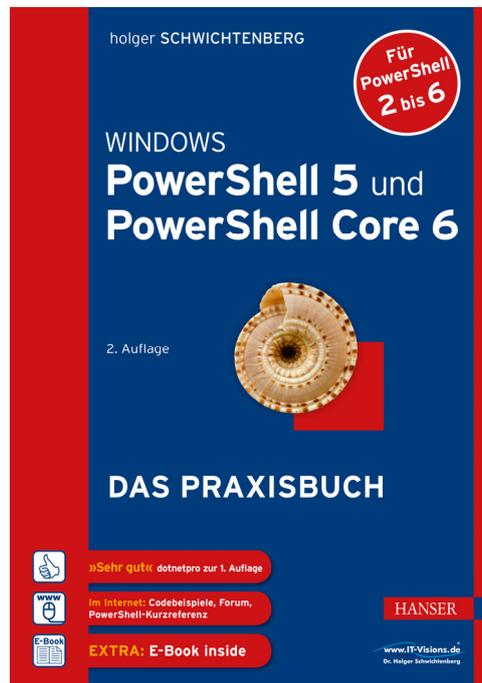


HANSER



Leseprobe

zu

„Windows PowerShell 5 und PowerShell Core 6“ (2. Auflage)

von Holger Schwichtenberg

ISBN (Buch): 978-3-446-45331-9

ISBN (E-Book): 978-3-446-45397-5

Weitere Informationen und Bestellungen unter
<http://www.hanser-fachbuch.de/9783446453319>

sowie im Buchhandel

© Carl Hanser Verlag München

Inhalt

Vorwort zur sechsten Auflage	XXIII
Über den Autor Dr. Holger Schwichtenberg	XXIX
Teil A: PowerShell-Basiswissen	1
1 Erste Schritte mit der PowerShell	3
1.1 Was ist die PowerShell?	3
1.2 Windows PowerShell versus PowerShell Core	4
1.3 Geschichte der PowerShell	4
1.4 Motivation zur PowerShell	6
1.5 Betriebssysteme mit vorinstallierter PowerShell	9
1.6 Windows PowerShell herunterladen und auf anderen Windows- Betriebssystemen installieren	10
1.7 Die Windows PowerShell testen	14
1.8 PowerShell Core installieren und testen	23
1.9 Woher kommen die Commandlets?	26
1.10 PowerShell Community Extensions (PSCX) herunterladen und installieren	27
1.11 Den Windows PowerShell-Editor „ISE“ verwenden	29
2 Architektur der PowerShell	33
3 Einzelbefehle der PowerShell	37
3.1 Commandlets	37
3.2 Aliase	50
3.3 Ausdrücke	58
3.4 Externe Befehle	59
3.5 Dateinamen	60

4	Hilfefunktionen	63
4.1	Auflisten der verfügbaren Befehle	63
4.2	Volltextsuche	65
4.3	Erläuterungen zu den Befehlen	66
4.4	Hilfe zu Parametern	67
4.5	Hilfe mit Show-Command	69
4.6	Hilfefenster	70
4.7	Allgemeine Hilfetexte	72
4.8	Aktualisieren der Hilfsdateien	73
4.9	Online-Hilfe	75
4.10	Fehlende Hilfetexte	76
4.11	Dokumentation der .NET-Klassen	77
5	Objektorientiertes Pipelining	81
5.1	Pipeline-Operator	81
5.2	.NET-Objekte in der Pipeline	82
5.3	Pipeline Processor.	84
5.4	Pipelining von Parametern	85
5.5	Pipelining von klassischen Befehlen	88
5.6	Anzahl der Objekte in der Pipeline	89
5.7	Zeilenumbrüche in Pipelines.	90
5.8	Zugriff auf einzelne Objekte aus einer Menge	90
5.9	Zugriff auf einzelne Werte in einem Objekt	92
5.10	Methoden ausführen	93
5.11	Analyse des Pipeline-Inhalts	95
5.12	Filtern	107
5.13	Zusammenfassung von Pipeline-Inhalten	110
5.14	„Kastrierung“ von Objekten in der Pipeline	111
5.15	Sortieren	112
5.16	Duplikate entfernen	113
5.17	Gruppierung	114
5.18	Berechnungen	116
5.19	Zwischenschritte in der Pipeline mit Variablen	116
5.20	Verzweigungen in der Pipeline	117
5.21	Vergleiche zwischen Objekten.	119
5.22	Zusammenfassung	120
5.23	Praxisbeispiele	121
6	PowerShell-Skripte	123
6.1	Skriptdateien.	123

6.2	Start eines Skripts.	125
6.3	Aliase für Skripte verwenden	126
6.4	Parameter für Skripte	127
6.5	Skripte dauerhaft einbinden (Dot Sourcing)	128
6.6	Das aktuelle Skriptverzeichnis	129
6.7	Sicherheitsfunktionen für PowerShell-Skripte	129
6.8	Anforderungsdefinitionen von Skripten	132
6.9	Skripte anhalten	132
6.10	Versionierung und Versionsverwaltung von Skripten	133
7	PowerShell-Skriptsprache	135
7.1	Hilfe zur PowerShell-Skriptsprache	135
7.2	Befehlstrennung	136
7.3	Kommentare	136
7.4	Variablen	137
7.5	Variablenbedingungen	147
7.6	Zahlen	148
7.7	Zeichenketten (Strings)	150
7.8	Reguläre Ausdrücke	159
7.9	Datum und Uhrzeit	166
7.10	Arrays	167
7.11	ArrayList	170
7.12	Assoziative Arrays (Hash-Tabellen)	171
7.13	Operatoren	172
7.14	Überblick über die Kontrollkonstrukte	177
7.15	Schleifen	177
7.16	Bedingungen	182
7.17	Unterroutinen (Prozedur/Funktionen)	184
7.18	Eingebaute Funktionen	191
7.19	Fehlerbehandlung	191
7.20	Objektorientiertes Programmieren mit Klassen	199
8	Ausgaben	203
8.1	Ausgabe-Commandlets	203
8.2	Benutzerdefinierte Tabellenformatierung	206
8.3	Benutzerdefinierte Listenausgabe	208
8.4	Mehrspaltige Ausgabe	208
8.5	Out-GridView	209
8.6	Standardausgabe	211
8.7	Einschränkung der Ausgabe	213

8.8	Seitenweise Ausgabe	213
8.9	Ausgabe einzelner Werte	214
8.10	Details zum Ausgabeoperator	216
8.11	Ausgabe von Methodenergebnissen und Unterobjekten in Pipelines	220
8.12	Ausgabe von Methodenergebnissen und Unterobjekten in Zeichenketten	220
8.13	Unterdrückung der Ausgabe	221
8.14	Ausgaben an Drucker	222
8.15	Ausgaben in Dateien	222
8.16	Umleitungen (Redirection)	223
8.17	Fortschrittsanzeige	223
8.18	Sprachausgabe	224
9	Das PowerShell-Navigationsmodell	227
9.1	Einführungsbeispiel: Navigation in der Registrierungsdatenbank	227
9.2	Provider und Laufwerke	228
9.3	Navigationsbefehle	231
9.4	Pfadangaben	231
9.5	Beispiel	233
9.6	Eigene Laufwerke definieren	234
10	Fernauführung (Remoting)	235
10.1	RPC-Fernabfrage ohne WS-Management	236
10.2	Anforderungen an PowerShell Remoting	237
10.3	Rechte für PowerShell-Remoting	238
10.4	Einrichten von PowerShell Remoting	239
10.5	Überblick über die Fernauführungs-Commandlets	241
10.6	Interaktive Fernverbindungen im Telnet-Stil	242
10.7	Fernauführung von Befehlen	243
10.8	Parameterübergabe an die Fernauführung	247
10.9	Fernauführung von Skripten	248
10.10	Ausführung auf mehreren Computern	249
10.11	Sitzungen	250
10.12	Implizites Remoting	255
10.13	Zugriff auf entfernte Computer außerhalb der eigenen Domäne	256
10.14	Verwaltung des WS-Management-Dienstes	259
10.15	PowerShell Direct für Hyper-V	261
10.16	Praxisbeispiel zu PowerShell Direct	263
11	PowerShell-Werkzeuge	267
11.1	PowerShell-Standardkonsole	267

11.2	PowerShell Integrated Scripting Environment (ISE)	276
11.3	PowerShell Script Analyzer	286
11.4	PowerShell Analyzer	292
11.5	PowerShell Tools for Visual Studio	293
11.6	PowerShell Pro Tools for Visual Studio	294
11.7	NuGet Package Manager	295
11.8	PowerShell-Erweiterung für Visual Studio Code	295
11.9	PowerShell Web Access (PSWA)	298
11.10	Azure Cloud Shell	304
11.11	ISE Steroids	304
11.12	PowerShellPlus	305
11.13	PoshConsole	308
11.14	PowerGUI	309
11.15	PrimalScript	310
11.16	PowerShell Help	312
11.17	CIM Explorer for PowerShell ISE	312
11.18	PowerShell Help Reader	313
11.19	PowerShell Remoting	314
12	Windows PowerShell Core 5.1 in Windows Nano Server	315
13	PowerShell Core 6.x für Windows, Linux und MacOS	317
13.1	Funktionsumfang der PowerShell Core	318
13.2	PowerShell Core-Konsole	324
13.3	VSCoDe-PowerShell	325
13.4	Verwendung auf Linux und MacOS	326
13.5	PowerShell-Remoting via SSH	330
Teil B: PowerShell-Aufbauwissen		333
14	Verwendung von .NET-Klassen	335
14.1	Microsoft Developer Network (MSDN)	335
14.2	Erzeugen von Instanzen	336
14.3	Parameterbehaftete Konstruktoren	338
14.4	Initialisierung von Objekten	339
14.5	Nutzung von Attributen und Methoden	340
14.6	Statische Mitglieder in .NET-Klassen und statische .NET-Klassen	342
14.7	Generische Klassen nutzen	346
14.8	Zugriff auf bestehende Objekte	347
14.9	Laden von Assemblies	347
14.10	Objektanalyse	350

14.11	Auflistungen (Enumerationen)	350
14.12	Verknüpfen von Aufzählungswerten	351
15	Verwendung von COM-Klassen	353
15.1	Erzeugen von COM-Instanzen	353
15.2	Nutzung von Attributen und Methoden	354
15.3	Liste aller COM-Klassen	355
15.4	Holen bestehender COM-Instanzen	356
15.5	Distributed COM (DCOM)	356
16	Zugriff auf die Windows Management Instrumentation (WMI) ...	357
16.1	Einführung in WMI	357
16.2	WMI in der PowerShell	384
16.3	Open Management Infrastructure (OMI)	386
16.4	Abruf von WMI-Objektmengen	386
16.5	Fernzugriffe	387
16.6	Filtern und Abfragen	387
16.7	Liste aller WMI-Klassen	391
16.8	Hintergrundwissen: WMI-Klassenprojektion mit dem PowerShell-WMI-Objektadapter	392
16.9	Beschränkung der Ausgabeliste bei WMI-Objekten	396
16.10	Zugriff auf einzelne Mitglieder von WMI-Klassen	398
16.11	Werte setzen in WMI-Objekten	398
16.12	Umgang mit WMI-Datumsangaben	400
16.13	Methodenaufrufe	401
16.14	Neue WMI-Instanzen erzeugen	402
16.15	Instanzen entfernen	403
16.16	Commandlet Definition XML-Datei (CDXML)	404
17	Dynamische Objekte	407
17.1	Erweitern bestehender Objekte	407
17.2	Komplett dynamische Objekte	409
18	Einbinden von C# und Visual Basic .NET	411
19	Win32-API-Aufrufe	413
20	Benutzereingaben	417
20.1	Read-Host	417
20.2	Benutzerauswahl	418
20.3	Grafischer Eingabedialog	419
20.4	Dialogfenster	420

20.5	Authentifizierungsdialog	420
20.6	Zwischenablage (Clipboard)	422
21	Fehlersuche	425
21.1	Detailinformationen	425
21.2	Einzelmodus	426
21.3	Zeitmessung	427
21.4	Ablaufverfolgung (Tracing)	428
21.5	Erweiterte Protokollierung aktivieren	429
21.6	Script-Debugging in der ISE	431
21.7	Kommandozeilenbasiertes Script-Debugging	431
22	Transaktionen	433
22.1	Commandlets für Transaktionen	433
22.2	Start und Ende einer Transaktion	434
22.3	Zurücksetzen der Transaktion	435
22.4	Mehrere Transaktionen	436
23	Standardeinstellungen ändern mit Profilskripten	437
23.1	Profilpfade	437
23.2	Ausführungsreihenfolge	439
23.3	Beispiel für eine Profildatei	439
23.4	Starten der PowerShell ohne Profilskripte	440
24	Digitale Signaturen für PowerShell-Skripte	441
24.1	Zertifikat erstellen	441
24.2	Skripte signieren	443
24.3	Verwenden signierter Skripte	444
24.4	Mögliche Fehlerquellen	445
25	Hintergrundaufträge („Jobs“)	447
25.1	Voraussetzungen	447
25.2	Architektur	448
25.3	Starten eines Hintergrundauftrags	448
25.4	Hintergrundaufträge abfragen	449
25.5	Warten auf einen Hintergrundauftrag	450
25.6	Abbrechen und Löschen von Aufträgen	450
25.7	Analyse von Fehlermeldungen	451
25.8	Fernausführung von Hintergrundaufträgen	451
25.9	Praxisbeispiel	452

26	Geplante Aufgaben und zeitgesteuerte Jobs	455
26.1	Geplante Aufgaben (Scheduled Tasks)	455
26.2	Zeitgesteuerte Jobs	459
27	PowerShell-Workflows	465
27.1	Ein erstes Beispiel	465
27.2	Unterschiede zu einer Function bzw. einem Skript	470
27.3	Einschränkungen bei Workflows	470
27.4	Workflows in der Praxis	472
27.5	Workflows in Visual Studio erstellen	479
28	Ereignissystem	497
28.1	WMI-Ereignisse	497
28.2	WMI-Ereignisabfragen	497
28.3	WMI-Ereignisse seit PowerShell 1.0	499
28.4	Registrieren von WMI-Ereignisquellen seit PowerShell 2.0	500
28.5	Auslesen der Ereignisliste	501
28.6	Reagieren auf Ereignisse	503
28.7	WMI-Ereignisse ab PowerShell-Version 3.0	505
28.8	Registrieren von .NET-Ereignissen	505
28.9	Erzeugen von Ereignissen	506
29	Datenbereiche und Datendateien	509
29.1	Datenbereiche	509
29.2	Datendateien	511
29.3	Mehrsprachigkeit/Lokalisierung	512
30	Desired State Configuration (DSC)	515
30.1	Grundprinzipien	516
30.2	DSC für Linux	516
30.3	Ressourcen	517
30.4	Verfügbare DSC-Ressourcen	517
30.5	Eigenschaften einer Ressource	520
30.6	Aufbau eines DSC-Dokuments	520
30.7	Commandlets für die Arbeit mit DSC	521
30.8	Ein erstes DSC-Beispiel	521
30.9	Kompilieren und Anwendung eines DSC-Dokuments	522
30.10	Variablen in DSC-Dateien	524
30.11	Parameter für DSC-Dateien	525
30.12	Konfigurationsdaten	526
30.13	Entfernen einer DSC-Konfiguration	529

30.14 DSC Pull Server	532
30.15 DSC-Praxisbeispiel 1: IIS installieren	540
30.16 DSC-Praxisbeispiel 2: Software installieren	541
30.17 DSC-Praxisbeispiel 3: Software deinstallieren	543
30.18 Realisierung einer DSC-Ressource	544
30.19 Weitere Möglichkeiten	545
31 PowerShell-Snap-Ins	547
31.1 Einbinden von Snap-Ins	547
31.2 Liste der Commandlets	551
32 PowerShell-Module	553
32.1 Überblick über die Commandlets	553
32.2 Modulararchitektur	554
32.3 Module aus dem Netz herunterladen und installieren mit PowerShellGet	555
32.4 Module manuell installieren	562
32.5 Doppeldeutige Namen	562
32.6 Auflisten der verfügbaren Module	563
32.7 Importieren von Modulen	565
32.8 Entfernen von Modulen	568
33 Ausgewählte PowerShell-Erweiterungen	569
33.1 PowerShell-Module in Windows 7 und Windows Server 2008 R2	570
33.2 PowerShell-Module in Windows 8.0 und Windows Server 2012	571
33.3 PowerShell-Module in Windows 8.1 und Windows Server 2012 R2	573
33.4 PowerShell-Module in Windows 10 und Windows Server 2016	576
33.5 PowerShell Community Extensions (PSCX)	580
33.6 PowerShellPack	584
33.7 www.IT-Visions.de: PowerShell Extensions	585
33.8 Quest Management Shell for Active Directory	586
33.9 Microsoft Exchange Server	587
33.10 System Center Virtual Machine Manager	588
33.11 PowerShell Management Library for Hyper-V (pshyperv)	589
33.12 Powershell Outlook Account Manager	590
33.13 PowerShell Configurator (PSConfig)	591
33.14 Weitere Erweiterungen	592
34 Delegierte Administration/Just Enough Administration (JEA) ...	593
34.1 JEA-Konzept	593
34.2 PowerShell-Sitzungskonfiguration erstellen	593
34.3 Sitzungskonfiguration nutzen	597
34.4 Delegierte Administration per Webseite	598

35	Tipps und Tricks zur PowerShell	599
35.1	Alle Anzeigen löschen	599
35.2	Befehlsgeschichte	599
35.3	System- und Hostinformationen	600
35.4	Anpassen der Eingabeaufforderung (Prompt)	601
35.5	PowerShell-Befehle aus anderen Anwendungen heraus starten	602
35.6	ISE erweitern	603
35.7	PowerShell für Gruppenrichtlinienskripte	604
35.8	Einblicke in die Interna der Pipeline-Verarbeitung	606
Teil C: PowerShell im Praxiseinsatz		609
36	Dateisystem	611
36.1	Laufwerke	612
36.2	Ordnerinhalte	617
36.3	Dateieigenschaften verändern	619
36.4	Eigenschaften ausführbarer Dateien	620
36.5	Kurznamen	622
36.6	Lange Pfade	622
36.7	Dateisystemoperationen	623
36.8	Praxisbeispiel: Zufällige Dateisystemstruktur erzeugen	624
36.9	Praxisbeispiel: Leere Ordner löschen	625
36.10	Einsatz von Robocopy	626
36.11	Dateisystemkataloge	629
36.12	Papierkorb leeren	630
36.13	Dateieigenschaften lesen	630
36.14	Praxisbeispiel: Fotos nach Aufnahmedatum sortieren	631
36.15	Datei-Hash	632
36.16	Finden von Duplikaten	633
36.17	Verknüpfungen im Dateisystem	635
36.18	Komprimierung	640
36.19	Dateisystemfreigaben	642
36.20	Überwachung des Dateisystems	653
36.21	Dateiversionsverlauf	654
36.22	Windows Explorer öffnen	655
36.23	Windows Server Backup	655
37	Festplattenverschlüsselung mit BitLocker	659
37.1	Übersicht über das BitLocker-Modul	660
37.2	Verschlüsseln eines Laufwerks	661

38	Dokumente	663
38.1	Textdateien	663
38.2	CSV-Dateien	664
38.3	Analysieren von Textdateien	667
38.4	INI-Dateien	670
38.5	XML-Dateien	671
38.6	HTML-Dateien	679
38.7	Binärdateien	679
39	Datenbanken	681
39.1	ADO.NET-Grundlagen	681
39.2	Beispieldatenbank	687
39.3	Datenzugriff mit den Bordmitteln der PowerShell	688
39.4	Datenzugriff mit den PowerShell-Erweiterungen	699
39.5	Datenbankzugriff mit SQLPS	702
39.6	Datenbankzugriff mit SQLPSX	702
40	Microsoft-SQL-Server-Administration	703
40.1	PowerShell-Integration im SQL Server Management Studio	704
40.2	SQL-Server-Laufwerk „SQLSERVER:“	705
40.3	Die SQLPS-Commandlets	708
40.4	Die SQL Server Management Objects (SMO)	710
40.5	SQLPSX	713
40.6	Microsoft-SQL-Server-Administration mit der PowerShell in der Praxis	721
41	ODBC-Datenquellen	727
41.1	ODBC-Treiber und -Datenquellen auflisten	728
41.2	Anlegen einer ODBC-Datenquelle	729
41.3	Zugriff auf eine ODBC-Datenquelle	730
42	Registrierungsdatenbank (Registry)	733
42.1	Schlüssel auslesen	733
42.2	Schlüssel anlegen und löschen	734
42.3	Laufwerke definieren	734
42.4	Werte anlegen und löschen	735
42.5	Werte auslesen	736
42.6	Praxisbeispiel: Windows-Explorer-Einstellungen	736
42.7	Praxisbeispiel: Massenanlegen von Registry-Schlüsseln	737
43	Computer- und Betriebssystemverwaltung	739
43.1	Computerinformationen	739

43.2	Versionsnummer des Betriebssystems	741
43.3	Zeitdauer seit dem letzten Start des Betriebssystems	741
43.4	BIOS- und Startinformationen	742
43.5	Windows-Produktaktivierung	742
43.6	Umgebungsvariablen	742
43.7	Schriftarten	745
43.8	Computername und Domäne	746
43.9	Herunterfahren und Neustarten	746
43.10	Windows Updates installieren	747
43.11	Wiederherstellungspunkte verwalten	751
44	Windows Defender	753
45	Hardwareverwaltung	755
45.1	Hardwarebausteine	755
45.2	Plug-and-Play-Geräte	757
45.3	Druckerverwaltung (ältere Betriebssysteme)	757
45.4	Druckerverwaltung (seit Windows 8 und Windows Server 2012)	758
46	Softwareverwaltung	761
46.1	Softwareinventarisierung	761
46.2	Installation von Anwendungen	764
46.3	Deinstallation von Anwendungen	765
46.4	Praxisbeispiel: Installationstest	765
46.5	Installationen mit PowerShell Package Management („OneGet“)	766
46.6	Versionsnummer ermitteln	769
46.7	Servermanager	770
46.8	Softwareeinschränkungen mit dem PowerShell-Modul „AppLocker“	781
47	Prozessverwaltung	787
47.1	Prozesse auflisten	787
47.2	Prozesse starten	788
47.3	Prozesse mit vollen Administratorrechten starten	789
47.4	Prozesse unter einem anderen Benutzerkonto starten	790
47.5	Prozesse beenden	791
47.6	Warten auf das Beenden einer Anwendung	792
48	Systemdienste	793
48.1	Dienste auflisten	793
48.2	Dienstzustand ändern	795
48.3	Diensteigenschaften ändern	796

49	Netzwerk	797
49.1	Netzwerkconfiguration (ältere Betriebssysteme)	797
49.2	Netzwerkconfiguration (ab Windows 8 und Windows Server 2012)	799
49.3	DNS-Client-Konfiguration	802
49.4	DNS-Namensauflösung	805
49.5	Erreichbarkeit prüfen (Ping)	807
49.6	Windows Firewall	808
49.7	Remote Desktop (RDP) einrichten	814
49.8	E-Mails senden (SMTP)	815
49.9	Auseinandernehmen von E-Mail-Adressen	817
49.10	Abruf von Daten von einem HTTP-Server	817
49.11	Praxisbeispiel: Linkprüfer für eine Website	819
49.12	Aufrufe von SOAP-Webdiensten	822
49.13	Aufruf von REST-Diensten	824
49.14	Aufrufe von OData-Diensten	826
49.15	Hintergrunddatentransfer mit BITS	827
50	Ereignisprotokolle (Event Log)	831
51	Leistungsdaten (Performance Counter)	835
51.1	Zugriff auf Leistungsindikatoren über WMI	835
51.2	Get-Counter	836
52	Sicherheitseinstellungen	839
52.1	Aktueller Benutzer	839
52.2	Grundlagen	840
52.3	Zugriffsrechtelisten auslesen	845
52.4	Einzelne Rechteinträge auslesen	846
52.5	Besitzer auslesen	848
52.6	Benutzer und SID	848
52.7	Hinzufügen eines Rechteintrags zu einer Zugriffsrechteliste	851
52.8	Entfernen eines Rechteintrags aus einer Zugriffsrechteliste	854
52.9	Zugriffsrechteliste übertragen	855
52.10	Zugriffsrechteliste über SDDL setzen	856
52.11	Zertifikate verwalten	857
53	Optimierungen und Problemlösungen	861
53.1	PowerShell-Modul „TroubleshootingPack“	861
53.2	PowerShell-Modul „Best Practices“	865

54	Active Directory	867
54.1	Benutzer- und Gruppenverwaltung mit WMI	868
54.2	Einführung in System.DirectoryServices	869
54.3	Basiseigenschaften	880
54.4	Benutzer- und Gruppenverwaltung im Active Directory	882
54.5	Verwaltung der Organisationseinheiten	890
54.6	Suche im Active Directory	891
54.7	Navigation im Active Directory mit den PowerShell Extensions	898
54.8	Verwendung der Active-Directory-Erweiterungen von www.IT-Visions.de	899
54.9	PowerShell-Modul „Active Directory“ (ADPowerShell)	901
54.10	PowerShell-Modul „ADDSDeployment“	924
54.11	Informationen über die Active-Directory-Struktur	926
55	Gruppenrichtlinien	929
55.1	Verwaltung der Gruppenrichtlinien	930
55.2	Verknüpfung der Gruppenrichtlinien	931
55.3	Gruppenrichtlinienberichte	933
55.4	Gruppenrichtlinienvererbung	934
55.5	Weitere Möglichkeiten	935
56	Lokale Benutzer und Gruppen	937
56.1	Modul „Microsoft.PowerShell.LocalAccounts“	937
56.2	Lokale Benutzerverwaltung in älteren PowerShell-Versionen	939
57	Microsoft Exchange Server	941
57.1	Daten abrufen	941
57.2	Postfächer verwalten	942
57.3	Öffentliche Ordner verwalten	943
58	Internet Information Server (IIS)	945
58.1	Überblick	945
58.2	Navigationsprovider	947
58.3	Anlegen von Websites	949
58.4	Praxisbeispiel: Massenanlegen von Websites	950
58.5	Ändern von Website-Eigenschaften	953
58.6	Anwendungspool anlegen	953
58.7	Virtuelle Verzeichnisse und IIS-Anwendungen	954
58.8	Website-Zustand ändern	955
58.9	Anwendungspools starten und stoppen	955
58.10	Löschen von Websites	956

