop-Process $_ }
```

Es liegt an den Eigenarten des jeweiligen Commandlets, ob sie als Standardparameter das gesamte Objekt (`$_`) oder eine bestimmte Eigenschaft (`$_.Fullname`) erwarten.

In manchen Situationen ist der Einsatz von Foreach-Object aber auch nicht möglich, denn man will mit Sort-Object die ganze Menge sortieren und nicht jedes Objekt einzeln:

```
"----- richtig:"
Get-Service x* | Sort-Object name
"----- falsch:"
Get-Service x* | Foreach-Object { Sort-Object $_.Name }
```

Schließlich gibt es Fälle, in denen Foreach-Object zwingend eingesetzt werden muss. Dies gilt insbesondere, wenn das nachfolgende Commandlet die Objekte nicht verarbeiten kann. Zudem quittiert die PowerShell diesen Befehl

```
Get-Service BI* | Write-Host $_.DisplayName -ForegroundColor yellow
```

mit dem Laufzeitfehler „The input object cannot be bound to any parameters for the command either because the command does not take pipeline input or the input and its properties do not“.

Richtig ist:

```
Get-Service BI* | foreach-object { Write-Host $_.DisplayName -ForegroundColor Yellow }
```

Ebenso ist Foreach-Object notwendig, wenn mehrere Befehle (also ganzer Befehlsblock) ausgeführt werden sollen. Befehlsblöcke werden in den Kapiteln „PowerShell-Skripte“ und „PowerShell-Skriptspache“ erläutert.

```
Get-Service BI* | foreach-object {  
    if ($_.Status -eq "Stopped")  
    {  
        Write-Host "Beendet Dienst " $_.DisplayName -ForegroundColor Yellow  
        Start-Service $_  
    }  
    else  
    {  
        Write-Host "Starte Dienst " $_.DisplayName -ForegroundColor Yellow  
        Stop-Service $_  
    }  
}
```

5.8.2 Parallelisierung mit Multithreading

In PowerShell 1.0 bis 6.2 erfolgt die Ausführung im Hauptthread der PowerShell, d. h., die einzelnen Durchläufe erfolgen nacheinander. Seit PowerShell 7.0 kann man mit dem Parameter `-parallel` die Ausführung auf verschiedene Threads parallelisieren (via Multithreading), sodass bei längeren Operationen in Summe das Ergebnis schneller vorliegt.



ACHTUNG: Die Multithreading hat immer einigen Overhead. Die Parallelisierung lohnt sich nur bei länger dauernden Operationen. Bei kurzen Operationen ist der Zeitverlust durch die Erzeugung und Vernichtung der Threads höher als der Zeitgewinn durch die Parallelisierung.

Das folgende Beispiel zeigt zwei Varianten der Abfrage, ob die Software „Classic Shell“ auf drei verschiedenen Computern installiert ist. Bei der ersten Variante ohne `-parallel` wird die leider etwas langwierige Abfrage der WMI-Klasse `Win32_Product` auf den drei Computern nacheinander in dem gleichen Thread ausgeführt. Bei der zweiten Variante mit `-parallel` wird die Abfrage parallel in drei verschiedenen Threads gestartet! Die Parallelisierung ist erst möglich seit PowerShell 7.0.



TIPP: Die Nummer des Threads fragt man ab mit der .NET-Klasse `Thread: [System.Threading.Thread]::CurrentThread.ManagedThreadId`

Listing 5.1 [\PowerShell\1_Basiswissen\Pipelining\Schleifen.ps1]

```

Write-Host "# ForEach-Object ohne -parallel" -ForegroundColor Yellow
"E27","E29","E44" | ForEach-Object {
    "Abfrage bei Computer $_ in Thread $($([System.Threading.Thread]::CurrentThread.
ManagedThreadId)"
    $e = Get-CimInstance -Class Win32_
Product -Filter "Name='Classic Shell'" -computername $_
    if ($e -eq $null) { "Kein Ergebnis bei $_!"}
    else { $e }
}
Write-Host ""
Write-Host " # ForEach-Object mit -parallel" -ForegroundColor Yellow
"E27","E29","E44" | ForEach-Object -parallel {
    "Abfrage bei Computer $_ in Thread $($([System.Threading.Thread]::CurrentThread.
ManagedThreadId)"
    $e = Get-CimInstance -Class Win32_
Product -Filter "Name='Classic Shell'" -computername $_
    if ($e -eq $null) { "Kein Ergebnis bei $_!"}
    else { $e }
}
# ohne Read-
Host würde das Skript die später eingehenden Ergebnisse nicht mehr anzeigen!
read-host

```