59	Virtuelle Systeme mit Hyper-V	957
59.1	Das Hyper-V-Modul von Microsoft	958
59.2	Die ersten Schritte mit dem Hyper-V-Modul	960
59.3	Virtuelle Maschinen anlegen	964
59.4	Umgang mit virtuellen Festplatten	970
59.5	Konfiguration virtueller Maschinen	973
59.6	Dateien kopieren in virtuelle Systeme	977
59.7	PowerShell Management Library for Hyper-V (für ältere Betriebssysteme)	979
60	Windows Nano Server	983
60.1	Das Konzept von Nano Server	983
60.2	Einschränkungen von Nano Server	985
60.3	Varianten des Nano Servers	987
60.4	Installation eines Nano Servers	987
60.5	Docker-Image	989
60.6	Fernverwaltung mit PowerShell	989
60.7	Windows Update auf einem Nano Server	992
60.8	Nachträgliche Paketinstallation	992
60.9	Abgespeckter IIS unter Nano Server	994
60.10	Nano-Serververwaltung aus der Cloud heraus	995
61	Docker-Container	997
61.1	Docker-Varianten für Windows	998
61.2	Docker-Installation auf Windows 10	999
61.3	Docker-Installation auf Windows Server 2016	1001
61.4	Installation von „Docker for Windows“	1002
61.5	Docker-Registries	1004
61.6	Docker-Images laden	1004
61.7	Container starten	1005
61.8	Container-Identifikation	1006
61.9	Container mit Visual Studio	1007
61.10	Befehle in einem Container ausführen	1009
61.11	Ressourcenbeschränkungen für Container	1011
61.12	Dateien zwischen Container und Host kopieren	1011
61.13	Dockerfile	1011
61.14	Docker-Netzwerke	1012
61.15	Container anlegen, ohne sie zu starten	1013
61.16	Container starten und stoppen	1013
61.17	Container beenden und löschen	1013
61.18	Images löschen	1014

61.19	Images aus Containern erstellen	1014
61.20	.NET Core-Container	1014
61.21	Images verbreiten	1017
61.22	Azure Container Service (ACS)	1019
62	Grafische Benutzeroberflächen (GUI)	1021
62.1	Einfache Nachfragedialoge	1021
62.2	Einfache Eingabe mit Inputbox	1023
62.3	Komplexere Eingabemasken	1024
62.4	Universelle Objektdarstellung	1026
62.5	WPF PowerShell Kit (WPK)	1027
62.6	Direkte Verwendung von WPF	1035
Teil D:	Profiwissen – Erweitern der PowerShell	1037
63	Entwicklung von Commandlets in der PowerShell- Skriptsprache	1039
63.1	Aufbau eines skriptbasierten Commandlets	1039
63.2	Verwendung per Dot Sourcing	1041
63.3	Parameterfestlegung	1042
63.4	Fortgeschrittene Funktion (Advanced Function)	1048
63.5	Mehrere Parameter und Parametersätze	1050
63.6	Unterstützung für Sicherheitsabfragen (-whatif und -confirm)	1052
63.7	Kaufmännisches Beispiel: Test-CustomerID	1054
63.8	Erweitern bestehender Commandlets durch Proxy-Commandlets	1057
63.9	Dokumentation	1063
64	Entwicklung eigener Commandlets mit C#	1067
64.1	Technische Voraussetzungen	1068
64.2	Grundkonzept der .NET-basierten Commandlets	1069
64.3	Schrittweise Erstellung eines minimalen Commandlets	1071
64.4	Erstellung eines Commandlets mit einem Rückgabeobjekt	1079
64.5	Erstellung eines Commandlets mit mehreren Rückgabeobjekten	1081
64.6	Erstellen eines Commandlets mit Parametern	1085
64.7	Verarbeiten von Pipeline-Eingaben	1087
64.8	Verkettung von Commandlets	1090
64.9	Fehlersuche in Commandlets	1094
64.10	Statusinformationen	1097
64.11	Unterstützung für Sicherheitsabfragen (-whatif und -confirm)	1102
64.12	Festlegung der Hilfeinformationen	1104
64.13	Erstellung von Commandlets für den Zugriff auf eine Geschäftsanwendung ...	1108

64.14	Konventionen für Commandlets	1109
64.15	Weitere Möglichkeiten	1111
65	PowerShell-Module erstellen	1113
65.1	Erstellen eines Skriptmoduls	1113
65.2	Praxisbeispiel: Umwandlung einer Skriptdatei in ein Modul	1115
65.3	Erstellen eines Moduls mit Binärdateien	1115
65.4	Erstellen eines Moduls mit Manifest	1116
65.5	Erstellung eines Manifest-Moduls mit Visual Studio	1123
66	Hosting der PowerShell	1125
66.1	Voraussetzungen für das Hosting	1126
66.2	Hosting mit PSHost	1127
66.3	Vereinfachtes Hosting seit PowerShell 2.0	1130
	Anhang A: Crashkurs „Objektorientierung“	1133
	Anhang B: Crashkurs .NET	1141
B.1	Was ist das .NET Framework?	1143
B.2	Was ist .NET Core?	1144
B.3	Eigenschaften von .NET	1145
B.4	.NET-Klassen	1146
B.5	Namensgebung von .NET-Klassen (Namensräume)	1146
B.6	Namensräume und Softwarekomponenten	1148
B.7	Bestandteile einer .NET-Klasse	1149
B.8	Vererbung	1150
B.9	Schnittstellen	1151
	Anhang C: Literatur	1152
	Anhang D: Weitere Informationen im Internet	1155
	Anhang E: Abkürzungsverzeichnis	1157
	Stichwortverzeichnis	1183

Vorwort zur sechsten Auflage

Liebe Leserin, lieber Leser,

willkommen zur fünften Auflage dieses PowerShell-Buchs! Das vor Ihnen liegende Buch behandelt die Windows PowerShell in der Version 5.1 sowie die PowerShell Core in der Version 6.0 (zum Redaktionsschluss dieses Buchs noch Beta-Version) von Microsoft sowie ergänzende Werkzeuge von Microsoft und Drittanbietern (z.B. PowerShell Community Extensions). Das Buch ist aber auch geeignet, wenn Sie noch PowerShell 2.0, 3.0, 4.0 oder 5.0 einsetzen. Welche Funktionen neu hinzugekommen sind, wird jeweils erwähnt.

■ Wer bin ich?

Mein Name ist Holger Schwichtenberg, ich bin derzeit 44 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 5Minds IT-Solutions GmbH & Co. KG Software (www.5Minds.de) im Auftrag von Kunden in zahlreichen Branchen.

Es ist mein Hobby und Nebenberuf, IT-Fachbücher zu schreiben. Dieses Buch ist, unter Mitzählung aller nennenswerten Neuauflagen, das 67. Buch, das ich allein oder mit Co-Autoren geschrieben habe. Meine weiteren Hobbys sind Mountain Biking, Lauf-Sport, 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 3.0, 4.0, 5.x oder 6.x unterstützten Betriebssystemen? (Windows 7/8/8.1/10, Windows Server 2008/2008 R2/2012/2012 R2/2016) Hinweis: Die PowerShell 6.0 für Linux und MacOS wird nur als Randthema kurz in diesem Buch behandelt, da es hier bislang kaum Befehle für die PowerShell gibt!	<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 1200 Seiten nicht alle Details erläutern kann?	<input type="radio"/> Ja	<input type="radio"/> Nein

Wenn Sie alle obigen Fragen mit „Ja“ beantwortet haben, ist das Buch 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 5.0 wurde das Buch um die neuen Funktionen in Windows PowerShell 5.1 sowie PowerShell Core 6.0 erweitert und inhaltlich optimiert. Praxiseinsatzkapitel wurden ergänzt zu Windows Update, Windows Nano Server und Docker-Containern. Zudem wurden die bestehenden Inhalte des Buchs an vielen Stellen erweitert und didaktisch optimiert.

■ Sind in diesem Buch alle Features der PowerShell beschrieben?

Die PowerShell umfasst mittlerweile über 1500 Commandlets mit jeweils zahlreichen Optionen. Zudem gibt es unzählige Erweiterungen mit vielen hundert weiteren Commandlets. Zudem 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 eingesetzt. Ich beschreibe in diesem Buch, was ich selbst in der Praxis, in meinen Schulungen und bei Kundeneinsätzen verwende. Es macht auch keinen Sinn, jedes Detail der PowerShell hier zu dokumentieren. Stattdessen gebe ich Ihnen **Hilfe zur Selbsthilfe**, damit Sie die Konzepte gut verstehen und sich dann Sonderfälle selbst erarbeiten können.

■ Wie aktuell ist dieses Buch?

Die Informationstechnik hat sich immer schon schnell verändert. Seit aber auch Microsoft das Themen „Agilität“ und „Open Source“ für sich entdeckt hat, ist die Entwicklung nicht mehr 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 häufig. Aufwärts- und Abwärtskompatibilität ist kein Ziel mehr. Es wird erwartet, dass Sie Ihre Lösungen ständig den neuen Gegebenheiten anpassen.
- Produkte werden nicht mehr so ausführlich dokumentiert wie früher. Teilweise erscheint Dokumentation erst deutlich nach dem Erscheinen der Software.
- Produkte werden schnell auch wieder abgekündigt, wenn sie sich aus der Sicht der Hersteller bzw. aufgrund des Nutzerfeedbacks nicht bewährt haben.

Unter diesen neuen Einflusstrends steht natürlich auch dieses etablierte Buch. Leider kann man ein gedrucktes Buch nicht so schnell ändern wie Software. Verlage definieren erhebliche Mindestauflagen, die abverkauft werden müssen, bevor neu gedruckt werden darf. Das E-Book ist keine Alternative. Die Verkaufszahlen zeigen, dass nur eine verschwindend 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 im Leser-Portal (siehe unten). Ich werde dies dann in Neuauflagen des Buchs berücksichtigen.

■ Wem ist zu danken?

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

- meinem Kollegen und Freund Peter Monadjemi, der rund 100 Seiten mit Beispielen zu der Vor-Vor-Vor-Auflage dieses Buchs beigetragen hat (Themen: Workflows, Bitlocker, ODBC, Hyper-V, DNS-Client, Firewall und SQL-Server-Administration),
- 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,
- meiner Frau und meinen Kindern dafür, dass sie mir das Umfeld geben, um neben meinem Hauptberuf an Büchern wie diesem zu arbeiten.

■ Woher bekommen Sie die Beispiele aus diesem Buch?

Unter <http://www.powershell-doktor.de/leser> biete ich ein **ehrenamtlich betriebenes** Webportal für Leser meiner Bücher an. 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,
- Feedback zu diesem Buch geben (Bewertung abgeben und Fehler melden) und
- technische Fragen in einem Webforum stellen.

Alle registrierten Leser erhalten auch Einladungen zu kostenlosen Community-Veranstaltungen sowie Vergünstigungen bei unseren öffentlichen Seminaren zu .NET und zur PowerShell. Bei der Registrierung müssen Sie das Kennwort **Rogue One** angeben.

■ Wie sind die Programmcodebeispiele organisiert?

Die Beispiele sind im Archiv organisiert nach den Buchteilen und innerhalb der Buchteile nach Kapitelnamen (verkürzt). In diesem Buch wird für den Zugriff auf die Beispieldateien das X:-Laufwerk verwendet. Dies müssen Sie auf Ihre Situation anpassen!

```
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

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     PowerShellOOP
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
```

■ Wo können Sie sich schulen oder beraten lassen?

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

■ Zum Schluss des Vorworts ...

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

Dr. Holger Schwichtenberg

Essen, im Juni 2017

Über den Autor

Dr. Holger Schwichtenberg



- Studienabschluss Diplom-Wirtschaftsinformatik an der Universität Essen
- Promotion an der Universität Essen im Gebiet komponentenbasierter Softwareentwicklung
- Seit 1996 selbstständig als unabhängiger Berater, Dozent, Softwarearchitekt und Fachjournalist
- Leiter des Berater- und Dozententeams bei *www.IT-Visions.de*

www.IT-Visions.de
Dr. Holger Schwichtenberg

- Leitung der Softwareentwicklung im Bereich Microsoft/.NET bei der 5Minds IT-Solutions GmbH & Co. KG (*www.5minds.de*)

5Minds
IT - SOLUTIONS

- Über 65 Fachbücher beim Carl Hanser Verlag, bei O'Reilly, Microsoft Press und Addison-Wesley sowie mehr als 950 Beiträge in Fachzeitschriften
- 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. Microsoft TechEd, Microsoft Summit, Microsoft IT Forum, BASTA, BASTA-on-Tour, .NET Architecture Camp, Advanced Developers Conference, Developer Week, OOP, DOTNET Cologne, MD DevDays, Community in Motion, DOTNET-Konferenz, VS One, NRW.Conf, Net.Object Days, Windows Forum)Zertifikate und Auszeichnungen von Microsoft:
 - Bereits 14 mal ausgezeichnet als Microsoft Most Valuable Professional (MVP)
 - Zertifiziert als Microsoft Certified Solution Developer (MCSD)
- Thematische Schwerpunkte:
 - Softwarearchitektur, mehrschichtige Softwareentwicklung, Softwarekomponenten, SOA

- Microsoft .NET Framework, Visual Studio, C#, Visual Basic
- .NET-Architektur/Auswahl von .NET-Technologien
- Einführung von .NET Framework und Visual Studio/Migration auf .NET
- Webanwendungsentwicklung und Cross-Plattform-Anwendungen mit HTML, ASP.NET, JavaScript/TypeScript und Webframeworks wie Angular
- Enterprise .NET, verteilte Systeme/Webservices mit .NET insbes. Windows Communication Foundation und WebAPI
- Relationale Datenbanken, XML, Datenzugriffsstrategien
- Objektrelationales Mapping (ORM), insbesondere ADO.NET Entity Framework und EF Core
- Windows PowerShell, PowerShell Core und Windows Management Instrumentation (WMI)
- Ehrenamtliche Community-Tätigkeiten:
 - Vortragender für die International .NET Association (INETA)
 - Betrieb diverser Community-Websites: www.dotnetframework.de, www.entwicklerlexikon.de, www.windows-scripting.de, www.aspnetdev.de u. a.
- Firmenwebsites: <http://www.IT-Visions.de> und <http://www.5minds.de>
- Weblog: <http://www.dotnet-doktor.de>
- Kontakt: kundenteam@IT-Visions.de sowie *Telefon 02 01-64 95 90-0*

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 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 h:\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 *h:\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 40 000. *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 nächstes Kapitel).

■ 5.2 .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.

■ 5.3 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.

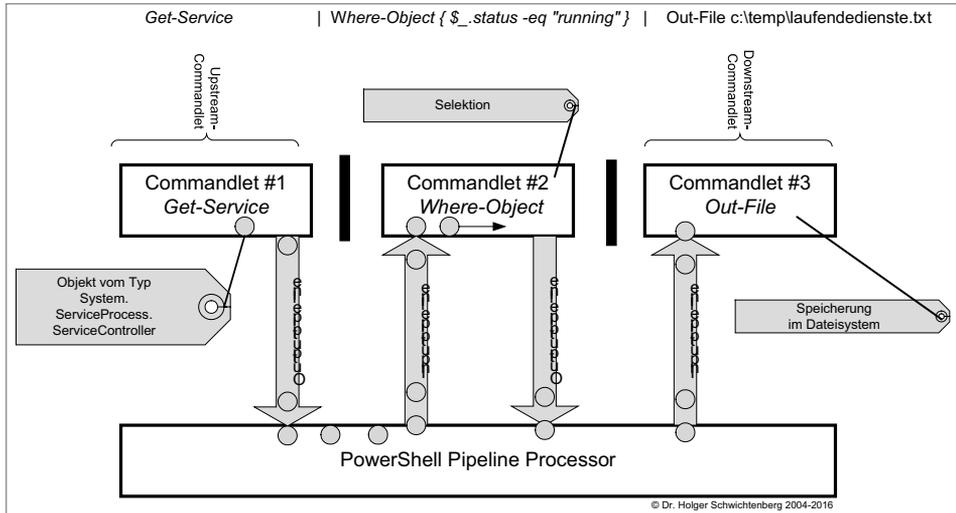


Bild 5.1 Der Pipeline Processor befördert die Objekte vom Downstream-Commandlet zum Upstream-Commandlet. Die Verarbeitung ist in der Regel asynchron.

Wie das obige Bild schon zeigt, beginnt ein nachfolgendes Commandlet mit seiner Arbeit, sobald es ein erstes Objekt aus der Pipeline erhält. Es kann also sein, dass das erste Commandlet noch gar nicht alle Objekte erzeugt hat, bevor die folgenden Commandlets schon die ersten Objekte asynchron weiterverarbeiten. Ein Commandlet wird sofort aufgerufen, sobald das erste Objekt bereitsteht. Man nennt dies „Streaming-Verarbeitung“. Streaming-Verarbeitung ist viel schneller als die klassische sequentielle Verarbeitung, weil die folgenden Commandlets in der Pipeline nicht auf vorhergehende warten müssen.

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.

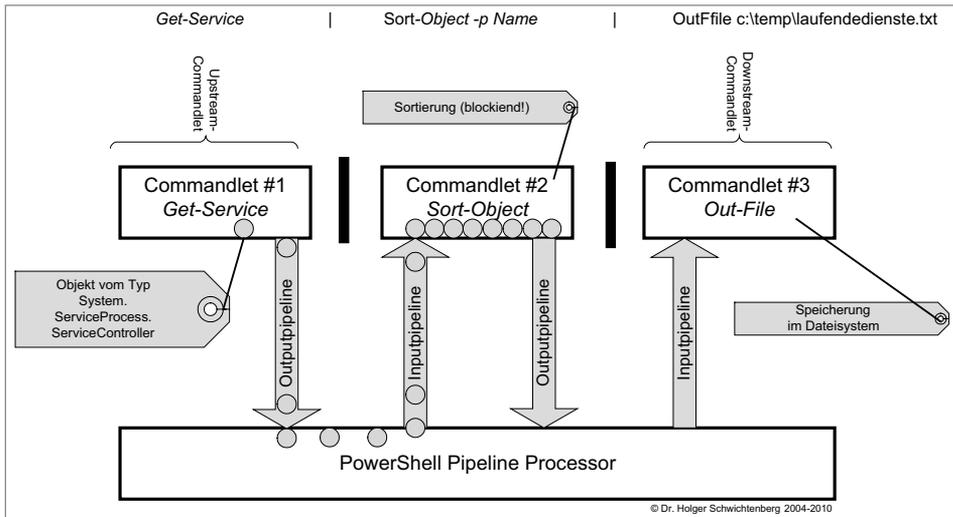


Bild 5.2 Sort-Object blockiert die direkte Weitergabe. Erst wenn alle Objekte angekommen sind, kann das Commandlet sortieren.

■ 5.4 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 Bildschirmabbildung zeigt einige Parameter des Commandlets `Get-Service`. Diese Liste erhält man durch den Befehl `Get-Help Get-Service -Parameter *`.

Interessant sind die mit gelbem 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.

```

-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
  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
  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
  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

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 <scripthlock>
  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 -computersname 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 @{Computersname="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

■ 5.5 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.

In dem folgenden Beispiel werden nur diejenigen Zeilen der Ausgabe von *netstat.exe* gefiltert, die ein großes „E“ gefolgt von zwei Ziffern enthalten.



TIPP: Die Syntax der regulären Ausdrücke in .NET wird in Kapitel 7 „PowerShell-Skriptsprache“ noch etwas näher beschrieben werden.

```

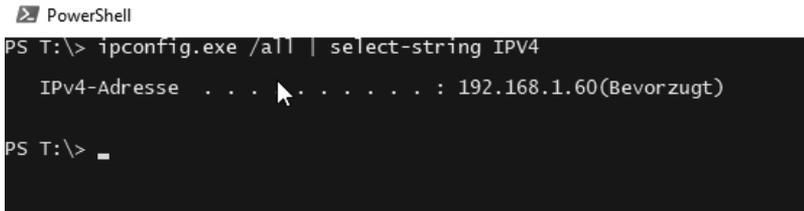
PowerShell - hs [elevated user] - C:\WINDOWS
17# netstat
Active Connections
  Proto Local Address           Foreign Address         State
  TCP   e01:1078                192.168.1.25:1025      ESTABLISHED
  TCP   e01:1142                65.55.5.84:https       ESTABLISHED
  TCP   e01:5590                E02:ldap               CLOSE_WAIT
  TCP   e01:5600                E02:ldap               CLOSE_WAIT
  TCP   e01:5858                nf-in-f99.google.com:https CLOSE_WAIT
  TCP   e01:6233                E02:ldap               ESTABLISHED
  TCP   e01:6266                E04:1789               TIME_WAIT
18# netstat | select-string "E\d\d" -case
  TCP   e01:5590                E02:ldap               CLOSE_WAIT
  TCP   e01:5600                E02:ldap               CLOSE_WAIT
  TCP   e01:6233                E02:ldap               ESTABLISHED
  TCP   e01:6295                E04:opsmgr            TIME_WAIT
19# _

```

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
```



```

PowerShell
PS T:\> ipconfig.exe /all | select-string IPV4
IPv4-Adresse . . . : 192.168.1.60 (Bevorzugt)
PS T:\>

```

Bild 5.7 Ausführung des obigen Befehls

■ 5.6 Anzahl der Objekte in der Pipeline

Die meisten Commandlets legen ganze Mengen von Objekten in die Pipeline (z.B. `Get-Process` eine Liste der Prozesse und `Get-Service` eine Liste der Dienste). Einige Commandlets legen aber nur einzelne Objekte in die Pipeline. Ein Beispiel dafür ist `Get-Date`, das ein einziges Objekt des Typs `System.DateTime` in die Pipeline legt. Es kann aber auch sein, dass ein Commandlet, das normalerweise eine Liste von Objekten liefert, im konkreten Fall nur ein einzelnes Objekt liefert (z.B. `Get-Process idle`). In diesem Fall liefert die PowerShell dem Benutzer nicht eine Liste mit einem Objekt, sondern direkt das ausgepackte Objekt.

Bis Version 2.0 war es so, dass man eine Liste durch Zugriff auf `Count` oder `Length` nach der Anzahl der Elemente fragen konnte, nicht aber ein einzelnes Objekt.

Das war also erlaubt:

```
(Get-Process).count
```

Das führte aber zu keinem Ergebnis:

```
(Get-Process idle).count
(Get-Date).count
```

Seit PowerShell-Version 3.0 ist dieser Unterschied aufgehoben, man kann immer `Count` und `Length` abfragen und die PowerShell liefert dann eben bei Einzelobjekten eine „1“ zurück. Allerdings schlägt die Eingabehilfe der PowerShell-Konsole und der PowerShell ISE weiterhin weder `Count` noch `Length` als Möglichkeit vor!

Praxisbeispiel: Wie viele Prozesse gibt es, die mehr als 20 MB Speicher verbrauchen?

```
(Get-Process | where-object { $_.WorkingSet64 -gt 20mb }).Count
```

```

PS C:\Windows\System32> (get-process | where-object { $_.WorkingSet64 -gt 20mb }).Count
21
PS C:\Windows\System32>

```

Bild 5.8 Aufruf von `Count` für eine Pipeline

■ 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 5.9 Zeilenumbruch nach Pipeline-Symbol

■ 5.8 Zugriff auf einzelne Objekte aus einer Menge

Ruft man ein Commandlet auf, das ein einzelnes Objekt liefert, hat man direkt dieses Objekt in Händen. Ruft man z. B. `Get-Date` ohne Weiteres auf, werden das aktuelle Datum und die aktuelle Zeit ausgegeben.

Bei einer Objektmenge kann man, wie oben bereits gezeigt, mit `Where-Object` filtern. Es ist aber auch 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.

```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) 2013 Microsoft Corporation. All rights reserved.

PS C:\WINDOWS\system32> Get-Date
Montag, 13. Januar 2014 16:11:57

PS C:\WINDOWS\system32> ^
```

Bild 5.10
Das aktuelle Datum
mit Zeit

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 kombinieren mit Bedingungen. 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 5.11 Zugriff auf einzelne Prozessobjekte

■ 5.9 Zugriff auf einzelne Werte in einem Objekt

Manchmal möchte man nicht ein komplettes Objekt bzw. eine komplette Objektmenge verarbeiten, sondern nur eine einzelne Eigenschaft.

Oben wurde bereits gezeigt, wie man mit `Format-Table` auf einzelne Eigenschaften zugreifen kann:

```
Get-Process | Format-Table ProcessName, WorkingSet64
```

Hat man nur ein einzelnes Objekt in Händen, geht das ebenfalls:

```
(Get-Process)[0] | Format-Table ProcessName, WorkingSet64
```

`Format-Table` liefert aber immer eine bestimmte Ausgabe, eben in Tabellenform mit Kopfzeile. Wenn man wirklich nur einen bestimmten Inhalt einer Eigenschaft eines Objekts haben möchte, so verwendet man die in objektorientierten Sprachen übliche Punktnotation, d. h., man trennt das Objekt und die abzurufende Eigenschaft durch einen Punkt (Punktnotation).

Beispiele:

```
(Get-Process)[0].ProcessName
```

Die Ausgabe ist eine einzelne Zeichenkette mit dem Namen des Prozesses.

```
(Get-Process)[0].WorkingSet64
```

Die Ausgabe ist eine einzelne Zahl mit der Speichernutzung des Prozesses.

Mit den Einzelwerten kann man weiterrechnen, z. B. errechnet man so die Speichernutzung in Megabyte:

```
(Get-Process)[0].WorkingSet64 / 1MB
```

```
PS C:\Windows\System32> <get-process>[0] | Format-Table ProcessName, WorkingSet64
ProcessName                                     WorkingSet64
-----
cmd                                             2727936
PS C:\Windows\System32> <get-process>[0].ProcessName
cmd
PS C:\Windows\System32> <get-process>[0].WorkingSet64
2727936
PS C:\Windows\System32> <get-process>[0].WorkingSet64 / 1MB
2.6015625
PS C:\Windows\System32>
```

Bild 5.12 Ausgabe zu den obigen Beispielen

Weitere Anwendungsfälle seien am Beispiel `Get-Date` gezeigt. `Year`, `Day`, `Month`, `Hour` und `Minute` sind einige der zahlreichen Eigenschaften der Klasse `DateTime`, die `Get-Date` liefert.

```

PS C:\Windows\System32> <Get-Date>.Year
2009
PS C:\Windows\System32> <Get-Date>.Day
9
PS C:\Windows\System32> <Get-Date>.Month
9
PS C:\Windows\System32> <Get-Date>.Hour
14
PS C:\Windows\System32> <Get-Date>.Minute
4

```

Bild 5.13

Zugriff auf einzelne Werte aus dem aktuellen Datum/der aktuellen Zeit

Einzelne Werte aus allen Objekten einer Objektmenge

Wenn man einen einzelnen Wert aus allen Objekten aus einer Objektmenge ausgeben wollte, so konnte man das bis PowerShell 2.0 nur über ein nachgeschaltetes Foreach-Object lösen, wobei innerhalb von Foreach-Object mit `$_` auf das aktuelle Objekt der Pipeline zu verweisen war:

```
Get-Process | foreach-object {$_ .Name }
```

Das geht seit PowerShell-Version 3.0 wesentlich prägnanter und eleganter:

```
(Get-Process).Name
```

Oder

```
(Get-Process).WorkingSet
```

Weiterhin Foreach-Object anwenden muss man für eine kombinierte Ausgabe:

```
Get-Process | foreach-object {$_ .Name + ": " + $_ .WorkingSet }
```

Mancher könnte denken, dass

```
(Get-Process).Name + ":" + (Get-Process).WorkingSet
```

auch als Schreibweise möglich wäre. Das liefert aber weder optisch noch inhaltlich ein korrektes Ergebnis, denn die Prozessliste wird zweimal abgerufen und könnte sich in der Zwischenzeit geändert haben!

■ 5.10 Methoden ausführen

Der folgende PowerShell-Pipeline-Befehl beendet alle Instanzen des Internet Explorers auf dem lokalen System, indem das Commandlet `Stop-Process` die Instanzen des betreffenden Prozesses von `Get-Process` empfängt.

```
Get-Process iexplore | Stop-Process
```

Die Objekt-Pipeline der PowerShell hat noch weitere Möglichkeiten: Gemäß dem objektorientierten Paradigma haben .NET-Objekte nicht nur Attribute, sondern auch Methoden. In einer Pipeline kann der Administrator daher auch die Methoden der Objekte aufrufen.

Objekte des Typs `System.Diagnostics.Process` besitzen zum Beispiel eine Methode `Kill()`. Der Aufruf dieser Methode ist in der PowerShell gekapselt in der Methode `Stop-Process`.

Wer sich mit dem .NET Framework gut auskennt, könnte die `Kill()`-Methode auch direkt aufrufen. Dann ist aber eine explizite `ForEach`-Schleife notwendig. Die Commandlets iterieren automatisch über alle Objekte der Pipeline, die Methodenaufrufe aber nicht.

```
Get-Process iexplore | ForEach-Object { $_.Kill() }
```

Durch den Einsatz von Aliasen geht das auch kürzer:

```
ps | ? { $_.name -eq "iexplore" } | % { $_.Kill() }
```

Und seit PowerShell-Version 3.0 kann man auf das `ForEach-Object` bzw. `%` verzichten, also

```
(Get-Process iexplore).Kill()
```

oder

```
(ps iexplore).Kill()
```

schreiben.

Der Einsatz der Methode `Kill()` diene hier nur zur Demonstration, dass die Pipeline tatsächlich Objekte befördert. Eigentlich ist die gleiche Aufgabe besser mit dem eingebauten Commandlet `Stop-Process` zu lösen.



ACHTUNG: Vergessen Sie beim Aufruf von Methoden nicht die runden Klammern, auch wenn die Methoden keine Parameter besitzen. Ohne die Klammern erhalten Sie Informationen über die Methode, es erfolgt aber kein Aufruf.

```
PS C:\Users\hs.ITU> Get-Process notepad | foreach < $_.kill >
MemberType           : Method
OverloadDefinitions  : <System.Void Kill()>
TypeNameOfValue      : System.Management.Automation.PSMethod
Value                : System.Void Kill()
Name                 : Kill
IsInstance           : True
MemberType           : Method
OverloadDefinitions  : <System.Void Kill()>
TypeNameOfValue      : System.Management.Automation.PSMethod
Value                : System.Void Kill()
Name                 : Kill
IsInstance           : True
```

Runde
Kammern ()
fehlen

Bild 5.14 Folgen des vergessenen Klammerspaars

Dies funktioniert aber nur dann gut, wenn es auch Instanzen des Internet Explorers gibt. Wenn alle beendet sind, meldet `Get-Process` einen Fehler. Dies kann das gewünschte Verhalten sein. Mit einer etwas anderen Pipeline wird dieser Fehler jedoch unterbunden:

```
Get-Process | Where-Object { $_.Name -eq "iexplore" } |
Stop-Process
```

Die zweite Pipeline unterscheidet sich von der ersten dadurch, dass das Filtern der Prozesse aus der Prozessliste nun nicht mehr von `Get-Process` erledigt wird, sondern durch ein eigenes Commandlet mit Namen `Where-Object` in der Pipeline selbst durchgeführt wird. `Where-Object` ist toleranter als `Get-Process` in Hinblick auf die Möglichkeit, dass es kein passendes Objekt gibt.

`ps` ist ein Alias für `Get-Process`, `kill` für `Stop-Process`. Außerdem hat `Get-Process` eine eingebaute Filterfunktion. Um alle Instanzen des Internet Explorers zu beenden, kann man also statt

```
Get-Process | Where-Object { $_.Name -eq "iexplore" } | Stop-Process
```

auch schreiben:

```
ps -name "iexplore" | kill
```

Weitere Beispiele für die Aufrufe von Methoden seien am Beispiel von `Get-Date` gezeigt, das ja nur ein Objekt der Klasse `DateTime` liefert. Die Klasse `DateTime` bietet zahlreiche Methoden an, um Datum und Zeit auf bestimmte Weise darzustellen, z.B. `GetShortDateString()`, `GetLongDateString()`, `GetShortTimeString()` und `GetLongTimeString()`. Die Ausgaben zeigt die Bildschirmabbildung.

```
PS C:\Windows\System32> Get-Date
Mittwoch, 9. September 2009 15:00:16

PS C:\Windows\System32> <Get-Date>.ToShortDateString()
09.09.2009
PS C:\Windows\System32> <Get-Date>.ToLongDateString()
Mittwoch, 9. September 2009
PS C:\Windows\System32> <Get-Date>.ToShortTimeString()
15:00
PS C:\Windows\System32> <Get-Date>.ToLongTimeString()
15:00:38
PS C:\Windows\System32> _
```

Bild 5.15

Ausgaben der Methoden der Klasse `DateTime`

■ 5.11 Analyse des Pipeline-Inhalts

Zwei der größten Fragestellungen bei der praktischen Arbeit mit der PowerShell sind:

- Welchen Typ haben die Objekte, die ein Commandlet in die Pipeline legt?
- Welche Attribute und Methoden haben diese Objekte?

Die Hilfe der Commandlets ist hier nicht immer hilfreich. Bei `Get-Service` kann man lesen:

```
OUTPUTS
System.ServiceProcess.ServiceController
```

Bei anderen Commandlets aber heißt es nur wenig hilfreich:

```
OUTPUTS
Object
```

In keinem Fall sind in der PowerShell-Benutzerdokumentation ([MS01] und [MS02]) die Attribute und die Methoden der resultierenden Objekte genannt. Diese findet man nur in der MSDN-Dokumentation des .NET Frameworks.

Im Folgenden werden zwei hilfreiche Commandlets sowie zwei Methoden aus dem .NET Framework vorgestellt, die im Alltag helfen, zu erforschen, was man in der Pipeline hat:

- ToString()
- GetType()
- Get-PipelineInfo
- Get-Member

Methode ToString()

Jedes .NET-Objekt bietet die Methode ToString(), weil diese Methode von der Basisklasse aller .NET-Klassen System.Object an alle Klassen vererbt wird. Das Standardverhalten von ToString() ist, dass der Name der Klasse geliefert wird, zu der das Objekt gehört. Das heißt, dass die Ausgabe für alle Instanzen der Klasse gleich ist. Nur wenige Klassen überschreiben die Implementierung und liefern eine Zeichenkette, die tatsächlich den Inhalt des Objekts wiedergibt.

Listing 5.1 Basiswissen\Pipelining\ToString.psl

```
(Get-Service).ToString() # System.Object[]
(Get-Service w*)[0].ToString() # W32Time
(Get-Process w*)[0].ToString() # System.Diagnostics.Process (winit)
(Get-Host)[0].ToString() # System.Management.Automation.Internal.Host.InternalHost
(Get-Date).ToString() # liefert aktuelles Datum
```

Methode GetType()

Da jede PowerShell-Variable eine Instanz einer .NET-Klasse ist, besitzt jedes Objekt in der Pipeline die Methode GetType(), die es von der Mutter aller .NET-Klassen (System.Object) erbt. GetType() liefert ein System.Type-Objekt mit zahlreichen Informationen. Meistens interessiert man sich nur für den Klassennamen, den man aus FullName (mit Namensraum) oder Name (ohne Namensraum) auslesen kann. GetType() ist eine Methode und daher muss der Pipeline-Inhalt in runden Klammern stehen.

Beispiele zeigt die folgende Bildschirmabbildung.

```

PS C:\Users\HS> <Get-Date>.GetType()
-----
IsPublic IsSerial Name          BaseType
-----
True     True     DateTime        System.ValueType

PS C:\Users\HS> <Get-Process>.GetType()
-----
IsPublic IsSerial Name          BaseType
-----
True     True     Object[]        System.Array

PS C:\Users\HS> <Get-Process>[0].GetType()
-----
IsPublic IsSerial Name          BaseType
-----
True     False    Process         System.ComponentModel.Component

PS C:\Users\HS> <Get-Process>[0].GetType().FullName
System.Diagnostics.Process
PS C:\Users\HS> _

```

Bild 5.16 Einsatz von GetType()

Erläuterung: „Name“ ist der Name der Klasse, zu der die Objekte in der Pipeline gehören. „BaseType“ ist der Name der Oberklasse. .NET unterstützt Vererbung, d.h., eine Klasse kann von einer anderen erben (höchstens von einer anderen Klasse; Mehrfachvererbung gibt es nicht!). Dies ist für die PowerShell meist aber irrelevant und Sie können diese Information ignorieren.

Bei Get-Date() ist ein DateTime-Objekt in der Pipeline. Der zweite Aufruf liefert nur die Information, dass eine Menge von Objekten in der Pipeline ist. Bei der Anwendung von GetType() auf eine Objektmenge in der Pipeline kann man leider noch nicht den Typ erkennen. Hintergrund ist, dass in einer Pipeline Objekte verschiedener Klassen sein können. Der dritte Aufruf, bei dem gezielt ein Objekt (das erste) herausgenommen wird, zeigt dann wieder an, dass es sich um Process-Objekte handelt. Den ganzen Klassennamen inklusive des Namensraums bekommt man nur, wenn man explizit die Eigenschaft FullName abfragt.

Get-PipelineInfo

Das Commandlet Get-PipelineInfo aus den PowerShell Extensions von www.IT-Visions.de liefert drei wichtige Informationen über die Pipeline-Inhalte:

- Anzahl der Objekte in der Pipeline (die Objekte werden durchnummeriert)
- Typ der Objekte in der Pipeline (ganzer Name der .NET-Klasse)
- Zeichenkettenrepräsentation der Objekte in der Pipeline

```

PowerShell - Holger Schwichtenberg (www.IT-Visions.de) - [Running as Administrator] - C:\WINDOWS
5# get-Childitem C:\inetpub\wwwroot | Get-PipelineInfo

      Count  TypeName                               String
-----
1      System.IO.DirectoryInfo             aspnet_client
2      System.IO.DirectoryInfo             images
3      System.IO.DirectoryInfo             www.dotnetframework.de
4      System.IO.DirectoryInfo             www.IT-Visions.de
5      System.IO.DirectoryInfo             www.powershell-doktor.de
6      System.IO.DirectoryInfo             _private
7      System.IO.DirectoryInfo             _vti_log
8      System.IO.FileInfo                   iisstart.htm
9      System.IO.FileInfo                   pageror.gif
10     System.IO.FileInfo                   postinfo.html
11     System.IO.FileInfo                   _vti_inf.html

6#
6#

```

Bild 5.17 Get-PipelineInfo liefert Informationen, dass sich in dem Dateisystemordner elf Objekte befinden. Davon sind sieben Unterordner (Klasse DirectoryInfo) und vier Dateien (Klasse FileInfo).

Das Stichwort Zeichenkettenrepräsentation (Spalte „String“ in der Bildschirmabbildung) ist erklärungsbedürftig: Jedes .NET-Objekt besitzt eine Methode ToString(), die das Objekt in eine Zeichenkette umwandelt, denn ToString() ist in der „Mutter aller .NET-Klassen“ System.Object implementiert und wird an alle .NET-Klassen und somit auch deren Instanzen weitergegeben. Ob ToString() eine sinnvolle Ausgabe liefert, hängt von der jeweiligen Klasse ab. Im Fall von System.Diagnostics.Process werden der Klassenname und der Prozessname ausgegeben. Dies kann man leicht mit `gps | foreach { $_.ToString() }` ermitteln (siehe das nächste Bild). Bei der Klasse System.ServiceProcess.ServiceController, deren Instanzen von Get-Service geliefert werden, ist die Konvertierung hingegen nicht so gut, denn die Zeichenkette enthält nur den Klassennamen, so dass die einzelnen Instanzen gar nicht unterschieden werden können.



HINWEIS: Die Konvertierung in den Klassennamen ist das Standardverhalten, das von System.Object geerbt wird, und dieses Standardverhalten ist leider auch üblich, da sich die Entwickler der meisten .NET-Klassen bei Microsoft nicht die „Mühe“ gemacht haben, eine sinnvolle Zeichenkettenrepräsentanz zu definieren.

ToString() ist üblicherweise keine Serialisierung des kompletten Objektinhalts, sondern im besten Fall nur der „Primärschlüssel“ des Objekts. Theoretisch kann eine .NET-Klasse bei ToString() alle Werte liefern. Das macht aber keine Klasse im .NET Framework. Bei vielen .NET-Klassen liefert ToString() nur den Klassennamen.


```

Administrator: Windows PowerShell
PS C:\> Get-Process | Get-Member

TypeName: System.Diagnostics.Process

Name                MemberType          Definition
-----
Handles             AliasProperty      Handles = HandleCount
Name                AliasProperty      Name = ProcessName
NPM                 AliasProperty      NPM = NonpagedSystemMemorySize
PM                  AliasProperty      PM = PagedMemorySize
UM                  AliasProperty      UM = VirtualMemorySize
WS                  AliasProperty      WS = WorkingSet
Disposed            Event               System.EventHandler Disposed(System.Object, System.EventArgs)
ErrorDataReceived   Event               System.Diagnostics.DataReceivedEventHandler ErrorDataReceived(System.Object, System.EventArgs)
Exited              Event               System.EventHandler Exited(System.Object, System.EventArgs)
OutputDataReceived Event               System.Diagnostics.DataReceivedEventHandler OutputDataReceived(System.Object, System.EventArgs)
BeginErrorReadLine Method               System.Void BeginErrorReadLine()
BeginOutputReadLine Method               System.Void BeginOutputReadLine()
CancelErrorRead     Method               System.Void CancelErrorRead()
CancelOutputRead    Method               System.Void CancelOutputRead()
Close                Method               System.Void Close()
CloseMainWindow     Method               bool CloseMainWindow()
CreateObjRef        Method               System.Runtime.Remoting.ObjRef CreateObjRef(type requestedType)
Dispose             Method               System.Void Dispose()
Equals              Method               bool Equals(System.Object obj)
GetHashCode         Method               int GetHashCode()
GetLifetimeService  Method               System.Object GetLifetimeService()
GetType             Method               type GetType()
InitializeLifetimeService Method           System.Object InitializeLifetimeService()
Kill                Method               System.Void Kill()
Refresh             Method               System.Void Refresh()
Start               Method               bool Start()
ToString            Method               string ToString()
WaitForExit         Method               bool WaitForExit(int milliseconds), System.Void WaitForExit()
WaitForInputIdle   Method               bool WaitForInputIdle(int milliseconds), bool WaitForInputIdle()
__NounName          NoteProperty        System.String __NounName=Process
BasePriority         Property             System.Int32 BasePriority {get;}
Container           Property             System.ComponentModel.IContainer Container {get;}
EnableRaisingEvents Property             System.Boolean EnableRaisingEvents {get;set;}
ExitCode            Property             System.Int32 ExitCode {get;}
ExitTime            Property             System.DateLine ExitTime {get;}
Handle              Property             System.IntPtr Handle {get;}
HandleCount         Property             System.Int32 HandleCount {get;}
HasExited           Property             System.Boolean HasExited {get;}
Id                  Property             System.Int32 Id {get;}
MachineName         Property             System.String MachineName {get;}
MainModule          Property             System.Diagnostics.ProcessModule MainModule {get;}
MainModuleHandle    Property             System.IntPtr MainModuleHandle {get;}
MainModuleTitle     Property             System.String MainModuleTitle {get;}
MaxWorkingSet       Property             System.IntPtr MaxWorkingSet {get;set;}
MinWorkingSet       Property             System.IntPtr MinWorkingSet {get;set;}
Modules             Property             System.Diagnostics.ProcessModuleCollection Modules {get;}

```

Bild 5.20 Teil 1 der Ausgabe von Get-Process | Get-Member

```

Auswählen Administrator: Windows PowerShell
NonpagedSystemMemorySize Property             System.Int32 NonpagedSystemMemorySize {get;}
NonpagedSystemMemorySize64 Property             System.Int64 NonpagedSystemMemorySize64 {get;}
PagedMemorySize     Property             System.Int32 PagedMemorySize {get;}
PagedMemorySize64   Property             System.Int64 PagedMemorySize64 {get;}
PagedSystemMemorySize Property             System.Int32 PagedSystemMemorySize {get;}
PagedSystemMemorySize64 Property             System.Int64 PagedSystemMemorySize64 {get;}
PeakPagedMemorySize Property             System.Int32 PeakPagedMemorySize {get;}
PeakPagedMemorySize64 Property             System.Int64 PeakPagedMemorySize64 {get;}
PeakVirtualMemorySize Property             System.Int32 PeakVirtualMemorySize {get;}
PeakVirtualMemorySize64 Property             System.Int64 PeakVirtualMemorySize64 {get;}
PeakWorkingSet      Property             System.Int32 PeakWorkingSet {get;}
PeakWorkingSet64    Property             System.Int64 PeakWorkingSet64 {get;}
PriorityBoostEnabled Property             System.Boolean PriorityBoostEnabled {get;set;}
PriorityClass        Property             System.Diagnostics.ProcessPriorityClass PriorityClass {get;set;}
PrivateMemorySize   Property             System.Int32 PrivateMemorySize {get;}
PrivateMemorySize64 Property             System.Int64 PrivateMemorySize64 {get;}
PrivilegedProcessorTime Property             System.TimeSpan PrivilegedProcessorTime {get;}
ProcessName          Property             System.String ProcessName {get;}
ProcessorAffinity    Property             System.IntPtr ProcessorAffinity {get;set;}
Responding           Property             System.Boolean Responding {get;}
SessionId            Property             System.Int32 SessionId {get;}
Site                 Property             System.ComponentModel.ISite Site {get;set;}
StandardError        Property             System.IO.StreamReader StandardError {get;}
StandardInput        Property             System.IO.StreamWriter StandardInput {get;}
StandardOutput       Property             System.IO.StreamReader StandardOutput {get;}
StartInfo            Property             System.Diagnostics.ProcessStartInfo StartInfo {get;set;}
StartTime            Property             System.DateLine StartTime {get;}
SynchronizingObject Property             System.ComponentModel.ISynchronizeInvoke SynchronizingObject {get;set;}
Threads              Property             System.Diagnostics.ProcessThreadCollection Threads {get;}
TotalProcessorTime   Property             System.TimeSpan TotalProcessorTime {get;}
UserProcessorTime    Property             System.TimeSpan UserProcessorTime {get;}
VirtualMemorySize    Property             System.Int32 VirtualMemorySize {get;}
VirtualMemorySize64 Property             System.Int64 VirtualMemorySize64 {get;}
WorkingSet           Property             System.Int32 WorkingSet {get;}
WorkingSet64         Property             System.Int64 WorkingSet64 {get;}
PRConfiguration      PropertySet          PRConfiguration (Name, Id, PriorityClass, FileVersion)
PSResources          PropertySet          PSResources (Name, Id, HandleCount, WorkingSet, NonPagedMemorySize, PagedM...
Company              ScriptProperty      System.Object Company {get-$this.Mainmodule.FileVersionInfo.CompanyName;}
CPU                  ScriptProperty      System.Object CPU {get-$this.TotalProcessorTime.TotalSeconds;}
Description           ScriptProperty      System.Object Description {get-$this.Mainmodule.FileVersionInfo.FileDescri...
FileVersion          ScriptProperty      System.Object FileVersion {get-$this.Mainmodule.FileVersionInfo.FileVersion;}
Path                 ScriptProperty      System.Object Path {get-$this.Mainmodule.FileName;}
Product              ScriptProperty      System.Object Product {get-$this.Mainmodule.FileVersionInfo.ProductName;}
ProductVersion       ScriptProperty      System.Object ProductVersion {get-$this.Mainmodule.FileVersionInfo.Product...

```

Bild 5.21 Teil 2 der Ausgabe von Get-Process | Get-Member

Die Ausgabe zeigt, dass aus der Sicht der PowerShell eine .NET-Klasse sieben Arten von Mitgliedern hat:

1. Method (Methode)
2. Property (Eigenschaft)
3. PropertySet (Eigenschaftssatz)
4. NoteProperty (Notizeigenschaft)
5. ScriptProperty (Skripteigenschaft)
6. CodeProperty (Codeeigenschaft)
7. AliasProperty (Aliaseigenschaft)



HINWEIS: Von den oben genannten Mitgliedsarten sind nur „Method“ und „Property“ tatsächliche Mitglieder der .NET-Klasse. Alle anderen Mitgliedsarten sind Zusätze, welche die PowerShell mittels des bereits erwähnten Extended Type System (ETS) dem .NET-Objekt hinzugefügt hat.

Die Ausgabe von `Get-Member` kann man verkürzen, indem man nur eine bestimmte Art von Mitgliedern ausgeben lässt. Diese erreicht man über den Parameter `-MemberType` (kurz: `-m`). Der folgende Befehl listet nur die Properties auf:

```
Get-Process | Get-Member -MemberType Properties
```

Außerdem ist eine Filterung beim Namen möglich:

```
Get-Process | Get-Member *set*
```

Der obige Befehl listet nur solche Mitglieder der Klasse `Process` auf, deren Name das Wort „set“ enthält.

Methoden (Mitgliedsart Method)

Methoden (Mitgliedsart `Method`) sind Operationen, die man auf dem Objekt aufrufen kann und die eine Aktion auslösen, z.B. beendet `Kill()` den Prozess. Methoden können aber auch Daten liefern oder Daten in dem Objekt verändern.



ACHTUNG: Beim Aufruf von Methoden sind immer runde Klammern anzugeben, auch wenn es keine Parameter gibt. Ohne die runden Klammern erhält man Informationen über die Methode, man ruft aber nicht die Methode selbst auf.

Eigenschaften (Mitgliedsart Property)

Eigenschaften (Mitgliedsart `Property`) sind Datenelemente, die Informationen aus dem Objekt enthalten oder mit denen man Informationen an das Objekt übergeben kann, z.B. `MaxWorkingSet`.



ACHTUNG: In PowerShell 1.0 sah die Aussage von Get-Member noch etwas anders aus (siehe nächste Bildschirmabbildung). Man sieht dort, dass es zu jedem Property zwei Methoden gibt, z. B. `get_MaxWorkingSet()` und `set_MaxWorkingSet()`. Die Ursache dafür liegt in den Interna des .NET Frameworks: Dort werden Properties (nicht aber sogenannte Fields, eine andere Art von Eigenschaften) durch ein Methodenpaar abgebildet: eine Methode zum Auslesen der Daten (genannt „Get-Methode“ oder „Getter“), eine andere Methode zum Setzen der Daten (genannt „Set-Methode“ oder „Setter“). Einige Anfänger störte die „Aufblähung“ der Liste durch diese Optionen. Seit PowerShell 2.0 zeigte Get-Member die Getter-Methoden (`get_`) und Setter-Methoden (`set_`) nur noch an, wenn man den Parameter `-force` verwendet.

```

Administrator: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe

PS C:\Windows\System32\WindowsPowerShell\v1.0> Get-Process | Get-Member

TypeName: System.Diagnostics.Process

Name           MemberType      Definition
-----
Handles        AliasProperty   Handles = HandleCount
Name           AliasProperty   Name = ProcessName
NPM            AliasProperty   NPM = NonpagedSystemMemorySize
UM             AliasProperty   UM = UnpagedSystemMemorySize
WS             AliasProperty   WS = WorkingSet
Dispose        Event            System.EventHandler Dispose(System.Object, System.EventArgs)
ErrorDataReceived Event            System.Diagnostics.DataReceivedEventHandler ErrorDataReceived(System.Object, System.EventArgs)
Exited         Event            System.EventHandler Exited(System.Object, System.EventArgs)
OutputDataReceived Event            System.Diagnostics.DataReceivedEventHandler OutputDataReceived(System.Object, System.EventArgs)
BeginErrorReadLine Method           System.Void BeginErrorReadLine()
BeginOutputReadLine Method           System.Void BeginOutputReadLine()
CancelErrorRead Method          System.Void CancelErrorRead()
CancelOutputRead Method          System.Void CancelOutputRead()
Close          Method          System.Void Close()
CloseMainWindow Method          bool CloseMainWindow()
CreateObjRef   Method          System.Runtime.Remoting.ObjRef CreateObjRef(type requestedType)
Dispose        Method          System.Void Dispose()
Equals         Method          bool Equals(System.Object obj)
GetHashCode    Method          int GetHashCode()
GetLifetimeService Method          System.Object GetLifetimeService()
GetType        Method          type GetType()
InitializeLifetimeService Method          System.Object InitializeLifetimeService()
Kill           Method          System.Void Kill()
Refresh        Method          System.Void Refresh()
Start          Method          bool Start()
ToString       Method          string ToString()
WaitForExit   Method          bool WaitForExit(int milliseconds), System.Void WaitForExit()
WaitForInputIdle Method          bool WaitForInputIdle(int milliseconds), bool WaitForInputIdle()
__NounName    NoteProperty    System.String __NounName=Process
BasePriority   Property         System.Int32 BasePriority {get;}
Container      Property         System.ComponentModel.IContainer Container {get;}
EnableRaisingEvents Property         System.Boolean EnableRaisingEvents {get;set;}
ExitCode       Property         System.Int32 ExitCode {get;}
ExitTime      Property         System.DateTime ExitTime {get;}
Handle         Property         System.IntPtr Handle {get;}
HandleCount    Property         System.Int32 HandleCount {get;}
HasExited     Property         System.Boolean HasExited {get;}
Id             Property         System.Int32 Id {get;}
MachineName   Property         System.String MachineName {get;}
MainModule    Property         System.Diagnostics.ProcessModule MainModule {get;}
MainWindowHandle Property         System.IntPtr MainWindowHandle {get;}
MainWindowTitle Property         System.String MainWindowTitle {get;}
MaxWorkingSet Property         System.IntPtr MaxWorkingSet {get;set;}
MinWorkingSet Property         System.IntPtr MinWorkingSet {get;set;}
Modules       Property         System.Diagnostics.ProcessModuleCollection Modules {get;}
  
```

Bild 5.22 Anzeige der Getter und Setter in PowerShell 1.0

Fortgeschrittene Benutzer bevorzugen die Auflistung der Getter und Setter. Man kann erkennen, welche Aktionen auf einem Property möglich sind. Fehlt der Setter, kann die Eigenschaft nicht verändert werden (z. B. `StartTime` bei der Klasse `Process`). Fehlt der Getter, kann man die Eigenschaft nur setzen. Dafür gibt es kein Beispiel in der Klasse `Process`. Dieser Fall kommt auch viel seltener vor, wird aber z. B. bei Kennwörtern eingesetzt, die man nicht wiedergewinnen kann, weil sie nicht im Klartext, sondern nur als Hash-Wert abgespeichert werden.