```

# ForEach-Object ohne -parallel
Abfrage bei Computer E27 in Thread 19
-----
Name           Caption           Vendor           Version           IdentifyingNumber           PSComputerName
-----
Classic Shell  Classic Shell     IvoSoft          4.1.0             {840C85B7-D3D6-4143-9AF9-DAE88FD... E27
Abfrage bei Computer E29 in Thread 19
Classic Shell  Classic Shell     IvoSoft          4.1.0             {840C85B7-D3D6-4143-9AF9-DAE88FD... E29
Abfrage bei Computer E44 in Thread 19
Kein Ergebnis bei E44!

# ForEach-Object mit -parallel
Abfrage bei Computer E27 in Thread 80
Abfrage bei Computer E29 in Thread 94
Abfrage bei Computer E44 in Thread 96
Kein Ergebnis bei E44!
Classic Shell  Classic Shell     IvoSoft          4.1.0             {840C85B7-D3D6-4143-9AF9-DAE88FD... E29
Classic Shell  Classic Shell     IvoSoft          4.1.0             {840C85B7-D3D6-4143-9AF9-DAE88FD... E27

```

Bild 5.9 Parallelität bei Foreach-Object in PowerShell 7

Die Anzahl der Threads, die Foreach-Object nutzen soll, kann man mit dem Parameter `-ThrottleLimit` begrenzen:

```

1..20 | ForEach-Object -parallel {
    Write-host "Objekt #$_ in Thread $($([System.Threading.Thread]::CurrentThread.
ManagedThreadId)"
    sleep -Seconds 2 } -ThrottleLimit 5

```

■ 5.9 Zugriff auf einzelne Objekte aus einer Menge

Es ist möglich, gezielt einzelne Objekte über ihre Position (Index) in der Pipeline anzusprechen. Die Positionsangabe ist in eckige Klammern zu setzen und die Zählung beginnt bei 0. Der Pipeline-Ausdruck ist in runde Klammern zu setzen.

Beispiele:

Der erste Prozess:

```
(Get-Process)[0]
```

Der dreizehnte Prozess:

```
(Get-Process)[12]
```

Alternativ kann man dies auch mit `Select-Object` unter Verwendung der Parameter `-First` und `-Skip` ausdrücken:

```
(Get-Process i* | Select-Object -first 1).name
(Get-Process i* | Select-Object -skip 12 -first 1).name
```



HINWEIS: Während `(Get-Date)[0]` in PowerShell vor Version 3.0 zu einem Fehler führt („Unable to index into an object of type System.DateTime.“), weil `Get-Date` keine Menge liefert, ist der Befehl seit PowerShell-Version 3.0 in Ordnung und liefert das gleiche Ergebnis wie `Get-Date`, da die PowerShell seit Version 3.0 ja aus Benutzersicht ein einzelnes Objekt und eine Menge von Objekten gleich behandelt. `(Get-Date)[1]` liefert dann natürlich kein Ergebnis, weil es kein zweites Objekt in der Pipeline gibt.

Die Positionsangaben kann man natürlich mit Bedingungen kombinieren. So liefert dieser Befehl den dreizehnten Prozess in der Liste der Prozesse, die mehr als 20 MB Hauptspeicher brauchen:

```
(Get-Process | where-object { $_.WorkingSet64 -gt 20mb } )[12]
```

```
PS C:\Windows\System32> (get-process)[0]
Handles  NPM(K)  PM(K)  WS(K)  UM(M)  CPU(s)  Id  ProcessName
-----  -
20      2      1968   2664   17     0,03    2784 cmd

PS C:\Windows\System32> (get-process)[12]
Handles  NPM(K)  PM(K)  WS(K)  UM(M)  CPU(s)  Id  ProcessName
-----  -
69      9      1484   4196   41     0,03    2100 dlpwdnt

PS C:\Windows\System32> (get-process | where-object { $_.WorkingSet64 -gt 20mb } )[12]
Handles  NPM(K)  PM(K)  WS(K)  UM(M)  CPU(s)  Id  ProcessName
-----  -
685     29     53924  59544  291    34,39   4984 powershell

PS C:\Windows\System32> .
```

Bild 5.10 Zugriff auf einzelne Prozessobjekte