Für den PowerShell-Nutzer bedeutet die Existenz von Gettern und Settern, dass er zwei Möglichkeiten hat, Daten abzurufen. Über die Eigenschaft (Property):

```
Get-Process | Where-Object { $_.name -eq "iexplore" } | Foreach-Object
{ $_.PriorityClass }
```

oder die entsprechende "Get"-Methode:

```
Get-Process | Where-Object { $_.name -eq "iexplore" } | Foreach-Object
{ $_.get_PriorityClass() }
```

Analog gibt es für das Schreiben die Option über die Eigenschaft:

```
Get-Process | Where-Object { $_.name -eq "iexplore" } | Foreach-Object
{ $_.PriorityClass = "High" }
```

oder die entsprechende "Set"-Methode:

```
Get-Process | Where-Object { $_.name -eq "iexplore" } | Foreach-Object
{ $_.set_PriorityClass("High") }
```



TIPP: Auch hier kann man wieder grundsätzlich die verkürzte Schreibweise seit PowerShell-Version 3.0 anwenden, also:

```
(Get-Process | Where-Object { $_.name -eq "iexplore" }).PriorityClass
(Get-Process | Where-Object { $_.name -eq "iexplore" }).get_PriorityClass()
(Get-Process | Where-Object { $_.name -eq "iexplore" }).set_
PriorityClass("High")
```

Syntaktisch nicht erlaubt ist aber:

```
(Get-Process | Where-Object { $_.name -eq "iexplore" }).PriorityClass =
"High"
```

Hier geht nur die o.g. Schreibweise mit `Foreach-Object`.

Eigenschaftssätze (PropertySet)

Eigenschaftssätze (PropertySet) sind eine Zusammenfassung einer Menge von Eigenschaften unter einem gemeinsamen Dach. Beispielsweise umfasst der Eigenschaftssatz `psResources` alle Eigenschaften, die sich auf den Ressourcenverbrauch eines Prozesses beziehen. Dies ermöglicht es, dass man nicht alle diesbezüglichen Eigenschaften einzeln nennen muss, sondern schreiben kann:

```
Get-Process | Select-Object psResources | Format-Table
```

Die Eigenschaftssätze gibt es nicht im .NET Framework; sie sind eine Eigenart der PowerShell und definiert in der Datei `types.ps1xml` im Installationsordner der PowerShell.



HINWEIS: Man kann einem Objekt zur Laufzeit eine Notizeigenschaft hinzufügen, siehe Kapitel 17 „Dynamische Objekte“.

Skripteigenschaften (ScriptProperty)

Eine **Skripteigenschaft (ScriptProperty)** ist eine berechnete Eigenschaft, also eine Information, die nicht im .NET-Objekt selbst gespeichert ist. Dabei muss die Berechnung nicht notwendigerweise eine mathematische Berechnung sein; es kann sich auch um den Zugriff auf die Eigenschaften eines untergeordneten Objekts handeln. Der Befehl

```
Get-Process | Select-Object name, product
```

listet alle Prozesse mit den Produkten auf, zu denen der Prozess gehört (siehe folgende Bildschirmabbildung). Dies ist gut zu wissen, wenn man auf seinem System einen Prozess sieht, den man nicht kennt und von dem man befürchtet, dass es sich um einen Schädling handeln könnte.

```

Administrator: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
PS C:\Windows\System32\WindowsPowerShell\v1.0>
PS C:\Windows\System32\WindowsPowerShell\v1.0> Get-Process | format-table name, product
Name                                     Product
----                                     -
smss                                     aPO Access Service (64-bit)
cmd                                       Betriebssystem Microsoft® Windows®
conhost                                  Betriebssystem Microsoft® Windows®
explorer                                 Betriebssystem Microsoft® Windows®
Internet Explorer                       Windows® Internet Explorer
Visual Studio .NET                      Betriebssystem Microsoft® Windows®
Microsoft SQL Server                    Betriebssystem Microsoft® Windows®
NVIDIA Driver Helper Service, Version 190.38
NVIDIA Driver Helper Service, Version 190.38
Microsoft Office Outlook                Betriebssystem Microsoft® Windows®
Windows® Search                          Windows® Search
Snagit                                    Betriebssystem Microsoft® Windows®
SnagIt                                    Betriebssystem Microsoft® Windows®
SnagPriv                                  Betriebssystem Microsoft® Windows®
SoundMAX Audio Settings (32-bit)

```

Bild 5.25 Auflistung der berechneten Eigenschaft „Product“

Die Information über das Produkt steht nicht in dem Prozess (Windows listet diese Information im Taskmanager ja auch nicht auf), aber in der Datei, die den Programmcode für den Prozess enthält. Das .NET Framework bietet über die `MainModule.FileVersionInfo.ProductName` einen Zugang zu dieser Information. Anstelle des Befehls

```
Get-Process | Select-Object name, Mainmodule.FileVersionInfo.ProductName
```

bietet Microsoft durch die Skripteigenschaft eine Abkürzung an. Diese Abkürzung ist definiert in der Datei *types.ps1xml* im Installationsordner der PowerShell.

```
<ScriptProperty>
  <Name>Product</Name>
  <GetScriptBlock>$this.MainModule.FileVersionInfo.ProductName</GetScriptBlock>
</ScriptProperty>
```

Bild 5.26 Definition einer Skripteigenschaft in der *types.ps1xml*

Skripteigenschaften gibt es nicht im .NET Framework; sie sind eine Eigenart der PowerShell. Man kann einem Objekt zur Laufzeit eine Skripteigenschaft hinzufügen, siehe Kapitel 17 „Dynamische Objekte“.

Codeeigenschaften (Code Property)

Eine **Codeeigenschaft (CodeProperty)** entspricht einer Script Property, allerdings ist der Programmcode nicht als Skript in der PowerShell-Sprache, sondern als .NET-Programmcode hinterlegt.

Aliaseigenschaft (AliasProperty)

Eine **Aliaseigenschaft (AliasProperty)** ist eine verkürzte Schreibweise für ein Property. Dahinter steckt keine Berechnung, sondern nur eine Verkürzung des Namens. Beispielsweise ist `WS` eine Abkürzung für `WorkingSet`. Auch die Aliaseigenschaften sind in der Datei *types.ps1xml* im Installationsordner der PowerShell definiert. Aliaseigenschaften sind ebenfalls eine PowerShell-Eigenart.

Hintergrundwissen: Extended Type System (ETS)

Wie bereits dargestellt, zeigt die PowerShell für viele .NET-Objekte mehr Mitglieder an, als eigentlich in der .NET-Klasse definiert sind. In einigen Fällen werden aber auch Mitglieder ausgeblendet. In beiden Fällen kommt das Extended Type System (ETS) zum Einsatz.

Die Ergänzung von Mitgliedern per ETS wird verwendet, um bei einigen .NET-Klassen, die Metaklassen für die eigentlichen Daten sind (z.B. `ManagementObject` für WMI-Objekte, `ManagementClass` für WMI-Klassen, `DirectoryEntry` für Einträge in Verzeichnisdiensten und `DataRow` für Datenbankzeilen), die Daten direkt ohne Umweg dem PowerShell-Nutzer zur Verfügung zu stellen.

Mitglieder werden ausgeblendet, wenn sie in der PowerShell nicht nutzbar sind oder es bessere Alternativen durch die Ergänzungen gibt.

In der Dokumentation nimmt das PowerShell-Entwicklungsteam dazu wie folgt Stellung: „Some .NET Object members are inconsistently named, provide an insufficient set of public members, or provide insufficient capability. ETS resolves this issue by introducing the ability to extend the .NET object with additional members.“ [MSDN54] Dies heißt im Klartext, dass das PowerShell-Team mit der Arbeit des Entwicklungsteams der .NET-Klassenbibliothek nicht ganz zufrieden ist.

Das Extended Type System (ETS) verpackt grundsätzlich jedes Objekt, das von einem Commandlet in die Pipeline gelegt wird, in ein PowerShell-Objekt des Typs `PSObject`. Die Imple-

mentierung der Klasse PSObject entscheidet dann, was für die folgenden Commandlets und Befehle sichtbar ist.

Diese Entscheidung wird beeinflusst durch verschiedene Instrumente:

- PowerShell-Objektadapter, die für bestimmte Typen wie ManagementObject, Management Class, DirectoryEntry und DataRow implementiert wurden,
- die Deklarationen in der *types.ps1xml*-Datei,
- in den Commandlets hinzugefügte Mitglieder,
- mit dem Commandlet Add-Member hinzugefügte Mitglieder.

■ 5.12 Filtern

Nicht immer will man alle Objekte weiterverarbeiten, die ein Commandlet liefert. Einschränkungskriterien sind Bedingungen (z. B. nur Prozesse, bei denen der Speicherbedarf größer ist als 10 000 000 Byte) oder die Position (z. B. nur die fünf Prozesse mit dem größten Speicherbedarf). Zur wertabhängigen Einschränkung verwendet man das Commandlet Where-Object (Alias where).

```
Get-Process | Where-Object {$_.ws -gt 10000000 }
```

Einschränkungen über die Position definiert man mit dem Select-Object (in dem nachfolgenden Befehl für das oben genannte Beispiel ist zusätzlich noch eine Sortierung eingebaut, damit die Ausgabe einen Sinn ergibt):

```
Get-Process | Sort-Object ws -desc | Select-Object -first 5
```

Analog dazu sind die kleinsten Speicherfresser zu ermitteln mit:

```
Get-Process | Sort-Object ws -desc | Select-Object -last 5
```

Etwas gewöhnungsbedürftig ist die Schreibweise der Vergleichsoperatoren: Statt >= schreibt man -ge (siehe Tabelle 5.1). Die Nutzung regulärer Ausdrücke ist möglich mit dem Operator -Match.

Dazu zwei **Beispiele**:

1. Der folgende Ausdruck listet alle Systemdienste, deren Beschreibung aus zwei durch ein Leerzeichen getrennten Wörtern besteht.

```
Get-Service | Where-Object {$_.DisplayName -match "^\w+ \w+$" }
```

```

PS T:\> Get-Service | Where-Object { $_.DisplayName -match "\w+ \w+$" }

-----
Status      Name              DisplayName
-----
Stopped    AppIDSvc          Application Identity
Running    AppInfo           Application Information
Running    AppMgmt           Application Management
Stopped    AppReadiness     App Readiness
Running    Audiosrv          Windows Audio
Running    Browser           Computer Browser
Running    CertPropSvc       Certificate Propagation
Running    CryptSvc          Cryptographic Services
Running    CscService        Offline Files
Stopped    defragsvc         Optimize drives
Running    Dhcp              DHCP Client
Running    Dnscache          DNS Client
Running    DoSvc             Delivery Optimization
Stopped    dot3svc           Wired AutoConfig
Running    DismSvc           Data Usage
Stopped    embeddedmode      Embedded Mode
    
```

Bild 5.27 Ausgabe zu obigem Beispiel

2. Der folgende Ausdruck listet alle Prozesse, deren Namen mit einem "i" starten und danach aus drei Buchstaben bestehen.

```
Get-Process | Where-Object { $_.ProcessName -match "^i\w{3}$" }
```

```

PS H:\> Get-Process | Where-Object { $_.ProcessName -match "^i\w{3}$" }

-----
Handles  NPM(K)  PM(K)  WS(K)  UM(M)  CPU(s)  Id ProcessName
-----
0         0        0       24     0       0        0 Idle
    
```

Bild 5.28 Ausgabe zu obigem Beispiel

Tabelle 5.1 Vergleichsoperatoren der PowerShell

Vergleich unter Ignorierung der Groß-/ Kleinschreibung	Vergleich unter Berücksichtigung der Groß-/ Kleinschreibung	Bedeutung
-lt / -ilt	-clt	Kleiner
-le / -ile	-cle	Kleiner oder gleich
-gt / -igt	-cgt	Größer
-ge / -ige	-cge	Größer oder gleich
-eq / -ieq	-ceq	Gleich
-ne / -ine	-cne	Nicht gleich
-like / -ilike	-clike	Ähnlichkeit zwischen Zeichenketten, Einsatz von Platzhaltern (* und ?) möglich
-notlike / -inotlike	-cnotlike	Keine Ähnlichkeit zwischen Zeichenketten, Einsatz von Platzhaltern (* und ?) möglich
-match / -imatch	-cmatch	Vergleich mit regulärem Ausdruck
-notmatch / -inotmatch	-cnotmatch	Stimmt nicht mit regulärem Ausdruck überein

Vergleich unter Ignorierung der Groß-/ Kleinschreibung	Vergleich unter Berücksichtigung der Groß-/ Kleinschreibung	Bedeutung
-is		Typvergleich, z. B. (Get-Date) -is [DateTime]
-in -contains		Ist enthalten in Menge
-notin -notcontains		Ist nicht enthalten in Menge

Tabelle 5.2 Logische Operatoren in der PowerShell-Sprache

Logischer Operator	Bedeutung
-not oder !	Nicht
-and	Und
-or	Oder

Vereinfachte Schreibweise von Bedingungen seit PowerShell-Version 3.0

Microsoft hat versucht, die Schreibweise von Bedingungen nach Where-Object seit PowerShell-Version 3.0 zu vereinfachen.

Die Bedingung

```
Get-Service | where-object { $_.status -eq "running" }
```

kann der Nutzer seitdem vereinfacht schreiben als

```
Get-Service | where-object status -eq "running".
```

Dass auch

```
Get-Service | where-object -eq status "running"
```

und

```
Get-Service | where-object status "running" -eq
```

zum gleichen Ergebnis führen, wirkt befremdlich.

Allerdings funktioniert die neue Syntaxform nur in den einfachsten Fällen. Bei der Verwendung von -and und -or ist die Verkürzung nicht möglich.

So sind folgende Befehle **nicht** erlaubt:

```
Get-Process | Where-Object Name -eq "iexplore" -or name -eq "Chrome" -or name -eq "Firefox" | Stop-Process
```

```
Get-Service | where-object status -eq running -and name -like "a*"
```

Korrekt muss es heißen:

```
Get-Process | Where-Object { $_.Name -eq "iexplore" -or $_.name -eq "Chrome" -or
$_name -eq "Firefox" } | Stop-Process

Get-Service | where-object { $_.status -eq "running" -and $_.name -like "a*" }
```

Grund für das Versagen bei komplexeren Ausdrücken ist, dass Microsoft die Syntaxvereinfachung über die Parameter abgebildet hat. So wird in der einfachsten Form `-eq` als Parameter von `where-object` betrachtet. Microsoft hätte da lieber den Parser grundsätzlich überarbeiten sollen.

■ 5.13 Zusammenfassung von Pipeline-Inhalten

Die Menge der Objekte in der Pipeline kann heterogen sein, d. h. verschiedenen .NET-Klassen angehören. Dies ist zum Beispiel automatisch der Fall, wenn man `Get-ChildItem` im Dateisystem ausführt: Die Ergebnismenge enthält sowohl `FileInfo`- als auch `DirectoryInfo`-Objekte.

Man kann auch zwei Befehle, die beide Objekte in die Pipeline senden, zusammenfassen, so dass der Inhalt in einer Pipeline wie folgt aussieht:

```
$( Get-Process ; Get-Service )
```

Dies ist aber nur sinnvoll, wenn die nachfolgenden Befehle in der Pipeline korrekt mit heterogenen Pipeline-Inhalten umgehen können. Die Standardausgabe der PowerShell kann dies. In anderen Fällen bedingt der Typ des ersten Objekts in der Pipeline die Art der Weiterverarbeitung (z. B. bei `Export-Csv`).

```
PowerShell - Holger Schwichtenberg (www.IT-Visions.de) - [Running as Administrator] - H:\DEV\ITVisions_PowerShell_CommandletLibrary\CommandletLibrary...
15# $( Get-Process i* ; Get-Service i* ) | Get-PipelineInfo

```

Count	TypeName	String
1	System.Diagnostics.Process	System.Diagnostics.Process (Idle)
2	System.Diagnostics.Process	System.Diagnostics.Process (iexplore)
3	System.Diagnostics.Process	System.Diagnostics.Process (inetinfo)
4	System.Diagnostics.Process	System.Diagnostics.Process (ISRSvc)
5	System.ServiceProcess.ServiceController	System.ServiceProcess.ServiceController
6	System.ServiceProcess.ServiceController	System.ServiceProcess.ServiceController
7	System.ServiceProcess.ServiceController	System.ServiceProcess.ServiceController
8	System.ServiceProcess.ServiceController	System.ServiceProcess.ServiceController
9	System.ServiceProcess.ServiceController	System.ServiceProcess.ServiceController

```
16#
```

Bild 5.29 Anwendung von `Get-PipelineInfo` auf eine heterogene Pipeline

■ 5.14 „Kastrierung“ von Objekten in der Pipeline

Die Analyse des Pipeline-Inhalts zeigt, dass es oftmals sehr viele Mitglieder in den Objekten in der Pipeline gibt. In der Regel braucht man aber nur wenige. Nicht nur aus Gründen der Leistung und Speicherschonung, sondern auch in Bezug auf die Übersichtlichkeit lohnt es sich, die Objekte in der Pipeline hinsichtlich ihrer Datenmenge zu beschränken.

Mit dem Befehl `Select-Object` (Alias: `Select`) kann ein Objekt in der Pipeline „kastriert“ werden, d. h., (fast) alle Mitglieder des Objekts werden aus der Pipeline entfernt, mit Ausnahme der hinter `Select-Object` genannten Mitglieder.

Beispiel:

```
Get-Process | Select-Object processname, get_minworkingset, ws | Get-Member
```

lässt von den `Process`-Objekten in der Pipeline nur die Mitglieder `processname` (Eigenschaft), `get_minworkingset` (Methode) und `workingset` (Alias) übrig (siehe folgende Bildschirmabbildung). Wie das Bild zeigt, ist das „Kastrieren“ mit drei Wermutstropfen verbunden:

- `Get-Member` zeigt nicht mehr den tatsächlichen Klassennamen an, sondern `PSCustomObject`, eine universelle Klasse der PowerShell.
- Alle Mitglieder sind zu Notizeigenschaften degradiert.

```
PS C:\Windows\System32\WindowsPowerShell\v1.0> Get-Process | select-object processname, get_minworkingset, ws | get-member
TypeName: Selected.System.Diagnostics.Process
Name      MemberType Definition
-----
Equals    Method     bool Equals(System.Object obj)
GetHashCode Method     int GetHashCode()
GetType   Method     type GetType()
ToString  Method     string ToString()
get_minworkingset NoteProperty get_minworkingset=null
ProcessName NoteProperty System.String ProcessName=0EAD18RU
WS        NoteProperty System.Int32 WS=4030464
PS C:\Windows\System32\WindowsPowerShell\v1.0>
```

Bild 5.30 Wirkung der Anwendung von `Select-Object`



TIPP: Mit dem Parameter `-exclude` kann man in `Select-Object` auch Mitglieder einzeln ausschließen.

Dass es neben den drei gewünschten Mitgliedern noch vier weitere in der Liste gibt, ist auch einfach erklärbar: Jedes, wirklich jedes `.NET`-Objekt hat diese vier Methoden, weil diese von der Basisklasse `System.Object` an jede `.NET`-Klasse vererbt und damit an jedes `.NET`-Objekt weitergegeben werden.

■ 5.15 Sortieren

Mit `Sort-Object` (Alias `Sort`) sortiert man die Objekte in der Pipeline nach den anzugebenden Eigenschaften. Die Standardsortierrichtung ist aufsteigend. Mit dem Parameter `-descending` (kurz: `-desc`) legt man die absteigende Sortierung fest.

Der folgende Befehl sortiert die Prozesse absteigend nach ihrem Speicherverbrauch:

```
Get-Process | Sort-Object workingset64 -desc
```

Mit Komma getrennt kann man mehrere Eigenschaften aufführen, nach denen sortiert werden soll. In folgendem Beispiel werden die Systemdienste erst nach Status und innerhalb eines Status dann nach Displayname sortiert.

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

Auch Listen elementarer Datentypen lassen sich sortieren. Hier muss man keine Eigenschaft angeben, nach der man sortieren will:

```
21, 32, 16, 34, 9, 10 | Sort-Object
```

Möchte man diese Zahlen nicht numerisch, sondern alphabetisch sortieren, dann gibt man als Parameter einen Ausdruck an, der eine Typkonvertierung mit einem Typbezeichner (Details zu Typkonvertierungen erfahren Sie im Kapitel 7 „*PowerShell-Skriptsprache*“) enthält:

```
21, 32, 16, 34, 9, 10 | Sort-Object { [string]$_ }
```

```

PowerShell
PS T:\> 21, 32, 16, 34, 9, 10 | Sort-Object
9
10
16
21
32
34
PS T:\> 21, 32, 16, 34, 9, 10 | Sort-Object { [string]$_ }
10
16
21
32
34
9
PS T:\>

```

Bild 5.31 Numerische versus alphabetische Sortierung von sechs Zahlen

■ 5.16 Duplikate entfernen

Get-Unique entfernt Duplikate aus einer Liste.

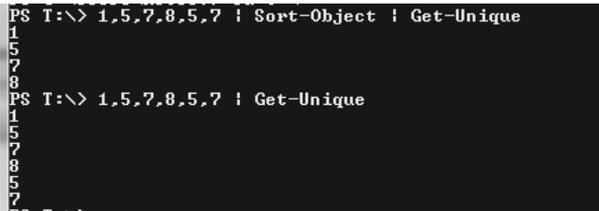
Achtung: Die Liste muss vorher sortiert sein!

Richtig ist daher:

```
1,5,7,8,5,7 | Sort-Object | Get-Unique
```

Falsch wäre:

```
1,5,7,8,5,7 | Get-Unique
```



```
PS T:\> 1,5,7,8,5,7 | Sort-Object | Get-Unique
1
5
7
8
PS T:\> 1,5,7,8,5,7 | Get-Unique
1
5
7
8
5
7
```

Bild 5.32

Richtiger und falscher Einsatz von Get-Unique



TIPP: Get-Unique arbeitet nicht nur auf elementaren Datentypen wie Zahlen und Zeichenketten, sondern auch auf komplexen Objekten, z. B. Get-Process | Sort-Object | Get-Unique.

Praxisbeispiel: Microsoft-Office-Wörterbücher zusammenfassen

Wer auf mehreren Rechnern arbeitet und kein Roaming-Profil nutzen kann oder will, kennt das Problem: Auf jedem PC gibt es ein eigenes benutzerdefiniertes Wörterbuch für Microsoft Word, Outlook etc. (.dic-Datei mit Namen benutzer.dic bzw. custom.dic). .Dic-Dateien sind einfache ASCII-Dateien und man kann natürlich mit jedem beliebigen Texteditor oder einem Merge-Werkzeug die Wörterbücher zusammenführen. Ganz elegant ist die Zusammenführung aber mit einem PowerShell-Einzeiler möglich. Der Befehl geht davon aus, dass sich im Ordner d:\Woerterbuecher mehrere .dic-Dateien befinden. Die Ausgabe ist ein konsolidiertes Wörterbuch MeinWoerterbuch.dic. Doppelte Einträge werden natürlich mit Get-Unique eliminiert.

```
Dir "T:\Woerterbuecher" -Filter *.dic | Get-Content | Sort-Object | Get-Unique |
Set-Content "T:\Woerterbuecher\MeinWoerterbuch.dic"
```

■ 5.17 Gruppierung

Mit Group-Object (Alias: Group) kann man Objekte in der Pipeline nach Eigenschaften gruppieren.

Mit dem folgenden Befehl ermittelt man, wie viele Systemdienste laufen und wie viele gestoppt sind:

```
Get-Service | Group-Object status
```

Dabei liefert das Commandlet drei Spalten (siehe nächste Bildschirmabbildung): Count, Name und Group (mit den Elementen in der Gruppe). Über die Eigenschaft Group kann man dann die Gruppenmitglieder abrufen, z.B. die Mitglieder der ersten Gruppe (Zählung beginnt bei 0, runde Klammern nicht vergessen):

```
(Get-Service | Group-Object status)[0].Group
```

Braucht man die Gruppenmitglieder nicht, verwendet man als Zusatz -NoElement (das spart etwas Speicherplatz, was aber nur bei großen Ergebnismengen relevant ist):

```
Get-Service | Group-Object status -NoElement
```

Ein weiteres Beispiel gruppiert die Dateien im System32-Verzeichnis nach Dateierweiterung und sortiert die Gruppierung dann absteigend nach Anzahl der Dateien in jeder Gruppe.

```
Get-ChildItem c:\windows\system32 | Group-Object extension |  
Sort-Object count -desc
```

```
PowerShell
PS T:\> Get-Service | Group-Object status
Count Name Group
-----
124 Running {AdobeARMSvc, AMD External Events Utility, AntiVirusKit Client, AppHostSvc...}
143 Stopped {AJRouter, ALG, AppIDSvc, AppMgmt...}

PS T:\> Get-Service | Group-Object status -NoElement
Count Name
-----
124 Running
143 Stopped

PS T:\> Get-ChildItem c:\windows\system32 | Group-Object extension | Sort-Object count -desc
Count Name Group
-----
3420 .dll {aadauthhelper.dll, aadcloudap.dll, aadjcsp.dll, aadtb.dll...}
671 .exe {acu.exe, AgentService.exe, aitstatic.exe, alg.exe...}
138 {0409, 1029, 1033, 1036...}
120 .NLS {C_037.NLS, C_10000.NLS, C_10001.NLS, C_10002.NLS...}
42 .msc {adsiedit.msc, azman.msc, certlm.msc, certmgr.msc...}
30 .dat {ande31a.dat, amdcdxx.dat, atiicdxx.dat, ativce02.dat...}
18 .cpl {appwiz.cpl, bchprops.cpl, desk.cpl, Firewall.cpl...}
17 .png {@AudioToastIcon.png, @BackgroundAccessToastIcon.png, @bitlockertoastimage.png, @edp...}
15 .tlb {activeds.tlb, amcompat.tlb, mqa0.tlb, mqa010.tlb...}
15 .ax {bdaplgin.ax, g7llcodc.ax, ksproxy.ax, kstvtune.ax...}
14 .mof {hypervisor.mof, msmqpub.mof, msmqtrc.mof, msmqtrcRemove.mof...}
13 .xml {AppDatabase.xml, AppxProvisioning.xml, DefaultParameters.xml, LServer_PKConfig.xml...}
13 .rs {cero.rs, cob-au.rs, csrr.rs, djctg.rs...}
8 .uce {bopomof.uce, gb2312.uce, ideograf.uce, kanji_l.uce...}
7 .bin {AverageRoom.bin, DefaultTrifs.bin, edgehtmlpluginpolicy.bin, LargeRoom.bin...}
6 .scr {Bubbles.scr, Mystify.scr, PhotoSaver.scr, Ribbons.scr...}
6 .ocx {dmview.ocx, hhctrl.ocx, msdxm.ocx, sysmon.ocx...}
6 .acm {imaadp32.acm, l3codeca.acm, l3codecp.acm, msadp32.acm...}
5 .xsl {dfsrHealthReport.xsl, dfsrPropagationReport.xsl, EventViewer_EventDetails.xsl, WsmP...}
5 .com {chcp.com, format.com, mode.com, more.com...}
5 .config {AppVStreamngUX.exe.config, ClusterUpdateUI.exe.config, DfsMgmt.dll.config, dsac.ex...}
4 .tsp {hidphone.tsp, kmddsp.tsp, remotesp.tsp, unimda.tsp}
```

Bild 5.33 Einsatz von Group-Object



TIPP: Wenn es nur darum geht, die Gruppen zu ermitteln und nicht die Häufigkeit der Gruppenelemente, dann kann man auch `Select-Object` mit dem Parameter `-unique` zum Gruppieren einsetzen:

```
Get-ChildItem | Select-Object extension -Unique
```



TIPP: Man kann bei `Group-Object` auch einen Ausdruck angeben, der wahr oder falsch liefert, und dadurch zwei Gruppen bilden.



BEISPIEL:

```
Get-ChildItem c:\Windows | Where { !$_.PsIsContainer } |
Group-Object { $_.Length -gt 1MB}
```

teilt alle Dateien im aktuellen Verzeichnis in zwei Gruppen ein: solche, die größer als 1 MByte sind, und solche, die es nicht sind (Verzeichnisse werden bereits vorher ausgeschlossen, auch wenn dies nicht erforderlich wäre, da sie die Größe 0 besitzen).

```
PS C:\Users\hs.ITU> Get-Childitem c:\Windows | Where { !$_.PsIsContainer } | Group-Object { $_.Length -gt 1MB}
Count Name Group
-----
46 False <Rscd_tmp.ini, hfsuc.exe, bootstat.dat, DtcInstall.log...>
2 True <explorer.exe, WindowsUpdate.log>
```

Bild 5.34 Ergebnis des obigen Befehls (Zahlen können in Abhängigkeit vom Betriebssystem abweichen)

Praxisbeispiel

Wenn man sich die Elemente der einzelnen Gruppen liefern lässt, so kann man diese weiterverwenden, indem man über die Eigenschaft `group` mit `Foreach-Object` iteriert.

Beispiel: Ermittle aus dem Verzeichnis `System32` alle Dateien, die mit dem Buchstaben „b“ beginnen. Beschränke die Menge auf diejenigen Dateien, die größer als 40 000 Byte sind, und gruppier die Ergebnismenge nach Dateierweiterungen. Sortiere die Gruppen nach der Anzahl der Einträge absteigend und beschränke die Menge auf das oberste Element. Gib für alle Mitglieder dieser Gruppe die Attribute `Name` und `Length` aus und passe die Spaltenbreite automatisch an.

```
Get-ChildItem c:\windows\system32 -filter b*.* | Where-Object { $_.Length -gt 40000 } |
Group-Object Extension | Sort-Object count -desc | Select-Object -first 1 | Select-
Object group | foreach { $_.group } | Select-Object name,length | Format-Table -
autosize
```

■ 5.18 Berechnungen

Measure-Object (Alias: measure) führt verschiedene Berechnungen (Anzahl, Durchschnitt, Summe, Minimum, Maximum) für Objekte in der Pipeline aus. Dabei sollte man die Eigenschaft nennen, über welche die Berechnung ausgeführt werden soll. Sonst wird die erste Eigenschaft verwendet, die aber häufig ein Text ist, den man nicht mathematisch verarbeiten kann.

Measure-Object liefert im Standard nur die Anzahl. Mit den Parametern `-sum`, `-min`, `-max` und `-average` muss man weitere Berechnungen explizit anstoßen.

Beispiel: Informationen über die Dateien in `c:\Windows`

```
Get-ChildItem c:\windows | Measure-Object -Property length -min -max -average -sum
```

```
PS C:\Windows\System32\WindowsPowerShell\v1.0> Get-Childitem c:\Windows\system32 | measure-object -Property length -min -max -average -sum
Count           : 2598
Average         : 465515.188030888
Sum             : 1205684337
Maximum        : 26575296
Minimum         : 35
Property        : length
PS C:\Windows\System32\WindowsPowerShell\v1.0>
```

Bild 5.35 Beispiel für den Einsatz von Measure-Object

■ 5.19 Zwischenschritte in der Pipeline mit Variablen

Ein Befehl mit Pipeline kann beliebig lang und damit auch beliebig komplex werden. Wenn der Befehl unübersichtlich wird oder man Zwischenschritte genauer betrachten möchte, bietet es sich an, den Inhalt der Pipeline zwischenspeichern. Die PowerShell ermöglicht es, den Inhalt der Pipeline in Variablen abzulegen. Variablen werden durch ein vorangestelltes Dollarzeichen [`$`] gekennzeichnet. Anstelle von

```
Get-Process | Where-Object { $_.name -eq "iexplore" } | Foreach-Object { $_.ws }
```

kann man die folgenden Befehle nacheinander in getrennte Zeilen eingeben:

```
$x = Get-Process
$y = $x | Where-Object { $_.name -eq "iexplore" }
$y | Foreach-Object { $_.ws }
```

Das Ergebnis ist in beiden Fällen gleich.

Der Zugriff auf Variablen, die keinen Inhalt haben, führt so lange nicht zum Fehler, wie man später in der Pipeline keine Commandlets verwendet, die unbedingt Objekte in der Pipeline erwarten.

```

Administrator: Windows PowerShell
PS T:\> $x
PS T:\> $x | get-member
get-member : Sie müssen ein Objekt für das Cmdlet "Get-Member" angeben.
In Zeile:1 Zeichen:6
+ $x | get-member
+ ~~~~~
+ CategoryInfo          : CloseError: (:) [Get-Member], InvalidOperationException
+ FullyQualifiedErrorId : NoObjectInGetMember,Microsoft.PowerShell.Commands.GetMemberCommand
PS T:\> _

```

Bild 5.36 Zugriff auf Variablen ohne Inhalt



ACHTUNG: Wenn ein Pipeline-Befehl keinen Inhalt liefert, dann erhält die Variable den Wert `$null`, der für „kein Wert“ steht.

Beispiel:

```
$x = Get-Service x*
```

Die Ausgabe für `$null` ist nichts.

■ 5.20 Verzweigungen in der Pipeline

Manchmal möchte man innerhalb einer Pipeline das Ergebnis nicht nur in der Pipeline weiterreichen, sondern auch in einer Variablen oder im Dateisystem zwischenspeichern. PowerShell bietet dafür verschiedene Möglichkeiten.



TIPP: Verzweigungen in der Pipeline lassen sich ganz einfach abbilden, indem man die Zwischenschritte in verschiedenen Variablen ablegt, auf die man später wieder zugreifen kann. Die in diesem Unterkapitel gezeigten Techniken sind für Leute gedacht, die unbedingt möglichst viel in einem einzigen Pipeline-Befehl unterbringen wollen.

Tee-Object

Der Verzweigung innerhalb der Pipeline dient das Commandlet `Tee-Object`, wobei hier das „Tee“ für „verzweigen“ steht. `Tee-Object` reicht den Inhalt der Pipeline unverändert zum nächsten Commandlet weiter, bietet aber an, den Inhalt der Pipeline wahlweise zusätzlich in einer Variablen oder im Dateisystem abzulegen.

Der folgende Pipeline-Befehl verwendet `Tee-Object` gleich zweimal für beide Anwendungsfälle:

```
Get-Service | Tee-Object -var a | Where-Object { $_.Status -eq "Running" } | select
name | Tee-Object -filepath t:\dienste.txt | ft name
```

Die erste Verwendung von Tee-Object speichert die Liste der Dienste-Objekte in der Variablen \$a und gibt die Objekte aber gleichzeitig weiter in die Pipeline.

Die zweite Verwendung speichert die Liste der laufenden Dienste in der Textdatei g:\dienste.txt und gibt sie zusätzlich an die Standardausgabe aus.

Nach der Ausführung des Befehls steht in der Variablen \$a eine Liste aller Dienste und in der Textdatei *dienste.txt* eine Liste der laufenden Dienste.



ACHTUNG: Bitte beachten Sie, dass man bei Tee-Object beim Parameter `-variable` den Namen der Variablen ohne den üblichen Variablenkennzeichner "\$" angeben muss.

Parameter -OutVariable

Alternativ zum Commandlet Tee-Object kann man den allgemeinen Parameter `-OutVariable` (kurz: `-ov`) einsetzen, der das Ergebnis eines Commandlets in einer Variable ablegt und dennoch das Ergebnis in der Pipeline weiterreicht. Das Beispiel aus dem vorherigen Unterkapitel kann man so umformulieren:

```
Get-Service -OutVariable a | Where-Object { $_.Status -eq "Running" } | select name |
Set-Content t:\dienste.txt -PassThru | ft name
```

Anders als Tee-Object kann `-OutVariable` nichts direkt in einer Datei speichern. Zum Speichern kommt daher hier `Set-Content` zum Einsatz mit `-PassThru`, was ein zusätzliches Durchleiten der Ergebnisse bewirkt.



ACHTUNG: Nach `-OutVariable` ist von der Variablen nur der Name anzugeben. Das Dollarzeichen muss weggelassen werden.

Parameter -PipelineVariable

Der mit PowerShell-Version 4.0 eingeführte allgemeine Parameter `-PipelineVariable` (kurz: `-pv`) sorgt dafür, dass das jeweils aktuelle Objekt nicht nur in der Pipeline weitergeleitet wird, sondern zusätzlich auch in einer Variablen abgelegt wird. Dies ist immer dann sinnvoll, wenn die Pipeline ein Objekt in seiner Struktur verändert (z.B. `SelectObject`), man aber später noch auf den früheren Zustand zugreifen will. Nach `-PipelineVariable` ist von der Variablen nur der Name anzugeben. Das Dollarzeichen muss weggelassen werden.

Beispiel 1

Das folgende Beispiel setzt dies ein, um am Ende eine Liste von Ausgaben aus zwei verschiedenen Objekten zu liefern: den Namen und das Workingset eines Prozesses von `Get-Process` und den Namen und den zugehörigen Security Identifier des Benutzers, unter dem der Prozess läuft. Die Pipeline beginnt mit dem Holen der laufenden Prozesse unter Einbeziehung der Benutzeridentität, die in der Form „Domäne\Benutzername“ geliefert wird. Dabei

wird das aktuelle Process-Objekt mit `-pv` auch in der Variablen `$p` abgelegt. Im zweiten Schritt wird für den Benutzernamen das zugehörige WMI-Objekt `Win32_User` geholt. Im dritten Pipeline-Schritt werden dann zuerst die zwei Informationen aus dem Process-Objekt ausgegeben (das sich in `$p` befindet) sowie die Informationen aus dem `Win32_UserAccount`-Objekt, die sich nun in der Pipeline befinden (`$_`).

```
Get-Process -IncludeUserName -pv p | % { Get-WmiObject Win32_UserAccount -filter
"name='$(($_.username -split "\\")[1])'" } | % { $p.name + ";" + $p.ws + ":" +
$_.Name + ";" + $_.SID }
```



ACHTUNG: Der Parameter `-PipelineVariable` funktioniert nicht wie gewünscht, wenn Commandlets in der Pipeline sind, die die Ergebnisse puffern (z. B. `Sort-Object`, `Group-Object`), da der Parameter `-PipelineVariable` sich ja immer nur auf das aktuelle Objekt bezieht, was in diesen Fällen also immer das letzte Objekt ist.

Beispiel 2

Der folgende Einzeiler listet alle 64516-IP-Adressen zwischen 192.168.0.0 und 192.168.254.254 auf.

```
1..254 | Foreach-Object -PipelineVariable x { $_ } | Foreach-Object { 1..254 } |
foreach-Object { "192.168.$x.$_" }
```

■ 5.21 Vergleiche zwischen Objekten

Mit `Compare-Object` kann man den Inhalt von zwei Pipelines vergleichen. Mit der folgenden Befehlsfolge werden alle zwischenzeitlich neu gestarteten Prozesse ausgegeben:

```
$ProzesseVorher = Get-Process
# Hier einen Prozess starten
$ProzesseNacher = Get-Process
Compare-Object $ProzesseVorher $ProzesseNacher
```

```

c:\ PoSh C:\WINDOWS
Windows PowerShell
Copyright (C) 2006 Microsoft Corporation. All rights reserved.

1# $vorher = get-process
2# notepad
3# notepad
4# mmc
5# $nachher = Get-process
6# compare-object $vorher $nachher

InputObject                                     SideIndicator
-----
System.Diagnostics.Process (mmc)                 =>
System.Diagnostics.Process (notepad)             =>
System.Diagnostics.Process (notepad)             =>

7#

```

Bild 5.37 Vergleich von zwei Pipelines

■ 5.22 Zusammenfassung

Die folgende Tabelle zeigt eine Übersicht der wichtigsten Pipelining-Commandlets.

Tabelle 5.3 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
Sort-Object (sort)	Sortieren der Objekte
Group-Object (group)	Gruppieren der Objekte
Start-Process/Stop-Process	Prozess starten/beenden
Foreach-Object { \$_... } (%)	Schleife über alle Objekte
Get-Member (gm)	Ausgabe der Metadaten (Reflection)
Measure-Object (measure)	Berechnung: -min -max -sum -average
Compare-Object (compare, diff)	Vergleichen von zwei Objektmengen

■ 5.23 Praxisbeispiele

Dieses Kapitel enthält einige Beispiele für die Anwendung von Pipelining und Ausgabebefehlen:

- Beende durch Aufruf der Methode `Kill()` alle Prozesse, die „chrome“ heißen, wobei die Groß-/Kleinschreibung des Prozessnamens irrelevant ist.

```
Get-Process | Where { $_.processname -ieq "chrome" } | foreach { $_.Kill() }
```

oder synonym und kürzer:

```
(Get-Process "chrome").Kill()
```

- Sortiere die Prozesse, die das Wort „chrome“ im Namen tragen, gemäß ihrer CPU-Nutzung und beende den Prozess, der in der aufsteigenden Liste der CPU-Nutzung am weitesten unten steht (also am meisten Rechenleistung verbraucht).

```
Get-Process | Where { $_.processname -ilike "*chrome*" } | Sort-Object -property cpu | Select-Object -last 1 | foreach { $_.Kill() }
```

- Gib die Summe der Speichernutzung aller Prozesse aus.

```
ps | Measure-Object workingset
```

- Gruppier die Einträge im System-Ereignisprotokoll nach Benutzernamen.

```
Get-EventLog -logname system | Group-Object username
```

- Zeige die letzten zehn Einträge im System-Ereignisprotokoll.

```
Get-EventLog -logname system | Select-Object -last 10
```

- Zeige für die letzten zehn Einträge im System-Ereignisprotokoll die Quelle an.

```
Get-EventLog -logname system | Select-Object -first 10 | Select-Object source
```

- Importiere die Textdatei `test.txt`, wobei die Textdatei als eine CSV-Datei mit dem Semikolon als Trennzeichen zu interpretieren ist und die erste Zeile die Spaltennamen enthalten muss. Zeige daraus die Spalten *ID* und *Url*.

```
Import-Csv d:\_work\test.txt -delimiter ";" | Select-Object ID,Url
```

- Ermittle aus dem Verzeichnis `System32` alle Dateien, die mit dem Buchstaben „a“ beginnen. Beschränke die Menge auf diejenigen Dateien, die größer als 40 000 Byte sind, und gruppier die Ergebnismenge nach Dateinamenerweiterungen. Sortiere die gruppierte Menge nach dem Namen der Dateierweiterung.

```
Get-ChildItem c:\windows\system32 -filter a*. * | Where-Object {$_.Length -gt 40000} | Group-Object Extension | Sort-Object name | Format-Table
```

- Ermittle aus dem Verzeichnis System32 alle Dateien, die mit dem Buchstaben „b“ beginnen. Beschränke die Menge auf diejenigen Dateien, die größer als 40 000 Byte sind, und gruppier die Ergebnismenge nach Dateierweiterungen. Sortiere die Gruppen nach der Anzahl der Einträge absteigend und beschränke die Menge auf das oberste Element. Gib für alle Mitglieder dieser Gruppe die Attribute Name und Length aus und passe die Spaltenbreite automatisch an.

```
Get-ChildItem c:\windows\system32 -filter b*. * | Where-Object {$_.Length -gt 40000}
| Group-Object Extension | Sort-Object count -desc | Select-Object -first 1 |
Select-Object group | foreach {$_.group} | Select-Object name,length | Format-Table
-autosize
```

Stichwortverzeichnis

Symbole

- \$_ 83, 93, 145, 410
- & 60, 176
- > 223
- >> 223
- and 109
- as 142
- Bit 11
- cmatch 159
- cnotmatch 159
- expression 606
- force 45
- imatch 159
- inotmatch 159
- ItemsSource 1035
- Join 158
- match 159
- notmatch 159
- or 109
- Parameter 67
- Split 158
- Verbose 46
- .cat 629
- .dll 41, 133, 347, 620, 1117f.
- .exe 133, 294
- .NET 3, 35, 82, 98, 145, 167, 338,
420, 1021, 1067, 1094, 1108,
1141
- Bibliothek 335
- Klasse 335, 1146
- Runtime Host 33
- .NET API Portability Analyzer 986
- .NET Core 3, 23, 35, 335, 1004,
1014, 1141, 1144
- .NET Data Provider 681f.
- .NET Framework 33, 237, 465, 547,
772, 1141, 1143, 1147
- 4.0 11
- .NET Standard 335, 1145
- .pfx 443

- .pkg 24
- .ps1 18, 56, 123, 1118
- .psd1 513, 554, 1042, 1118, 1123
- .psm1 1042, 1113 ff., 1118, 1123
- .psproj 293
- .yaml 1016
- 32-Bit 11, 277, 687, 740
- 64-Bit 277, 687, 740
- [Type] 343

A

- Ablaufverfolgung 3, 428 f.
- About 135
- Absent 530
- AbsoluteTimerInstruction 374
- abstract 1139
- Accelerator 140
- Accent Grave
 - Gravis 47
- Access Control Entry 840
- Access Control List 840, 851
- Access Control Type 841
- Access Mask 840
- AccessControl 839, 844
- AccessMask 644
- Account Manager 569, 590
- AccountDisabled 886
- AceFlags 841
- ACL 843, 856
- ACS 1019
- Active Directory 3, 229, 360, 519,
569 f., 586, 876, 882, 886, 901,
925 f., 1043
- PowerShell 901
- Struktur 926
- Suche 891
- Active Directory Application Mode
926
- Active Directory Domain Services
924
- Active Directory Service Interface
siehe ADSI
- Active-Scripting 129
- ActiveScriptEventConsumer 375
- ActiveX Data Objects 681, 692, 872
- ADAccount 907
- ADComputer 907
- Add() 346
- Add-ADGroupMember 911, 923
- Add-Computer 746
- Add-Content 663, 679
- Add-DirectoryEntry 585
- Add-DistributionGroupMember
942
- Add-Feature 659
- Add-JobTrigger 459
- Add-LDAPObject 899, 1044 f.
- Add-LocalGroupMember 938
- Add-Member 107, 407, 410, 1052
- Add-ODBCDSN 727
- Add-PSSnapin 1071, 1076
- Add-Type 348 f., 411, 419, 631, 711
- Add-VirtualHardDisk 589
- Add-VMDisk 981
- Add-VMDrive 981
- Add-VMHardDiskDrive 959, 971
- Add-VMNIC 981
- Add-VMSwitch 959
- Add-WBSystemState 656
- Add-WindowsFeature 656, 770,
775 ff., 924
- AddCommand() 1131
- AddScript() 1128, 1131
- ADSDDeployment 924 ff.
- Administration
 - delegiert 593
 - webbasiert 298, 598
- Administrator 261
- Administratorrechte 238, 258, 261,
273 ff., 399, 558, 649, 789, 839,
1001, 1004

ADO.NET 681, 689, 872, 891
 ADODB.Connection 873
 ADPowerShell 901, 907
 ADRMS 570
 ADSI 869, 873 f., 877, 879, 939
 – .NET 867, 869, 873
 – Bindung 875 f.
 – COM 873, 879
 – Container 881
 – Pfad 875
 AdsPath 891
 ADUser 907
 Advanced Function 1039, 1048
 ADWS 902
 AgentPC 869
 Akte X 868
 Aktivierung 591
 Aktivität 472, 476
 Alias 37, 50, 227, 611
 Aliaseigenschaft 101, 106
 AliasInfo 51
 AllNodes 527
 AllowClobber 558
 AllowEmptyCollection 1048
 AllowEmptyString 1048
 AllowNull 1048
 AllSigned 130
 Alvin Kersh 868
 Änderungshistorie 686
 Animation 1035
 Ankerelement 161
 Anwendungspool 953, 955
 Anzeigesprache 512
 Apache 1004
 AppDomain 349, 71
 AppendChild(). 675
 Apple Software Package 24
 AppLockerPolicy 782 f.
 appSettings 301
 Architektur 33
 Args 127, 145
 args 247
 Array 167, 171, 173, 1150
 ArrayList 170 f.
 AsJob 448, 469
 ASP.NET 360, 1007
 Assembly 347, 547, 553, 560, 621,
 1076, 1105
 – verbreiten 1148
 AssocClass 795
 ASSOCIATORS OF 376
 Assoziation 368
 – WMI 364, 368
 Asynchronous 828
 Attribut 1133, 1138, 1149
 – indiziert 1150
 Audio 342

Aufgabe
 – geplant 455
 Aufzählung 351
 Ausdruck 58
 – regulär 159
 Ausdruckauflösung 152
 Ausdrucksmodus 58
 Ausführungsrichtlinie 129
 Ausgabe
 – mehrspaltig 208
 – unterdrücken 221
 Ausgabeobjekt 1079
 Auslagerungsdatei 591
 Authentifizierung 420, 886, 923
 AuthorizationRuleCollection 845 f.
 AutoUpdate 747
 Azure Cloud Shell 304
 Azure Container Registry 1004
 Azure Container Service *siehe* ACS

B

Background Intelligent Transfer
 Service *siehe* BITS
 BackgroundColor 146, 283, 303
 Backspace 154
 Backup 655, 657, 710
 Backup-GPO 931
 Backup-SqlDatabase 708, 710
 Base 891, 913
 bash 327
 Basisauthentifizierung 235
 Basisimage 1014, 1016
 Basisklasse 685
 Batterie 757
 Bedingung 182
 Beep 154
 Beep() 344
 Befehl
 – extern 37, 59
 Befehls-Add-On 70
 Befehls-geschichte 599
 Befehlsmodus 58
 Befehlsobjekt 690
 Begin 633
 BeginProcessing() 1070, 1074
 Benutzer 359, 896, 1134, 1137
 – Active Directory 882
 – anlegen 885
 – lokal 937
 – löschen 887
 – umbenennen 887
 – verschieben 888
 Benutzer-DSN 729
 Benutzerabmeldung 604
 Benutzeranmeldung 604
 Benutzerdaten lesen 916

Benutzergruppe 922
 Benutzerkennwort 886
 Benutzerkontensteuerung *siehe*
 UAC
 Benutzerkonto 916, 1154
 Benutzername 420
 Benutzerschnittstelle 358
 Berechnung 116
 Best Practice 570, 865
 Betriebssystembasis-Image 983
 Bezeichner 1147
 Beziehung 1137
 Bibliothek 1144
 Big Endian 891
 Bild 584
 Bildschirmschoner 413, 869
 Bindung
 – ADSI 875
 – serverlos 876
 – WMI 383
 Bing 1030
 Binärdatei 679
 Binärmodul 1113
 BIOS 359
 BitLocker
 – Überblick 659
 Bitmap 631 f.
 BITS 39, 827, 830
 Blatt 873
 BMC 358
 Boot-Konfiguration 359
 break 177, 179, 188, 192
 bxor 620, 939
 Bypass 130
 ByPropertyName 85 ff.
 Byte 148
 ByVal 85, 87
 BZIP2 641

C

C# 135, 411, 413, 1039, 1067 f., 1074,
 1141
 C++ 1145 f.
 C++/CLI 1067
 CAB 74
 Canvas 1032
 Carriage Return 154
 cat 327
 CATID 377
 CD 347
 Certificate
 – Zertifikat 857
 ChangeAccess 649
 Checkpoint-Computer 751
 Checkpoint-VM 958, 974
 Children 872

- ChildSession 252
 - CHKDSK 360
 - Chkdsk() 401
 - Chocolatey 768f.
 - Chrome 519, 768
 - CIL 33, 1143
 - CIM 6, 358, 361
 - Repository 369
 - CIM Explorer 312
 - CIM Query Language *siehe* CQL
 - CimClass 382f., 385, 394
 - CimClassProperties 394
 - CimInstance 382f., 385, 394f.
 - CimInstanceProperties 394
 - CimProperty 394
 - Cisco 358
 - City 908f.
 - class 199
 - ClassCreationEvent 498
 - ClassDeletionEvent 498
 - ClassModificationEvent 498
 - Clear-BitLockerAutoUnlock-Funktion 660
 - Clear-Content 664
 - Clear-DnsClientCache 806
 - Clear-EventLog 236, 833
 - Clear-History 600
 - Clear-Host 191, 599
 - Clear-Item 611
 - Clear-RecycleBin 630
 - Clear-Variable 144
 - ClearCase 134
 - Click 1033
 - Clipboard *siehe* Zwischenablage
 - cliXml 675
 - Close() 1032
 - CLR 17, 33, 1117, 1143
 - cmd.exe 82
 - Cmdlet 1102
 - Cmdlet Help Editor 1105
 - CmdletBinding 1048, 1052
 - cn 883
 - Codeausschnitt 278, 296
 - Codeeigenschaft 101, 106
 - Color 759
 - COM 35, 353, 630, 1146
 - Kategorie 377
 - Klasse 355
 - Komponente 359
 - Moniker 875
 - Sicherheit 373
 - Comma-Separated Values 667
 - Command Line Event Consumer 375
 - Command Mode 58
 - Commandlet 3, 37, 49, 59, 63, 83, 233
 - binär 1067
 - erstellen 1037, 1039, 1067
 - Klasse 1069
 - Konvention 1092, 1109
 - Provider 228
 - Proxy 1057
 - Verkettung 1090
 - CommandNotFoundException 197
 - CommitChanges() 342, 870, 879f., 886
 - Common Information Model *siehe* CIM
 - Common Intermediate Language *siehe* CIL
 - Common Language Runtime *siehe* CLR
 - Common Language Specification 1143
 - Common Management Information Protocol 358
 - Common Parameter 43
 - Common Type System 1143
 - compare 120
 - Compare-Object 119f.
 - Compare-VM 958, 976
 - Complete-BITSTransfer 828
 - Complete-Transaction 433f., 436
 - Component Object Model *siehe* COM
 - Compress-Archive 640
 - Computer 367, 896, 1137
 - Computergruppe 301
 - ComputerInfo 739
 - ComputerName 127, 759
 - Computername 746
 - Computerrichtlinie 430
 - Computerverwaltung 739, 789
 - configuration 520
 - ConfigurationData 527
 - ConfigurationID 537
 - ConfigurationNamingContext 905
 - Confirm 44, 46
 - confirm 45, 760, 801, 913, 916, 1052, 1054, 1102
 - ConfirmPreference 46, 1054
 - Connect-VMNetworkAdapter 959
 - Connection 688
 - Console.WriteLine() 1097, 1110
 - ConsolePaneBackgroundColor 283
 - Container 881, 931, 987, 997
 - Container-Klasse 873
 - Continue 177, 179, 192, 194
 - continue 45, 188
 - Convert-Html 679
 - Convert-String 667
 - Convert-VHD 959, 970, 973
 - Convert-Xml 677
 - ConvertFrom-CSV 328
 - ConvertFrom-String 668
 - ConvertFrom-StringData 510
 - ConvertTo-ContainerImage 1014
 - ConvertTo-CSV 667
 - ConvertTo-DataTemplate 1035
 - ConvertTo-SecureString 422, 923
 - ConvertTo-WebApplication 954
 - ConvertTo-XML 677
 - copy 623
 - Copy-ContainerFile 1011
 - Copy-GPO 930
 - Copy-Item 192, 252, 611, 623, 626f., 734, 858
 - Copy-NetFirewallRule 809
 - Copy-ToZip 641
 - Copy-VMFile 977
 - Copy/Paste 269
 - CORBA 1146
 - Count 89, 169, 188
 - Country 909
 - CPU 121
 - CQL 376, 391
 - Create() 402, 1058
 - CreateCommand() 690
 - CreateElement() 675
 - CreateInstance() 758
 - CreateObject() 356
 - CreationTime 631, 1138
 - Creators Update 26, 64
 - Credentials 923
 - Credentials 911
 - CSV 56, 417, 664ff., 737, 919, 950
 - CSV-Datei 121
 - CultureInfo 1079
 - Cursor 685f.
 - CustomerId 1056
 - CVS 134
- ## D
- Dana Scully 868
 - Data Source Name *siehe* DSN
 - DataReader 684ff., 690, 692ff.
 - DataRow 106f.
 - DataSet 684f., 692, 694ff.
 - DataTable 694f.
 - Datei 203, 360, 1134
 - Eigenschaft 619, 630
 - kopieren 623
 - löschen 39
 - Rechte 360
 - verschieben 623
 - Dateiname 37
 - Dateinamenerweiterung 121, 123

- Dateisystem 3, 625, 843, 1137, 1148
 - Dateisystemfreigabe 519, 643
 - Dateisystemkatalog 629
 - Dateisystemstruktur 624
 - Dateiversionsverlauf 654
 - Datenabfrage 377
 - Datenbank 369, 586, 681
 - Datenbankmanagementsystem 688
 - Datenbankverbindung 688
 - Datenbankzeile 106
 - Datenbankzugriff 681
 - Datenbereich 509
 - Datenbindung 1035
 - Datendatei 509
 - Datenmenge 227
 - Datenquelle 727f.
 - Datenquellensteuerelement 684
 - Datentyp 138, 145, 171, 877, 1090, 1149
 - .NET 83
 - PowerShell 138
 - WMI 363
 - Datenzugriff 689
 - DateTime 92, 95, 97, 344
 - Datum 166
 - Day 92
 - DB2 683, 727
 - dBase 727
 - DbCommand 690
 - DbDataReader 692
 - DBG 431
 - DbProviderFactories 683
 - DCOM 235, 356, 369, 385, 748, 1085, 1098
 - Konfiguration 359
 - dcpromo 924
 - DDL 376
 - Debug 44, 46
 - debug 1097
 - Debug-Modus 431
 - Debugging 3, 296, 431, 479
 - DebugPreference 46, 1097
 - Decimal 148
 - Deep Throat 868
 - Default Domain Policy 935
 - DefaultNamingContext 905
 - Deinstallation 765
 - Delete() 1138
 - Deleting 759
 - Delimiter 664
 - Deployment 1148
 - Description 883, 909
 - DESCRIPTION 1056
 - Description 1063
 - Deserialisierung 244
 - Desired State Configuration *siehe* DSC
 - Desktop 359
 - Desktop Management Task Force *siehe* DMTF
 - Desktop-Anwendungen 1144
 - Destruktor 1134, 1150
 - Deutsche Telekom 441
 - Dezimalzahl 148
 - DHCP 797f., 800
 - Diagnose *siehe* Fehlersuche
 - Dialogfenster 420, 1021
 - diff 120
 - Digest 235
 - Directory 843
 - Directory Management Objects 926
 - DirectoryEntry 106f., 141, 338f., 871ff., 877ff., 882, 885, 888
 - DirectoryInfo 98, 110, 407, 619, 843, 1138f.
 - DirectorySearcher 141, 892
 - DirectorySecurity 845
 - DirectoryString 883
 - Disable-ComputerRestore 751
 - Disable-JobTrigger 459
 - Disable-Mailbox 942
 - Disable-NetFirewallRule 809
 - Disable-PnpDevice 757
 - Disable-PSRemoting 241
 - Disable-PSSessionConfiguration 251, 253
 - Disable-VMIntegrationService 962
 - Disk Quotas 360
 - Dismount-VHD 970
 - DisplayName 909
 - Distinguished Name 875, 880, 883, 905, 909
 - Distributed COM *siehe* DCOM
 - Distributed Component Object Model *siehe* DCOM
 - Distributed File System 360
 - Distributed Managements Objects *siehe* DMO
 - DML 376
 - DMO 711
 - DMTF 235, 358
 - DNS 263, 805, 924, 986f.
 - DNS-Client 802
 - DNS-Konfigurationseinstellungen - per WMI abfragen 803
 - DNS-Server 360, 800, 804
 - DnsClient 799, 806
 - DNSClient 800, 802
 - do 177
 - Docker 983, 986, 989, 997, 999, 1001f., 1016
 - Docker Compose File 1007
 - Docker Hub 1004
 - Docker-Image 1004
 - docker.exe 997
 - Dockerfile 1011, 1016
 - DockPanel 1032
 - Dokument 663
 - Dokumentation
 - .NET 77
 - Active Directory 885
 - Dollarzeichen 116, 152
 - Domain 809
 - Domain Controller 925
 - Domäne 746, 876, 927, 1134, 1137
 - Beitritt 263, 746
 - hinzufügen 746
 - DOS 3
 - Dot Sourcing 56, 128f., 440, 547, 1041f., 1045
 - DotNetTypes.Format.ps1xml 203, 211
 - Double 148
 - DownloadString() 340, 817
 - DriveInfo 344
 - Driver 730
 - DriveType 350
 - Druckauftrag 758
 - löschen 758
 - Drucker 203, 222, 359, 759
 - verwalten 757f.
 - Druckerport 758
 - Druckerverwaltung 757f., 797
 - DSC 317, 515, 521
 - Linux 317
 - DSC Pull Server 532, 537
 - DSN 727, 729f.
 - DuplexingMode 759
 - DVD 347, 976, 980
- ## E
- E-Mail 816, 823
 - Adresse 817
 - EmailEvent 379
 - senden 815
 - echo 58
 - Edit-NanoServerImage 987
 - Eigenschaft 101
 - Eigenschaftenzwischenspeicher 870
 - Eigenschaftssatz 101, 103
 - Eingabe 417
 - Eingabeaufforderung 601f.
 - Eingabedialog 419
 - Eingabemaske 1024
 - Eingabeobjekt 1087
 - Eingabesteuerelement 1035
 - Eingabeunterstützung 277, 296
 - Einzelschrittmodus 426

Elevated 131f., 789, 839, 1004
Else 183
Emacs 271
EmailAddress 909
EmailEvent 379
Enable-BitLocker 661
Enable-ComputerRestore 751
Enable-JobTrigger 459
Enable-NetFirewallRule 809, 815
Enable-ODBCPerfCounter 727
Enable-PnpDevice 757
Enable-PSRemoting 239, 277, 759, 814
Enable-PSSessionConfiguration 251, 253
Enable-VMIntegrationService 962
Encoding 664
End 633
EndProcessing() 1070, 1074
endregion 285
Enter-PSSession 241f., 251, 256, 330, 431, 989
Enum 351
EnumerateCollection 1082
Enumerationsklasse 350
env 326, 744
Environment 517
Ereignis 362, 1133, 1150
- PowerShell 497, 506
- WMI 374, 497
Ereignisabfrage 378
Ereigniskonsument 374f.
- permanent 374
- temporär 374
Ereignisprotokoll 121, 359f., 370, 376, 429, 519, 831
- Überwachung 379, 499
Ereignisprovider 374
Ereignissystem 497
Error 145, 197, 759
ErrorAction 44ff., 194, 196, 626
ErrorActionPreference 46, 146, 196, 917
ErrorBackgroundColor 22
ErrorRecord 192, 196, 198
ErrorVariable 45, 196
Ethernet 799
ETS 83, 101, 106, 393, 874
Event *siehe* Ereignis
EventConsumer 362
EventViewerConsumer 375
EXAMPLE 1056
Example 1063
Exception 192, 197f., 1093, 1099
Exchange Management Shell 587, 941
Exchange Server 77, 360, 370, 941

ExecuteNonQuery() 690
ExecuteReader() 690, 692
ExecuteRow() 690
ExecuteScalar() 690
ExecutionPolicy 19, 21
EXIF 631
Exists() 878
exit 177
Exit-PSSession 243, 251
Expand-Archive 640
explorer.exe 655
Export-Alias 56
Export-CliXml 417, 675
Export-Console 549, 1077
Export-Counter 837
Export-CSV 110, 417, 667
Export-ModuleMember 554
Export-PfxCertificate 859
Export-VM 959, 975
Export-VMSnapshot 974
Express 704
Expression 206
Expression Mode 58
Extended Reflection 83
Extended Type System *siehe* ETS
Extensible Application Markup Language *siehe* XML
Extrinsic Event 374

F

facsimileTelephoneNumber 909
Failover Cluster 570
false 40, 45, 140, 145
Fax 909
FBI 868, 888
Feature 772
FeatureOperationResult 776
Fehler 45
Fehlerbehandlung 191, 1145
Fehlerklasse 177, 198
Fehlermeldung 40
Fehlersuche 425, 1094
Fehlertext 177
Fernaufruf 236
Fernausführung
- Hintergrundauftrag 451
Fernausführung 127, 235
Fernverwaltung 235
Fernzugriff 235
Festplatte
- virtuell 970
Festplattenverschlüsselung 659
Fibre-Channel 958
Field 102
File 843, 1136
File History 654
FileInfo 98, 110, 407, 619f., 843, 1138f.
FileInformation 785
FileSecurity 843, 845
FileSystem 584, 611
FileSystemAccessRule 846
FileSystemInfo 1138
FileSystemObject 1136
FileSystemRights 351
FileSystemWatcher 505
FileVersionInfo 769
filter 617
Find() 872
Find-Module 559f.
Find-Package 768
Firefox 519
Firewall 591
Firewall-Regel 812
First 91
For 178, 180
Force 43, 618, 913
force 45
ForEach 94, 115, 121f., 177, 180, 872, 1083
foreach 477
Foreach-Object 93f., 103, 115f., 120, 169, 181, 187, 427, 476, 663, 1082
ForegroundColor 68
Forest 927
Form Feed 154
Format 206
Format-List 81, 204, 845
Format-Table 92, 103, 121, 204, 206, 212f., 845
Format-Wide 203f., 208f.
Format-Xml 672f.
Formatkennzeichner 216
Fortschrittsanzeige 224
Fox Mulder 868
FoxPro 727
Framework Class Library 335
Freigabe 652
FullAccess 649
FullName 1138
function 37, 177, 185, 191, 227
Funktion 37, 184f., 227
- eingebaut 191
- fortgeschritten 1048

G

GAC 347f.
Ganzzahl 148
Gast 957
Gateway 800
GeneralizedTime 883

- Geplante Aufgabe 455
- Gesamtstruktur 927
- Geschäftsanwendung 1108
- Get-Acl 839, 843, 845, 855
- Get-ADComputer 911
- Get-ADDomain 927
- Get-ADDomainController 927
- Get-ADForest 927
- Get-ADGroup 911, 922
- Get-ADGroupMember 911, 923
- Get-ADObject 41, 867, 898, 911 ff.
- Get-ADOptionalFeature 927
- Get-ADOrganizationalUnit 911, 915
- Get-ADPrincipalGroupMembership 923
- Get-ADRootDSE 927
- Get-ADUser 911, 916 f.
- Get-Alias 51
- Get-AppLockerFileInformation 782
- Get-AppLockerPolicy 782 f.
- Get-AuthenticodeSignature 444
- Get-BitLockerVolume 660 f.
- Get-BITSTransfer 828 f.
- Get-BPAModel 865
- Get-BPAResult 865
- Get-CDRomDrive 585, 755
- Get-ChildItem 38 ff., 42, 81 f., 110, 114 f., 121 f., 227, 617, 640, 733, 763
- BitLocker 661
- Get-CimAssociatedInstance 384
- Get-CimClass 357, 384, 391
- Get-CimInstance 357, 379, 384 f., 387 f., 402, 756, 759
- Get-Clipboard 422 f., 563, 580
- Get-Command 49, 63 ff., 565
- Get-ComputerInfo 739, 1079
- Get-Computerinfo 746
- Get-Computername 1072
- Get-ComputerRestorePoint 751
- Get-Container 1013 f.
- Get-Containerimage 1004
- Get-ContainerImage 1006, 1014, 1018
- Get-Content 611, 626, 663, 679, 737, 762, 1036
- Get-Counter 236, 835 f.
- Get-Credential 69, 259, 420, 594, 790, 816
- Get-Culture 157
- Get-DataRow 586, 700
- Get-DataTable 586, 699
- Get-Date 95, 166 f.
- Get-DHCPServer 27, 798
- Get-DirectoryChildren 585
- Get-DirectoryEntry 152, 585, 1092
- Get-DirSize 1039
- Get-Disk 612, 614, 755, 1081 f., 1085, 1087, 1089 f., 1098
- Get-DisplaySetting 757
- Get-DnsClient 802 f.
- Get-DnsClient-Funktion - Beispiel 803
- Get-DnsClientCache 806
- Get-DnsClientServerAddress 802 f.
- Get-DomainController 27 f., 876
- Get-DSCConfiguration 531
- Get-DVDDrive 589
- Get-Event 501
- Get-EventLog 16, 121, 236, 831 f.
- Get-ExCommand 941
- Get-ExecutionPolicy 19
- Get-ExportedType 621
- Get-Filecatalog 77
- Get-FileHash 632 f.
- Get-FileVersionInfo 620, 769
- Get-FirewallRule-Funktion 810
- Get-FloppyDrive 589
- Get-Flug 1108
- Get-Flugziele 1108
- Get-Font 745
- Get-FreeDiskSpace 614, 616
- Get-GPInheritance 934
- Get-GPO 930 f.
- Get-GPOReport 933
- Get-GPPermissions 936
- Get-GPPrefRegistryValue 935
- Get-GPRegistryValue 936
- Get-GPResultantSetOfPolicy 933
- Get-GPStarterGPO 930
- Get-Help 65 ff., 72 f., 76, 135, 1039, 1056, 1064, 1104
- Get-History 599
- Get-Host 600
- Get-HotFix 236
- Get-Item 630, 733, 948, 953
- Get-ItemProperty 631, 733
- Get-Job 447, 449
- Get-JobTrigger 460
- Get-Keyboard 756
- Get-LDAPChildren 899, 1044
- Get-LDAPObject 899, 1044 f.
- Get-LocalGroupMember 938
- Get-LocalUser 937
- Get-Location 50, 611
- Get-LogicalDiskInventory 612
- Get-LoremIpsum 624
- Get-Mailbox 941 f.
- Get-Mailboxdatabase 941
- Get-Member 96, 99, 101 f., 111, 120, 344, 350, 394, 874, 1082
- Get-Members 888
- Get-MemoryDevice 585, 755
- Get-Methode 102
- Get-Module 207, 553, 563, 1113
- Get-MountPoint 27
- Get-MPComputerStatus 753
- Get-MultiTouchMaximum 757
- Get-NanoServerPackage 987
- Get-NetAdapter 799
- Get-NetAdapterBinding 799
- Get-NetFirewallAddressFilter 809
- Get-NetFirewallAddressFilter-Funktion 811
- Get-NetFirewallApplicationFilter 809
- Get-NetFirewallInterfaceFilter 809
- Get-NetFirewallInterfaceTypeFilter 809
- Get-NetFirewallPortFilter 809
- Get-NetFirewallProfile 809 f.
- Get-NetFirewallRule 809, 811 f.
- Get-NetIPInterface 800
- Get-NetworkAdapter 585, 756
- Get-ODBCDriver 727
- Get-ODBCDSN 727
- Get-OSVersion 741
- Get-Package 768 f.
- Get-PackageProvider 767
- Get-PackageSource 768
- Get-Passagier 1108
- Get-PipelineInfo 96 f., 110, 1091
- Get-PnpDevice 757
- Get-PnpDeviceProperty 757
- Get-PointingDevice 756
- Get-PowerShellDataSource 1035
- Get-Printer 759
- Get-PrintJob 759
- Get-Process 38 f., 41 f., 50, 56, 81, 86, 93, 95, 99, 101, 110 f., 116, 121, 213, 222, 236, 327, 427, 787, 1090, 1092
- Get-Processor 585, 755 f.
- Get-PSBreakpoint 432
- Get-PSDrive 612
- Get-PSProvider 229
- Get-PSRepository 561
- Get-PSSession 251
- Get-PSSessionConfiguration 251, 253, 596
- Get-PSSnapIn 550
- Get-PswaAuthorizationRule 301
- Get-Random 149
- Get-ReparsePoint 639
- Get-Service 68 f., 86, 89, 95, 110, 127, 222, 236, 793 f., 1057 f.
- Get-SHA1 633
- Get-ShortPath 622
- Get-SmbShare 644, 649
- Get-SmbShareAccess 650
- Get-SoundDevice 755

Get-SqlData 716
 Get-Storagegroup 941
 Get-Tapedrive 755
 Get-TargetResource 545
 Get-TerminalSession 27
 Get-TraceSource 428
 Get-Transaction 433, 436
 Get-Unique 113
 Get-Uptime 741
 Get-USB 756
 Get-USBController 585, 756
 Get-Variable 138, 141, 146
 Get-VHD 970
 Get-VideoController 585, 755
 Get-VirtualHardDisk 589
 Get-VM 959 f., 970
 Get-VMBIOS-VM 961
 Get-VMBuildScript 980 f.
 Get-VMHost 589, 959
 Get-VMMemory 961
 Get-VMProcessor 961
 Get-VMSnapshot 974
 Get-VMSummary 981
 Get-VMThumbnail 980 f.
 Get-WBBackupSet 657
 Get-WBPolicy 656
 Get-WBSummary 656
 Get-WebApplication 948
 Get-WebitemState 955
 Get-Website 948, 953
 Get-WebvirtualDirectory 948
 Get-WindowsEdition 741
 Get-WindowsFeature 770, 772 ff.
 Get-WinEvent 236
 Get-WmiObject 34, 236, 347, 357,
 384 f., 387 f., 390, 616, 755, 761,
 794 f., 797, 835, 869
 Get-xDscOperation 539
 GetAccessRules() 845 ff.
 GetDrives() 342
 GetFactoryClasses() 683
 GetLongDateString() 95
 GetLongTimeString() 95
 GetNames() 351
 GetObject() 356
 GetRelated() 794
 GetShortDateString() 95
 GetShortTimeString() 95
 GetTempName() 354
 Getter 102, 1149
 GetType() 96, 145, 220
 Gigabyte 148
 Git 134
 Gitternetz 1032
 GivenName 908 f.
 Gleichheitszeichen 173
 global 143, 1032

Global Assembly Cache *siehe* GAC
 Global Unique Identifier 880, 1146
 GlobalSign 441
 gm 120
 Google 1030
 GPMC 929
 Grafikkarte 359, 386
 Grant-SmbShareAccess 650
 Gravis 47 f., 90, 136, 154
 Grid 1032 f.
 GridView 209
 Group 114, 517, 873
 GROUP BY 376, 498
 Group-Object 114, 120 f., 832
 Gruppe 896, 911, 919 f.
 - Active Directory 882
 - anlegen 889
 - auflisten 888
 - lokal 937
 - Mitglied aufnehmen 889
 Gruppenmitglieder 911
 Gruppenmitgliedschaft 890
 Gruppenrichtlinie 430, 570, 604,
 929 ff., 934
 - Vererbung 934
 Gruppierung 114
 GUID 616
 Gültigkeitsbereich 139, 143
 GZIP 641

H

Haltepunkt 280, 431
 Hardlink 637
 Hardware 359, 585
 Hardwareverwaltung 755
 Hash-Tabelle 171 f., 339, 402
 Hashtable 171, 173, 339, 527, 545
 Hashwert 629
 HAVING 376, 498
 Heimatomdner 145
 Help-Info 74
 HelpMessage 1048
 Herausgeber 442, 445
 Here-String 150
 Herunterfahren 746
 Hexadezimalzahl 148
 hidden 201
 Hilfe 63
 Hilfetext 72
 Hintergrundauftrag 447
 Hintergrunddatentransfer 827
 Hintergrundübertragungsdienst
 39
 History 599
 HKCU 227
 HKEY_CURRENT_USER 734
 HKEY_LOCAL_MACHINE 734
 HKLM 227
 Home 145
 home 327
 HomeDirectory 909
 HomeDrive 909
 Host 146, 283, 303, 600, 957
 Hosting 1125
 hostname.exe 746
 Hotfix 359
 HTML 679, 952, 1024
 HtmlWebResponseObject 818
 HTTP 235, 532
 HTTPS 235
 Hyper-V 261, 569, 589, 957, 977,
 983, 986 f.
 - Überblick 957
 Hyper-V-Container *siehe* Docker
 Hyper-V-Integrationsdienste 962
 Hyper-V-Modul
 - Überblick 958
 Hypervisor 957

I

IADs 870 f., 873
 IADsComputer 873
 IADsContainer 871, 873
 IADsGroup 873
 IADsUser 873, 886
 idempotent 515 f.
 Identity 840
 IdentityReference 848
 Identität 953
 IDL 370
 IEnumerable 872
 if 177, 182
 IIS 229, 298, 876, 986 f.
 - 8.0 945
 - Internet Information Server
 945
 - Nano 994
 IIS Management Service 994
 IIS-Anwendung 954
 IISAdmin 794
 IISAdministration 945, 994
 IISpy 350
 Impersonifizierung 420, 878
 - WMI 373
 Implizites Remoting 255
 Import-Alias 56
 Import-CliXml 417, 600, 676
 Import-Counter 837
 Import-Csv 121, 619
 Import-CSV 417, 666, 919
 Import-GPO 931
 Import-INIFile 670

- Import-Module 479, 554, 560, 565, 1119f.
 - Import-PSSession 256
 - Import-VM 959, 975f.
 - IncludeUserName 788
 - Index 168
 - Indexer 1150
 - Informix 683, 727
 - Ingres 683
 - Inheritance Flags 841
 - INI-Datei 670
 - inlinescript 474
 - Innertext 675
 - Input 145
 - InputBox 348f., 419
 - Inputbox 1023
 - InputBox() 419, 1023
 - InputObject 86, 350, 1092, 1094
 - inquire 45, 194
 - Install-ADDSDomainController 925
 - Install-ADDSForest 925
 - Install-Module 305, 557, 559, 561
 - Install-PswaWebApplication 300
 - Install-WindowsFeature 300, 533
 - Installation 764
 - Installationsordner 11, 106, 145
 - Installationstechnologie 764
 - installutil.exe 1071, 1076
 - InstanceCreationEvent 498f., 501
 - InstanceDeletionEvent 362, 374, 498
 - InstanceModificationEvent 374, 498
 - Instanz 339, 1134
 - Instanziierung 1135
 - Instanzmitglied 1150
 - int 139
 - Int32 139
 - Int64 148
 - INTEGER 883
 - Integrated Scripting Environment *siehe* ISE
 - IntelliSense 29, 38, 293
 - Interface 1136
 - Interface Definition Language 370
 - InternalHost 600
 - InternalHostUserInterface 418
 - International .NET Association XXX
 - Internet Control Message Protocol 807
 - Internet Information Server 360, 945
 - Internet Information Services 359f., 532, 540, 794, 876, 945
 - Interpretermodus 271
 - IntervallTimerInstruction 374
 - Intrinsic Event 374
 - InvalidOperationException 872
 - Invoke-BPAModel 865
 - Invoke-CimMethod 384, 402
 - Invoke-Command 241, 243, 245, 247ff., 251, 258, 261, 330, 431, 448, 989
 - Invoke-ContainerImage 1005f.
 - Invoke-DbCommand 699
 - Invoke-DBMaint 713
 - Invoke-Expression 176
 - Invoke-History 599
 - Invoke-Query 716
 - Invoke-SqlBackup 723
 - Invoke-SqlCmd 702, 704, 708ff., 721, 723
 - Invoke-SqlCommand 586
 - Invoke-WebRequest 818f., 824
 - Invoke-WmiMethod 384, 401
 - InvokeMethod() 383
 - IP
 - Adresse 119, 141, 263, 377, 519, 797, 800
 - Konfiguration 377
 - Routing 360
 - IPAddress 141, 800
 - ipconfig 59, 88
 - IPHostEntry 805
 - IRQ 359
 - Is64BitOperatingSystem 740
 - Is64BitProcess 740
 - ISA 376
 - IsCoreCLR 324
 - ISE 224, 276, 281, 325, 989, 1125
 - Debugging 280
 - Integrated Scripting Environment 276
 - ISE Steroids 304
 - IsePack 584
 - IsLinux 324
 - ISO 957, 964, 967, 976
 - IsOSX 324
 - IsWindows 324
 - Item() 871
- J**
- Java 1146
 - JEA 593
 - Jeffrey Snover 125
 - Job-Trigger 459
 - zeitgesteuerte Jobs 459
 - John Doggett 868
 - Join() 158
 - Join-String 158
 - JPEG 631
 - JScript .NET 411
- K**
- Kennwort 420f., 869, 886
 - Kerberos 235, 373
 - Kill() 94
 - Kilobyte 148
 - Klammer 41
 - rund 161
 - Klammeraffe 58
 - Klasse 138, 342, 363, 367, 407, 1134, 1136f., 1149
 - .NET 199, 335, 1069, 1146, 1148
 - CIM 361
 - COM 336, 353
 - Commandlet 1070, 1089, 1111
 - PowerShell 199f.
 - statisch 344
 - WMI 359f.
 - Klassendiagramm 1138f.
 - Klassenhierarchie 1138
 - Klassenmitglied 342, 1150
 - Klassenname 338
 - Known Host 331
 - Kommandomodus 271
 - Kommandozeilenbefehl 59
 - Kommentar 136
 - Komponentenorientierung 1143f.
 - Komposition 1035
 - Komprimierung 640ff.
 - Konstruktor 1134, 1150
 - Konstruktorfunktion 337ff., 353
 - Kontakt 896
 - Konvention 1109
 - Kopieren/Einfügen 269
 - Kosinus 344
 - Kreuzzuweisung 175
- L**
- Label 206
 - LastAccessTime 1138
 - LastExitCode 145
 - Laufwerk 227f., 234, 612, 734
 - virtuell 970
 - LDAP 867, 874, 876, 891
 - Suchanfrage 871, 891
 - Suche 897
 - LDAP-Query 892
 - Leaf 881
 - Least Privilege 593
 - Leistung 835
 - Leistungsindikator 835f.

Length 89, 407
 Limit-EventLog 236, 831, 833
 LinearGradientBrush 1033
 Linie 1138
 LINK 1056
 Linux 3f., 10, 14, 23, 230, 317f.,
 320, 326, 330, 353, 998, 1002,
 1146
 – Container 1014, 1016
 – Dateisystem 327, 329
 Literal 216
 Little Endian 891
 Lizenzierung 591
 Load-ContainerImage 1017
 LoadFrom() 347
 LoadWithPartialName() 347
 Log File Event Consumer 376
 Logarithmus 344
 Logoff 604
 Logon 604
 Lokalisierung 366, 512
 Loopback 238
 Is 327

M

Machine.config 683
 MachineName 237, 1074
 MacOS 3f., 10, 14, 23, 230, 317f.,
 320, 326, 353, 1146
 – Dateisystem 329
 MailAddress 817
 MailMessage 815
 makecert.exe 442
 MAML 73, 1104
 man 327
 Manage-Bde 659
 Managed Code 870
 Managed Object 358
 Managed Object Format 370
 Managed Provider 681, 891
 Management Infrastructure API
 382f.
 ManagementBaseObject 382
 ManagementClass 106f., 141, 382f.,
 385, 388, 393, 1081
 ManagementEventWatcher 499
 ManagementObject 107, 141, 382f.,
 385, 388, 393, 395f., 401, 794,
 1081f.
 ManagementObjectCollection 393,
 1081f.
 ManagementObjectSearcher 141,
 388, 1081
 ManagementScope 499
 Mandatory 1048, 1085
 Manifest 554

Manifestmodul 1113
 Maschinencode 1143
 Match 107
 MaximumDriveCount 234
 MaximumErrorCount 145
 maxSessionsAllowedPerUser 301
 md 626, 632, 737
 measure 116
 Measure-Command 427
 Measure-Object 116, 120f.
 Measure-VM 959
 Megabyte 148
 Mehrsprachigkeit 512
 Mercurial 134
 Merge-VHD 970
 Message 192
 MessageBox 420, 1021
 MessageBoxButtons 1022
 MessageBoxDefaultButton 1022
 MessageBoxIcon 1022
 Metamodell 368
 Metaobjekt 880
 Method *siehe* Methode
 Methode 101, 1133, 1138, 1150
 – Getter 1149
 – Setter 1149
 Microsoft Access 11, 696
 – Treiber 687
 Microsoft Certified Solution
 Developer XXIX
 Microsoft Developer Network 77,
 335
 Microsoft Developer Network *siehe*
 MSDN
 Microsoft Excel 919
 Microsoft Exchange 925
 Microsoft Exchange Server 519,
 587, 941
 Microsoft Office 360
 Microsoft Outlook 792
 Microsoft Print Ticket XML 759
 Microsoft SQL Server 682, 702
 Microsoft System Center 379
 Microsoft Word 356
 Microsoft.ACE.OLEDB 687
 Microsoft.GroupPolicy 930
 Microsoft.Jet.OLEDB 687
 Microsoft.Management.Infra-
 structure.CimClass 394
 Microsoft.Management.Infra-
 structure.CimClassProperties
 394
 Microsoft.Management.Infra-
 structure.CimInstance 394
 Microsoft.Management.Infra-
 structure.CimInstanceProperties
 394

Microsoft.Management.Infra-
 structure.CimProperty 394
 Microsoft.Update 747
 Microsoft.Vhd.PowerShell 970
 Microsoft.VisualBasic 419, 1023
 Microsoft.VisualBasic.Interaction
 356, 419, 1023
 Microsoft.Win32 763
 Microsoft.Win32.RegistryKey 733
 Minute 92
 Mitglied 1133
 – .NET 1149
 – statisch 1150
 – WMI 398
 MMC 373
 Modul 553, 562
 Module Browser 558
 Modulo 173
 MOF 370, 525, 527
 Monad 5
 Monica Reyes 868
 Moniker 875
 Mono 1146
 Month 92
 more 191, 214
 Most Valuable Professional XXIX
 Mount-SpecialFolder 613
 Mount-VHD 970
 move 623
 Move-ADObject 911f.
 Move-Item 611, 623, 632
 Move-Mailbox 942
 Move-VM 959
 MSCL 33
 mscorlib.dll 1148
 MSDN 173, 335, 1028
 MSDN Library 77, 335
 MSFT_Printer 759
 MSFT_PrintJob 759
 MSFT_SmbShare 649
 MSFT_SmbShareAccessControl
 Entry 650
 MSFT_WUOperationsSession 747,
 749, 992
 MSFT_WUSettings 747
 MSFT_WUUpdate 747
 MSI 541, 761, 764f., 1071
 MTA 1027
 Multithreading 1145
 MySQL 683, 693, 702, 713, 1004
 MySqlConnection 694
 MySqlLib 702

N

Nachkommastelle 148, 344
 Name 86, 909

- Namensauflösung 805
 - Namensraum 363, 365, 367f., 843, 926, 1146
 - .NET 1146, 1148
 - ADSI 875
 - WMI 365, 368
 - Namensraumhierarchie 1147
 - Namespace-ID 875
 - NamespaceCreationEvent 498
 - NamespaceDeletionEvent 498
 - NamespaceModificationEvent 498
 - Nano Server 10, 315, 983, 986
 - IIS 994
 - Installation 987
 - Paketinstallation 987, 992
 - NanoWbem 357
 - NativeObject 872, 874
 - Navigation 227
 - Navigation Provider 228
 - Navigationsbefehl 231
 - Navigationsmodell 611
 - Navigationsparadigma 227
 - Navigationsprovider 867
 - NetAdapter 799f.
 - NetSecurity 809
 - NetSecurity-Modul
 - Überblick 808
 - Netsh 802, 804, 812f.
 - Per PowerShell aufrufen 804
 - netstat 59
 - NetTCPIP 799f.
 - Network Load Balancing 360
 - NetworkInterface 1079
 - Netzlaufwerk 359
 - Netzlaufwerksverbindung 360
 - Netzwerkadapter 957
 - Netzwerkcenter 240
 - Netzwerkkarte 359, 377, 797, 1134
 - Geschwindigkeit 801
 - Netzwerkkartenprofil 814
 - Netzwerkkonfiguration 591, 797
 - Netzwerkmanagement 357
 - Netzwerkprofil
 - per PowerShell setzen 814
 - Netzwerkverbindung 360, 519, 799f.
 - Neustart 263, 746, 776
 - New Line 154
 - new() 337, 339, 353
 - New-ADGroup 911, 922
 - New-ADObject 912
 - New-ADOrganizationalUnit 911, 915
 - New-ADUser 911, 916, 918
 - New-AppLockerPolicy 782, 785
 - New-Buchung 1108f.
 - New-Button 67, 1031, 1033
 - New-CheckBox 1035
 - New-CimInstance 384, 402
 - New-CimSession 387
 - New-CimSessionOption 387
 - New-ComboBox 1035
 - New-Container 1013
 - New-DSCChecksum 538
 - New-Ellipse 1035
 - New-Event 506
 - New-EventLog 236, 831f.
 - New-FileCatalog 629
 - New-GLink 931
 - New-GPO 930
 - New-GPStarterGPO 930
 - New-Grid 1033
 - New-Guid 337
 - New-Hardlink 638
 - New-HardwareProfile 589
 - New-IISSite 945, 994
 - New-Image 1035
 - New-Int64Animation 1035
 - New-Item 228, 611, 635, 664, 733f., 737
 - New-ItemProperty 735, 737
 - New-JobTrigger 460, 462
 - New-Junction 638, 640
 - New-Label 1028, 1033
 - New-Line 1035
 - New-ListBox 1035
 - New-LocalUser 18
 - New-Mailbox 942
 - New-Mailboxdatabase 942
 - New-MediaElement 1035
 - New-Menu 1035
 - New-Module 553
 - New-NanoServerImage 987f.
 - New-NetFirewallRule 809, 812
 - New-NetIPAddress 800
 - New-Object 336f., 344, 347, 353, 1052, 1151
 - New-PasswordBox 1033
 - New-ProgressBar 1035
 - New-PSDrive 234, 734, 763, 906
 - New-PSSession 239, 241, 251, 254f., 259, 261, 330, 597
 - New-PSSessionConfigurationFile 593
 - New-RadioButton 1035
 - New-Rectangle 1035
 - New-RichTextBox 1035
 - New-ScheduledJobOption 463
 - New-ScrollBar 1035
 - New-SelfSignedCertificate 442
 - New-Service 793
 - New-Shortcut 636
 - New-Slider 1035
 - New-SmbShare 45, 649
 - New-StatusBar 1035
 - New-Storagegroup 942
 - New-Storyboard 1035
 - New-TextBlock 1035
 - New-TextBox 1033
 - New-TimeSpan 167
 - New-TreeView 1035
 - New-UrlShortcut 637
 - New-VHD 966, 970f.
 - New-ViewBox 1035
 - New-VirtualDVDDrive 589
 - New-VirtualNetworkAdapter 589
 - New-VM 589, 959, 964, 981
 - New-VMSwitch 959
 - New-WebApplication 954
 - New-WebAppPool 953
 - New-WebServiceProxy 822ff.
 - New-Website 945, 949, 994
 - New-WebVirtualDirectory 954
 - New-Window 1033
 - New-Zip 641
 - NoAccess 649
 - Node 675
 - node.js 1004
 - NoElement 114
 - Non-Terminating Error 192, 1098, 1101
 - NoProfile 440
 - Northwind 704, 724
 - Notation 1138
 - umgekehrt polnische 891
 - NoteProperty 407, 670
 - NOTES 1056
 - Notes 1063
 - Notizeigenschaft 101, 104, 111
 - Novell 876
 - Now 343
 - Nslookup 802, 806
 - NT Event Log Event Consumer 376
 - NTAccount 844, 847
 - NtAccount 848
 - NTFS 370
 - NTLM 235
 - NTSecurityDescriptor 883
 - NuGet Package Manager 295
 - null 117, 127, 182, 185, 221
- O**
- Object 1140
 - Object Linking and Embedding Database 692
 - ObjectCategory 883, 895, 914
 - ObjectClass 883, 895, 908f., 914
 - ObjectGUID 884, 909
 - ObjectSecurity 843

- ObjectSecurityDescriptor 883
 - ObjectSid 883
 - ObjectVersion 884
 - Object[] 188
 - Objekt 668, 1110, 1133 f., 1136 f.
 - .NET 1026
 - dynamisch 407, 1039
 - WMI 360
 - Objekt-Pipeline 619
 - Objektadapter 107, 392 f., 684
 - Objektassoziation
 - WMI 368
 - Objektbaum 1137
 - Objektidentifikation
 - ADSI 875 f.
 - Objektmenge 619
 - Objektorientierung 82
 - Objekttyp 1134
 - OData 532
 - ODBC 681 f., 727, 730
 - Office 909
 - OFS 145
 - ogv 203
 - OK 1022
 - OKCancel 1022
 - OleDb 681 f., 692, 872
 - Provider 696, 872, 891
 - OleDbCommand 690, 696
 - OleDbConnection 688, 692, 696
 - OleDbDataAdapter 696
 - OMI 357, 386
 - OneGet 766
 - ONELEVEL 891
 - OneLevel 913
 - On_Click 1033
 - OO 1133, 1143 f.
 - OOP 1133
 - Open Database Connectivity
 - Einstellung 360
 - Open Management Infrastructure
 - OMI 386
 - Open Source 5
 - Open() 688
 - OpenSSH 330
 - OpenView 379
 - Operator 109, 158, 168, 172
 - Optimize-VHD 970
 - Oracle 682, 693
 - OracleCommand 690
 - OracleConnection 688
 - Ordner 39, 60, 231, 617, 622 f., 625
 - Dateisystem 360
 - Ordnerstatistik 618
 - Organisationseinheit 896, 919 f.
 - anlegen 890
 - OSS
 - Open Source 5
 - Out-Default 203, 211, 1128, 1130
 - Out-File 203, 222
 - Out-GridView 69, 203 f., 209 f.
 - Out-Host 203, 213 f.
 - Out-Null 203 f., 221
 - Out-Printer 203, 222, 758
 - Out-Speech 203, 224, 559
 - Out-SqlScript 713
 - OutBuffer 45
 - Outlook 113, 569, 590
 - OUTPUTS 1056
 - OutVariable 44, 118
 - ov *siehe* OutVariable
- P**
- PackageManagement 555 f.
 - Page File 591
 - Paketinstallation
 - Nano Server 992
 - Panel 1032
 - PaperSize 759
 - Papierkorb 630
 - parallel 476
 - PARAMETER 1056
 - Parameter 38 f., 87, 189, 1048, 1063, 1085, 1094
 - Abkürzung 42 f.
 - Skript 127
 - Parameterliste 127
 - ParentSession 252
 - parsen 667
 - Partition 911
 - PascalCasing 1147
 - PASH 317
 - PassThru 118, 911
 - Pause 759
 - PE 621
 - PercentComplete 223
 - Perforce 134
 - Performance Counter Provider 835
 - Performance Monitor 359, 370
 - PERL 135
 - Persistenz 478
 - Pester 1123
 - Petabyte 148
 - Pfad 367
 - ADSI 875
 - Verzeichnisdienst 875
 - WMI 363, 365, 367
 - Pfadangabe 231
 - Pfadname
 - Linux 327
 - Pfeilspitze 1138
 - Pflichtparameter 16
 - PHP 135
 - PhysicalDeliveryOfficeName 884, 909
 - PIN 660
 - Ping 59, 360, 807, 1098
 - Ping-Host 27 f., 807
 - Ping-VM 981
 - Pipe 82
 - Pipeline 3, 33, 81, 83, 97, 116, 221, 397
 - Ausgabe 1079
 - Eingabe 1087
 - Pipeline Processor 84, 1070
 - PipelineVariable 44, 119, 213
 - Pipelining 81, 173, 227
 - Plattform Invoke 413
 - Plattformunabhängigkeit 1143
 - Platzhalter 41, 216
 - Plug-and-Play 757
 - Polymorphismus 1140
 - Portable-Executable-Format *siehe* PE
 - PoshConsole 308
 - Position 1048, 1085
 - Postfach 942
 - Postfix-Notation 891
 - Power Management 370
 - PowerGUI 276
 - PowerShell 3, 50, 81
 - Extension 585
 - Hosting 3
 - Konsole 267
 - Laufwerk 227, 234, 734
 - Remoting 314
 - Sicherheit 129
 - Skriptsprache 135
 - Version 1.0 5
 - Version 2.0 5
 - Version 3.0 5
 - Version 4.0 5
 - Version 5.0 5
 - Version 5.1 5
 - Web Admin 598
 - PowerShell Analyzer 292
 - PowerShell Community Extensions *siehe* PSCX
 - PowerShell Core XXIII, 3 f., 23, 35, 317
 - Funktionsumfang 318
 - installieren 23
 - Konsole 324
 - Module 559
 - Version 5.1 10, 315, 317, 986
 - Version 6.x 4, 23, 317
 - WMI 357
 - PowerShell Direct 261, 263, 989

PowerShell Gallery 517, 556
 PowerShell Management Library
 for Hyper-V 958, 979
 PowerShell Remoting *siehe*
 Remoting
 PowerShell Script Analyzer 286,
 296 f.
 PowerShell Web Access *siehe*
 PSWA
 PowerShell-Remoting 298
 powershell.exe 23, 49, 567, 602,
 1027
 PowerShellGet 555 f.
 PowerShellPlus 276, 305, 1092
 PowerStudio 293
 PrimalScript 310
 Principal 844
 Print Ticket XML 759
 Printer 203
 Printing 759
 PrintManagement 758
 Private 809
 Privileg 374
 Process 86, 111, 119, 517, 633, 1092
 ProcessRecord() 1070, 1074, 1085
 Professional Developer Conference
 5
 profile 437
 profile.ps1 348, 1041, 1094
 ProfilePath 909
 Profilskript 274, 437, 440
 Programmcodeanalyse 286
 Programmgruppe 359
 Programmiersprache 182
 Programmiersprachenunabhängig-
 keit 1143
 Prompt 601
 PromptForChoice() 418
 Propagation Flags 841
 Property 102, 213, 1133, 1149
 Property Cache 879
 PropertyCollection 871
 PropertyDataCollection 383, 393 f.
 PropertyGrid 1026
 PropertyNames 871
 PropertyValueCollection 871, 878
 ProtectedFromAccidentalDeletion
 909, 913
 Protokolldatei 376
 Protokollierung 429
 Provider 230
 – ADO.NET 681
 – Dateisystem 611
 – PowerShell 228
 – Verzeichnisdienst 867
 – WMI 360
 Proxy 823

Proxy-Commandlet 255, 1057
 ProxyCommand 1058
 Prozedur 185
 Prozess 38 f., 121, 359, 369
 – auflisten 222, 787
 – beenden 791
 ps 327
 PSBase 874, 880 f.
 PSCmdlet 1069
 PSCodeGen 584
 PSComputerName 249
 PSConfig 569
 PSCredential 420 f., 790, 923
 PSCustomObject 111, 409, 665, 670,
 1052
 PSCX 28, 203, 422, 557, 559 f.,
 569, 580, 635, 641 f., 867
 psd1 511
 PSDiagnostics 570
 PSDriveInfo 612
 PSDSCRunAsCredential 520
 psedit 278, 285
 PSEdition 315
 PSHome 145
 PSHost 145, 1127
 PSHostRawUserInterface 1127 f.
 PSHostUserInterface 1127 f.
 PSImageTools 584
 psISE 283
 PSItem 82, 410
 PSModuleAutoLoadingPreference
 145, 565
 PSModulePath 554, 1114
 PSObject 106, 1130 f.
 PSReadline 270 f., 324 f.
 PSRemotingJob 448
 PSRSS 584
 PSScheduledJob 455
 PSScriptRoot 129
 PSSession 250
 PSSnapIn 1072
 PSSystemTools 584, 612, 756 f.
 psUnsupportedConsoleApplica-
 tions 281 f.
 PSUserTools 584
 PSVariable 138
 psversiontable 315
 PSWA 298, 300
 Public 809, 1085
 Public Network 240
 Pull-ContainerImage 1004
 Punktnotation 92, 340, 398
 Push-ContainerImage 1018
 Put() 398 f.
 pv *siehe* PipelineVariable
 Python 135

Q

Quantifizierer 161
 Quantor 161
 QueryDialect 391
 Quest 569, 586

R

RawUI 283
 RDP 263, 814
 Read-Host 417, 422, 471 f.
 ReadAccess 649
 Receive-Job 447, 449 f., 469
 Rechenleistung 121
 Recovery Console 983 f.
 recurse 40, 617, 911
 recursive 911, 916
 Redirection *siehe* Umleitung
 Redstone 5, 26, 64
 REFERENCES OF 376
 Referenzkopie 173 f.
 Refresh() 1138
 RefreshCache() 879
 RefreshFrequencyMins 537
 Regel 782, 812
 region 285
 Register-CimIndicationEvent 384,
 505
 Register-DnsClient 803
 Register-Event 503
 Register-ObjectEvent 653
 Register-Packagesource 560, 768
 Register-PSSessionConfiguration
 251, 254, 593
 Register-ScheduledJob 461 f.
 Register-WmiEvent 500, 505
 Registrierungsdatenbank 3, 227,
 233 f., 360, 530, 733
 – Schlüssel 733
 Registry 368, 370, 517, 733
 RegistryKey 763, 843
 RegistrySecurity 845
 RegistryValueChangeEvent 374,
 498
 Regulärer Ausdruck 140, 159, 629
 Relative Distinguished Name 880 f.
 Remote Desktop Protocol *siehe*
 RDP
 Remote Desktop Service 570
 Remote Procedure Call *siehe* RPC
 Remote Server Administration
 Tools 570, 867, 903
 Remoting 127, 235, 263, 330, 759,
 989
 – Implizit 255, 1057
 – Port 261

- Remove-ADGroup 923
- Remove-ADGroupMember 911, 923
- Remove-ADObject 911 ff.
- Remove-ADOrganizationalUnit 915
- Remove-ADUser 45, 916
- Remove-Buchung 1108
- Remove-CimInstance 384, 404
- Remove-Computer 746
- Remove-Container 1013
- Remove-ContainerImage 1014, 1018
- Remove-DirectoryEntry 585, 1092
- Remove-Event 503
- Remove-EventLog 236, 831
- Remove-GPLink 931
- Remove-GPO 930
- Remove-GPPrefRegistryValue 936
- Remove-GPRegistryValue 936
- Remove-Item 39, 43, 45, 611, 623, 734, 737
- Remove-ItemProperty 736
- Remove-Job 447, 450
- Remove-JobTrigger 460
- Remove-LDAPObject 899, 1044
- Remove-LocalUser 937
- Remove-Module 554, 568
- Remove-NetFirewallRule 809, 813
- Remove-NetFirewallRule-Funktion 813
- Remove-NetIPAddress 800
- Remove-NetRoute 800
- Remove-ODBCDsn 727
- Remove-PrintJob 759 f.
- Remove-PSBreakpoint 432
- Remove-PSSession 251 f.
- Remove-PswaAuthorizationRule 301
- Remove-SmbShare 43, 45, 649
- Remove-Variable 144
- Remove-VM 959, 969 f.
- Remove-VMSnapshot 974
- Remove-WebApplication 956
- Remove-WebAppPool 956
- Remove-Website 956
- Remove-WebVirtualDirectory 956
- Remove-WindowsFeature 770, 776
- Remove-WmiObject 384, 403
- Remove_DirectoryEntry 1102
- Rename-ADObject 911 f.
- Rename-Computer 746
- Rename-Drive 616
- Rename-GPO 930
- Rename-Item 623
- Rename-NetAdapter 802
- Rename-NetFirewallRule 809
- Rename-VM 959
- Rename-VMSnapshot 974
- Repair-VM 959
- Replace 157
- Replikation 359
- Repository 369
- requires 132
- Resize-VHD 959, 970
- Resolve-Assembly 348
- Resolve-DnsName 803, 806
- Resolve-Host 805
- Resolve-Path 232
- ResponseHeaders 340
- Ressource 517
- REST 824
- Restart-Computer 236, 746 f., 777
- Restart-PrintJob 759
- Restart-Service 46, 258, 261, 793, 795 f.
- Restart-VM 959
- Restore-ADObject 912
- Restore-Computer 751
- Restore-DscConfiguration 529
- Restore-GPO 931
- Restore-VMSnapshot 974 f.
- Restricted 130
- Restricted Runspace 593
- Resume-PrintJob 759
- Resume-Service 793, 795
- Resume-VM 959
- return 177, 233, 1070
- Revoke-SmbShareAccess 650
- Richtlinienergebnisbericht 933
- Robocopy 626 ff.
- Rolle 772, 986
- Rollendienst 772
- rootcimv2 367
- RPC 236
- RSAT 958
- RSS 584, 817 f.
- Ruby on Rails 1004
- RuleCollection 783
- Run-ContainerImage 1005 f., 1009 ff.
- RunNow 461
- Runspace 292, 593
- RuntimeException 197
- Rückgabeobjekt 1079
- S**
- sa 712
- SAM 876
- SAMAccountName 883, 886, 891, 909
- SAPI.SPVoice 224, 355
- Sapient 310, 312 f., 590 f.
- Save-ContainerImage 1017
- Save-Help 74
- Save-Module 557
- Save-VM 959
- Schablone 1134 f.
- Schalter 40, 1111
- Schattenkopie 655
- Scheduled Task 455
- ScheduledJob 461
- Schema 880, 907
 - Active Directory 885
 - WMI 368
- Schemaabfrage 377
- SchemaNameCollection 872
- SchemaNamingContext 905
- Schleife 180
- Schlüssel 227
- Schlüsselattribut
 - WMI 362
- Schnittstelle 200, 685, 1136, 1140
 - .NET 1151
- Schriftart 745
- Schtasks.exe 456
- Scope *siehe* Gültigkeitsbereich
- script 143
- Script 531
- Script Analyzer 286
- Scripting.FileSystemObject 354
- ScriptMethod 407
- ScriptPaneBackgroundColor 283
- SDDL 596, 644, 856
- sealed 1139
- Searcher 747
- SearchScope 913
- Secure Shell *siehe* SSH
- Secure String 659 f.
- Security Descriptor 840
- Security Descriptor Definition Language 253
- Security Identifier 840, 845, 847 f.
- Security Service Provider 373
- Select
 - PowerShell 111
- SELECT 376, 498
 - WQL 376 f., 379
- Select-Object 16, 81 f., 91, 103, 107, 111, 115, 120 f., 213, 396, 674, 1094
- Select-String 59, 88, 629, 664
- Select-Xml 673 f.
- SelectNodes() 673 f.
- SelectSingleNode() 673 f.
- Semaphore 843
- Semikolon 121, 789
- Send-MailMessage 815 f.
- Send-SmtpMail 815 f.
- sequence 473

- Serialisierung 98, 244
- Seriennummer 742
- Server 911
- Server Management Objects *siehe* SMO
- Server Manager 924
- ServerCertificateValidationCallback 824
- ServerRemoteHost 303
- ServerURL 537
- Service 517
- ServiceController 244, 793
- Serviceorientierung *siehe* SOA
- Session 356, 747
- sessionState 301
- Set-Acl 839, 852, 855
- Set-ADAccountPassword 911
- Set-ADGroup 922
- Set-ADObject 912 f.
- Set-ADOrganizationalUnit 915
- Set-ADUser 916, 918
- Set-Alias 55
- Set-AppLockerPolicy 782
- Set-AuthenticodeSignature 443
- Set-BPAResult 865
- Set-CimInstance 384, 400
- Set-Clipboard 422 f.
- Set-Content 611, 663, 679, 817
- Set-DataRow 700
- Set-DataTable 586, 700
- Set-Date 167
- Set-DistributionGroup 942
- Set-DnsClientServerAddress 800, 802 f.
- Set-ExecutionPolicy 19, 21, 123, 129 f., 444
- Set-FileTime 620
- Set-FirewallProfile 810
- Set-GPIInheritance 935
- Set-GLink 931
- Set-GPPermissions 936
- Set-GPPrefRegistryValue 935
- Set-GPRegistryValue 935
- Set-Item 258, 611
- Set-ItemProperty 620, 736, 815
- Set-JobTrigger 460
- Set-Location 50, 227, 611, 733
- Set-Mailbox 942
- Set-Methode 102
- Set-NetFirewallPortFilter 809
- Set-NetFirewallProfile 809
- Set-NetFirewallRule 809, 812
- Set-NetIPInterface 800
- Set-ODBCDriver 727
- Set-ODBCdsn 727
- Set-PrintConfiguration 759
- Set-PSBreakpoint 431 f.
- Set-PSDebug 141, 425 f.
- Set-PSReadlineOption 22, 271, 325
- Set-PSSessionConfiguration 251
- Set-ScheduledJob 461
- Set-Service 793, 796
- Set-StrictMode 141
- Set-TargetResource 544
- Set-TraceSource 429
- Set-Variable 138, 146, 1031 f.
- Set-VHD 970
- Set-VM 959, 961
- Set-VMMemory 981
- Set-VolumeLabel 616
- Set-WmiInstance 384, 400
- Set-WSManQuickConfig 240
- SetInfo() 870, 879 f.
- Setter 102, 1149
- SHA256 629
- Shell 3, 81
- Shell.Application 630, 642
- Shielded VM 987
- ShouldProcess() 1102 f.
- Show() 1021
- Show-Command 69 f., 279
- Show-EventLog 236, 832
- Show-HyperVMMenu 980
- Show-NetFirewallRule 809
- Show-Service 236
- Show-VMMenu 980
- ShowDialog() 1036
- Shutdown 604
- Sicherheit
 - COM 372
 - Dateisystem 360, 370
 - PowerShell 129
 - WMI 372
- Sicherheitsabfrage 1052, 1102
- Sicherheitsbeschreibung 840
- Sicherheitseinstellung 839
- Sicherheitsmodell 3
- Sicherheitsrichtlinie 131
- SID 840
- Side-by-Side Executing 1145
- Signatur
 - digital 441
- Signieren 442
- SilentlyContinue 45, 194, 626, 917
- Simple Network Management 358 f., 370
- Simple Object Access Protocol *siehe* SOAP
- Sitzung 250 ff.
- Sitzungskonfiguration 253, 593, 597
- Skip 91
- SkipNetworkProfileCheck 241, 814
- Skript 123, 125
 - PowerShell 123
- Skriptausführungsrechte 19
- Skriptausführungsrichtlinie 20
- Skriptblock 143, 243, 1031
- Skriptdatei 123
- Skripteigenschaft 101, 105
- Skriptmodul 1113
- Skriptsprache 1037, 1039
- SMB 986
- SMO 704, 710 f., 721 f., 724
- Smoking Man 868
- SMTP 815 f.
- SmtplibClient 815
- SNA Server 370
- Snap-In 547, 553, 567, 1071 f., 1094, 1105
- Snapshot
 - Hyper-V 974
- Snippet 278
- SOA 1143 f.
- SOAP 235, 369, 822
- Software 359, 591
 - deinstallieren 765
 - installieren 541, 543, 764
 - inventarisieren 761
 - verwalten 761
- Software Restriction Policy 781
- Softwareentwickler 335
- Softwareentwicklungsplattform 1144
- Softwarekomponente 347
- Softwarepaket 768
- Softwarequelle 768
- Sort-Object 81 ff., 107, 112, 114 f., 120 ff., 188, 1070
- Sortieren 112
- Speech 203
- SpeechSynthesizer 224
- Speicher 89
- Speicherbereinigung 1145
- Speicherverbrauch 686
- Speicherverwaltung 1145
- Spitzname 1136
- Spoolerdienst 759
- Spooling 759
- Sprachausgabe 203, 224, 355
- Sprache 512
- Sprachkürzel 512
- SQL 376, 730
- SQL Server 703
 - Agent 721
 - Laufwerk 705
- SQL Server Management Studio 705, 721
- SQLASCOMMANDLETS 703

- Sqlcmd.exe 709
- SqlCommand 690, 709
- SqlConnection 339, 688, 692, 694
- SqlDataSourceEnumerator 684
- SQLPS 229, 702, 704, 708
- SQLPSX 702, 704, 713 f., 721
- SqlServerCe 682
- SqlServerCmdletSnapin100 703
- SSH 330
- sshs 330
- SSHTransport 330
- SSL 300, 824
- STA 1027
- StackPanel 1032
- StackTrace 145
- Stammzertifizierungsstelle 442, 445
- Standarddrucker 222
- Standardkonsole 267
- Start-BITSTransfer 828
- Start-Container 1013
- Start-ContainerProcess 1009 f.
- Start-DscConfiguration 523
- Start-Job 447 ff.
- Start-Process 120, 455, 787 f., 790
- Start-PSSession 451
- Start-Service 245, 793, 795
- Start-Sleep 132
- Start-Transaction 433 ff.
- Start-Transcript 471
- Start-VM 959, 968, 981
- Start-WBBackup 656
- Start-WBFileRecovery 657
- Start-WBHyperVRecovery 657
- Start-WBSystemStateRecovery 657
- Start-WBVolumeRecovery 657
- Start-Webitem 955
- Start-Website 955
- Startmenü 359
- Startup 604
- static 200
- Status 223, 759
- Stop 45, 194
- Stop-Computer 236, 746
- Stop-Container 1013
- Stop-Job 447, 450
- Stop-Process 94, 120, 471, 787, 791, 1092
- Stop-Service 39, 793, 795
- Stop-VM 959
- Stop-WBJob 656
- Stop-Webitem 955
- Stop-Website 955
- StopProcessing() 1070
- Stored Procedure 695
- Streaming 84
- StreetAddress 909
- String 150, 157
- Subnetzmaske 800
- SUBTREE 891
- SubTree 913
- Subversion 134
- Suche
 - Active Directory 891
 - Assembly 621
 - LDAP 872
 - Verzeichniseintrag 881
 - XML 673
- SupportsShouldProcess 1102
- Surname 909
- Suse 10
- Suspend 44
- Suspend-PrintJob 759
- Suspend-Service 793, 795
- Suspend-VM 959
- Switch 40, 177, 182, 957
- SwitchParameter 1111
- Sybase 683
- Symbolic Link 637, 639
- SymLink 639 f.
- SYNOPSIS 1056
- Synopsis 1063
- Syntaxfarbhervorhebung 296
- System 1147 f.
- System ACL 844
- System Center Virtual Machine Manager 588
- System Management Server 370
- System-DSN 729
- System.ApplicationException 197
- System.Boolean 231
- System.Collections.Hashtable 171
- System.Console 345, 599
- System.Data 337
- System.Data.Odbc 682, 730
- System.Data.OleDb 682, 689
- System.Data.OLEDB 682, 689
- System.Data.OracleClient 682, 689
- System.Data.SqlClient 682, 689, 709
- System.Data.SqlClient.
SqlConnection 339
- System.Data.SqlServerCe 682
- System.DateTime 166, 339 f., 343
- System.Diagnostics.EventLog 831
- System.Diagnostics.Process 83, 94, 98, 211, 787, 1090
- System.DirectoryServices 337, 867 ff., 871, 873 f., 877, 882, 885, 891, 926, 939
- System.DirectoryServices.
ActiveDirectory 926
- System.Directoryservices.
DirectoryEntry 339, 341
- System.dll 1148
- System.Enum 351
- System.Environment 242, 740 ff., 745, 839, 1074 f., 1079
- System.Globalization.CultureInfo 601
- System.Int32 139, 148
- System.IO.Compression 642
- System.IO.Directory 843
- System.IO.DirectoryInfo 407, 617 f.
- System.IO.DriveInfo 342, 345, 347, 350, 612, 614
- System.IO.DriveType 350
- System.IO.File 843
- System.IO.FileInfo 407, 617 f., 630
- System.Management 337, 1081
- System.Management.Automation 77, 1069, 1072, 1081, 1130
- System.Management.Automation.
Cmdlet 1070
- System.Management.Automation.
PathInfo 232 f.
- System.Management.Automation.
PSCustomObject 665
- System.Management.Automation.
PSDriveInfo 612
- System.Management.Manage-
mentObject 167
- System.Math 344
- System.Media.SoundPlayer 342
- System.Net.Mail 815, 817
- System.Net.WebClient 340, 817, 824
- System.Object 96, 98, 409, 948, 1090, 1094
- System.Random 150, 339
- System.Reflection 347
- System.Security 844
- System.Security.AccessControl 843
- System.ServiceProcess.Service-
Controller 98, 793 f., 1090
- System.String 150, 155, 328, 1074
- System.TimeSpan 166
- System.Type 96, 145, 220
- System.ValueType 173
- System.Windows 1027
- System.Windows.FontStyle 1030
- System.Windows.Forms 347, 631, 1021
- System.Xml.Node 675
- System32 115, 121 f.
- Systemattribut
 - WMI 363
- Systemdienst 85, 359, 793
 - auflisten 377
 - überwachen 379

Systemende 604
 SystemEvent 498
 Systemklassen
 – WMI 361
 Systemmanagement 357
 SystemParametersInfo 413
 Systemstart 604
 Systemwiederherstellung 751
 Sysvol 604

T

T-SQL 709
 Tab Completion 270
 Tabellenformatierung 206
 TabPanel 1032
 Tabulator 154
 Tabulatorvervollständigung 269
 Tag-ContainerImage 1017
 TAR 641
 TaskScheduler 584
 TCP/IP 800
 Team Foundation Server *siehe* TFS
 Tee-Object 117f.
 Telnet 242
 Terminal Services 360
 Terminating Error 192, 1098
 Terrabyte 148
 Test-32Bit 756
 Test-64Bit 756
 Test-AppLockerPolicy 782, 784
 Test-Assembly 620
 Test-Connection 28, 87f., 807f.
 Test-CustomerID 1054
 Test-DbConnection 699
 Test-DscConfiguration 523
 Test-FileCatalog 76, 629
 Test-IsAdmin 839
 Test-ModuleManifest 555
 Test-Path 231
 Test-PswaAuthorizationRule 301
 Test-ServiceHealth 941
 Test-SqlScript 713
 Test-TargetResource 544
 Test-UserGroupMembership 890
 Test-VHD 959, 970
 Test-Xml 672
 Textanzeige 1035
 Textdatei 121, 663
 Texteingabefeld 419, 1023
 TextInfo 157
 TFS 134
 Thawte 441
 this 410, 1032
 Thread 476
 Thread-Modell 1027
 ThrottleLimit 249

throw 177, 197
 ThrowTerminatingError() 1098
 Thumbprint 857
 TIFF 631
 TimeSpan 427
 Tivoli 379
 TLS 824
 ToLower() 157
 Ton 344
 ToString() 96, 98, 346, 1090, 1140
 TotalProcessorTime 220
 ToTitleCase() 157
 ToUpper() 157
 TPM 659
 Trace-Command 606
 Tracing *siehe* Ablaufverfolgung
 Transaktion 433
 Transformation 1035
 Translate() 848
 trap 177, 191f., 197f.
 Treiber 591
 – ODBC 728
 Trigger 459
 Troubleshooting Pack 571, 861
 true 140, 145, 756
 Trusted Host 258
 Trusted Platform Module *siehe*
 TPM
 Trustee 840
 Try-Catch-Finally 191, 198
 Try...Catch 918
 Tuva 983
 Typ 139, 335, 1134, 1148
 – Namensgebung 1148
 Typadapter 139f., 169
 Typbezeichner 139
 Type Cast *siehe* Typkonvertierung
 types.ps1xml 56 f., 106 f.
 Typisierung 138
 Typkennzeichner *siehe* Typbe-
 zeichner
 Typkonvertierung 112, 142
 Typname *siehe* Typbezeichner

U

UAC 20, 126, 131, 273, 789
 Überladung 190
 Ubuntu 10, 23
 Umgebungsvariable 359
 – Linux 326
 Umlaut 664
 Umleitung 223
 Undefined 130
 Undo-Transaction 433, 435 f.
 UniformGrid 1032
 Uninstall-Package 769

Universal Coordinated Time 364,
 400
 Unix 3, 81f., 135, 611, 637
 Unlock-BitLocker 661f.
 Unregister-PSSessionConfiguration
 251, 255, 596
 Unrestricted 130
 Unterbrechungsfreie Stromver-
 sorgung *siehe* USV
 Unternamensraum 1146
 Unterordner 98, 617
 Unterroutine 184
 Unterschlüssel 227
 until 177
 Update 359, 591
 – Einstellungen 750
 – installieren 749
 – suchen 747
 Update-Help 74
 UpdateColl 747
 UsePropertyCache 879
 User 517, 873, 882, 895
 User Account Control *siehe*
 Benutzerkontensteuerung
 user32.dll 413
 UserDomainName 839
 UserName 839
 UseTestCertificate 300
 UseTransaction 434
 using 474
 USV 757
 UTF8 664

V

Validate-CustomerID 1054
 ValidateCount 1048
 ValidateLength 147, 1049, 1111
 ValidateNotNull 1048, 1111
 ValidatePattern 147, 1049, 1111
 ValidateRange 147, 1049
 ValidateScript 147, 1049
 ValidateSet 147
 ValueFromPipeline 1048, 1087,
 1089, 1094
 ValueFromPipelineByPropertyName
 1048, 1089
 ValuesCollection 871
 ValueType 173
 Variable 96, 118, 137, 145, 216,
 227
 – Auflösung 151
 – eingebaut 145, 324
 – vordefiniert 145, 324
 – Workflow 475
 Variablenauflösung 151f., 216
 Variablenkennzeichner 118, 137

- Variablentypisierung 138
 - VB 411, 413
 - Verbindungszeichenfolge 339, 688, 695
 - Verbose 44, 46, 523, 1097
 - VerbosePreference 46, 146, 1097
 - VerbsCommon 1110
 - VerbsCommunications 1110
 - VerbsData 1110
 - VerbsDiagnostic 1110
 - VerbsLifeCycle 1110
 - VerbsSecurity 1110
 - Vererbung 368, 1138, 1150
 - Vererbungsdiagramm 1138
 - Vererbungshierarchie 381, 907, 1138
 - WMI 368
 - Vergleich 119
 - Vergleichsoperator 107
 - Verifikation 1145
 - VeriSign 441
 - Verknüpfung 636
 - Verzeichnisattribut 878
 - Verzeichnisdienst 106, 370, 585, 880, 892
 - Verzeichnisdienstklasse 880
 - Verzeichnisobjekt 877, 882
 - Verzweigung 117
 - VHD 969 f., 980, 987
 - VHDX 966, 970, 980
 - Video 1035
 - View 211
 - VirtualHardDisk 970
 - Virtualisierung 957
 - VirtualizingStackPanel 1032
 - Virtuelle Maschine *siehe* VM
 - Virtuelles System 957
 - Virus 131
 - Visual Basic 411, 1141
 - Visual Basic .NET 1067 f.
 - Visual Basic 6.0 1145
 - Visual Studio 293, 295, 479, 1007, 1014, 1016, 1037, 1067 f., 1094
 - Container 1007
 - Visual Studio Code 295, 325, 1011
 - Visual Studio Team Services *siehe* VSTS
 - Visual Web Developer Express 1068
 - VM 589, 957, 964
 - VMbus 261
 - VMGUID 261
 - VMName 261
 - void 221
 - Volume Shadow Copy Service
 - siehe* VSS
 - VolumeLabel 341
 - VSCode
 - Visual Studio Code 295
 - VSCode-PowerShell 295, 325
 - VSS 654
 - VSTS 134
- W**
- Wait 523
 - Wait-Job 447, 450
 - Wait-Process 792
 - WaitForAll 545
 - WaitForAny 545
 - WaitForSome 545
 - Walter Skinner 868
 - WarningAction 44 f., 194
 - WarningVariable 45
 - Warnung 45
 - WAS 794
 - WBEM 6, 358
 - WDAC 727
 - Web Administration 571
 - Web Based Enterprise Management 358
 - Web Service Description Language 823
 - WebAdministration 945, 947, 994
 - Webanwendung 1144
 - Webdienst 822
 - Weblog 817, 1157
 - Webserver 229, 950
 - Webservices 822, 824
 - Web Services Description Language *siehe* WSDL
 - Website 819, 950, 956
 - Well-Known GUID 876
 - Well-Known Object 876
 - Well-Known Security Principal 849
 - WellKnownSidType 849
 - Werkzeug 267
 - Wertemenge 167
 - Wertkopie 173 f.
 - whatif 44 f., 146, 623, 760, 1052, 1102
 - WhatIfPreference 46
 - WHERE 376
 - Where-Object 50 f., 81 f., 94 f., 107, 109, 116, 120 f., 222, 793, 1070, 1082, 1094
 - while 177
 - Whistler 366
 - whoami.exe 274
 - Width 206
 - Wiederherstellungspunkt 751
 - Win32 361
 - Win32 OpenSSH 330
 - Win32-API 413
 - Win32_Account 869
 - Win32_ACE 646
 - Win32_Battery 757
 - Win32_Bios 742
 - Win32_BootConfiguration 742
 - Win32_CDROMDrive 397
 - Win32_CDROMdrive 755
 - Win32_CodecFile 763
 - Win32_ComponentCategory 377
 - Win32_ComputerShutdownEvent 374, 498
 - Win32_Computersystem 740
 - Win32_Currenttime 167
 - Win32_Desktop 869
 - Win32_Diskdrive 755
 - Win32_Group 869
 - Win32_Keyboard 756
 - Win32_LocalTime 167
 - Win32_LogicalDisk 367 f., 377, 401, 612, 614 f., 1081
 - Win32_MappedLogicalDisk 616
 - Win32_MemoryDevice 755
 - Win32_NetworkAdapter 756
 - Win32_NetworkAdapterConfiguration 377, 797, 803 f.
 - Win32_NTLogEvent 377, 379, 499
 - Win32_OperatingSystem 740 f.
 - Win32_OSRecoveryConfiguration 742
 - Win32_PerfRawData 835
 - Win32_PerfRawData_PerfOS_Processor 835
 - Win32_PerfRawData_PerfProc_Process 835
 - Win32_PingStatus 807
 - Win32_PointingDevice 756
 - Win32_PowerManagementEvent 498
 - Win32_Printer 757, 759, 797
 - Win32_Printjob 757 f.
 - Win32_Process 499
 - Win32_Processor 755 f.
 - Win32_ProcessStartTrace 498
 - Win32_Product 761, 764 f.
 - Win32_Quickfixengineering 763
 - Win32_SecurityDescriptor 646
 - Win32_Service 377, 379, 499, 793
 - Win32_Share 644 f.
 - Win32_SoundDevice 755
 - Win32_SystemConfiguration-ChangeEvent 374, 498
 - Win32_Tapedrive 755
 - Win32_TCPIPPrinterPort 757 f., 797
 - Win32_Trustee 646
 - Win32_USBController 756
 - Win32_UserAccount 119, 367, 869

- Win32_VideoController 386, 397, 755, 757
 - Win32_Volume 616
 - Win32_WindowsProductActivation 742
 - window 1032
 - Windows 10 5, 10, 261, 267, 270, 576
 - Anniversary Update 5
 - Rolle 772, 986
 - Windows 2000 366
 - Windows 7 570
 - Windows 8 391, 945
 - Windows 8.1 573
 - Windows 9x 369
 - Windows Activation Service *siehe* WAS
 - Windows as a Service 5
 - Windows Communication Foundation 360
 - Windows Container *siehe* Docker
 - Windows Data Access Components *siehe* WDAC
 - Windows Defender 753, 987
 - Windows Driver Model 370
 - Windows Explorer 655, 736
 - Windows Firewall 591, 808, 813
 - per PowerShell konfigurieren 808
 - Windows Forms 294, 1021, 1024, 1026
 - Windows Installer 370, 781
 - Windows Management Framework 11, 358, 759
 - Windows Management Instrumentation 33
 - Windows ME 369
 - Windows Nano Server 10, 983, 999, 1004
 - Windows PowerShell XXIII, 3f.
 - Windows PowerShell Community Extensions 580
 - Windows Pre Installation Environment *siehe* WinPE
 - Windows Presentation Foundation *siehe* WPF
 - Windows Remote Management *siehe* WinRM
 - Windows Script Host 34, 129
 - Windows Server 2003 4, 366, 369, 891
 - Windows Server 2012 391, 571, 924, 945
 - Windows Server 2012 R2 304, 573
 - Windows Server 2016 5, 10, 261, 267, 270, 576
 - Windows Server-Container *siehe* Docker
 - Windows Server Core 276, 999, 1004
 - Windows Troubleshooting Platform 861
 - Windows Update 747, 750
 - Agent API 992
 - Nano Server 992
 - Windows Vista 1141
 - Windows XP 33, 366, 375
 - Windows-Authentifizierung 712
 - Windows-Firewall
 - im Netzwerk abfragen 814
 - WinMgmt.exe 369f.
 - WinPE 659
 - WinRM 235, 237, 239f., 369, 448, 747
 - WITHIN 376, 498
 - WKGUID 876
 - WMI 3, 6, 33, 167, 235, 357, 360, 368, 388, 1081, 1085, 1098
 - Class Explorer 381
 - Command Shell 33
 - Data Query 377
 - Ereignis 374
 - Event Query 376, 378
 - Klasse 381
 - Namespace 365
 - Object Browser 380f.
 - Query Language 376, 390
 - Repository 369, 374, 392
 - Schema 368, 377
 - Schema-Query 377
 - Steuerung 369
 - WMI API 382
 - WMI Object Browser 380
 - WMI Query Language *siehe* WQL
 - WMIClass 357, 388
 - WMISEARCHER 357, 388, 390
 - Word 113
 - Workflow 465, 471ff., 479
 - Designer 480
 - Einschränkungen 470
 - Persistenz 478
 - verschachtelt 475
 - WorkflowInfo 480
 - WorkingSet 106
 - WorkingSet64 83
 - World Wide Wings 1108
 - Wörterbuch 113
 - WPF 276, 294, 349, 469, 1021, 1027
 - WPF PowerShell Kit 584, 1027
 - WPK 584, 1027
 - WQL 376, 391, 497, 1081
 - WrapPanel 1032
 - Write-BZip2 641
 - Write-Clipboard 423
 - Write-Error 215
 - Write-EventLog 236, 831, 833
 - Write-GZip 27, 641
 - Write-Host 68, 203, 215, 303, 472
 - Write-Output 58
 - Write-Progress 223f., 406
 - Write-Tar 641
 - Write-Warn 215
 - Write-Zip 641
 - WriteDebug() 1098
 - WriteError() 1098, 1101
 - WriteObject() 1070, 1075, 1082f.
 - WriteVerbose() 1098, 1101
 - WriteWarning() 1098, 1101
 - WS-Management 235f., 239, 259, 369, 382, 385, 387
 - WScript.Shell 636
 - WSDL 822
 - WSH 874
 - WSMan 229, 260
 - Wurzelnamensraum 1146
 - www.IT-Visions.de 569, 585, 598, 612, 699
- ## X
- X-Files 868
 - x64 957
 - x86 957
 - XAML 469, 479, 1035
 - XamlReader 1036
 - XCopy-Deployment 1145, 1148
 - xDscDiagnostics 539
 - xDscWebService 534
 - XFilesServer 868
 - XML 73, 417, 549, 652, 671f., 675, 677, 824, 1104
 - XML Application Markup Language *siehe* XAML
 - XML-Schema 672
 - XML-Webservice 1146
 - XmlAttribute 674
 - XmlDocument 677
 - XmlElement 674
 - XPathDocumentNavigator 674
 - XslCompiledTransform 677
- ## Y
- YAML 1007
 - Year 92
 - YesNo 1022
 - YesNoCancel 1022

Z

- Zahl 148
- Zahlenliteral 148
- Zeichenkette 150 f., 158, 216, 1091, 1110
 - ersetzen 157
 - Operation 156
 - trennen 158
 - verbinden 158
- Zeichensatz 664
- Zeilenumbruch 47, 90
 - Pipeline 90
- Zeitmessung 427
- Zeitplandienst 359
- Zertifikat 300, 442, 857
 - selbst signiert 442
- Zertifikatsdatei 443
- Zertifikatspeicher 3, 227, 857
- Zertifikatsverwaltung 441 f., 445
- ZIP 640 f.
- ZipFile 642
- Zufallszahl 149 f.
- Zugriff verweigert 400
- Zugriffsrechteliste 839, 845
- Zuweisungsoperator 173
- Zwischenablage 422
- Zwischencode 1143
- Zwischenschritt 116
- Zwischenspeicher 686