

HANSER



Leseprobe

zu

„Windows PowerShell 5.1 und PowerShell Core 6.1“

von Holger Schwichtenberg

ISBN (E-Book): 978-3-446-45923-6

Weitere Informationen und Bestellungen unter
<http://www.hanser-fachbuch.de/978-3-446-45923-6>

sowie im Buchhandel

© Carl Hanser Verlag, München

Inhalt

Vorwort	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 Windows PowerShell herunterladen und auf anderen Windows- Betriebssystemen installieren	4
1.4 Die Windows PowerShell testen	8
1.5 PowerShell Core installieren und testen	18
1.6 Woher kommen die Commandlets?	24
1.7 PowerShell Community Extensions (PSCX) herunterladen und installieren ..	25
1.8 Den Windows PowerShell-Editor „ISE“ verwenden	26
2 Fakten zur PowerShell	31
2.1 Geschichte der PowerShell	31
2.2 Motivation zur PowerShell	33
2.3 Betriebssysteme mit vorinstallierter PowerShell	36
2.4 Einflussfaktoren auf die Entwicklung der PowerShell	37
2.5 Anbindung an Klassenbibliotheken	39
2.6 PowerShell versus WSH	39
3 Einzelbefehle der PowerShell	43
3.1 Commandlets	43
3.2 Aliase	56
3.3 Ausdrücke	64
3.4 Externe Befehle	65
3.5 Dateinamen	66

4	Hilfefunktionen	69
4.1	Auflisten der verfügbaren Befehle	69
4.2	Volltextsuche	71
4.3	Erläuterungen zu den Befehlen	72
4.4	Hilfe zu Parametern	73
4.5	Hilfe mit Show-Command	75
4.6	Hilfefenster	76
4.7	Allgemeine Hilfetexte	78
4.8	Aktualisieren der Hilfsdateien	79
4.9	Online-Hilfe	81
4.10	Fehlende Hilfetexte	82
4.11	Dokumentation der .NET-Klassen	83
5	Objektorientiertes Pipelining	87
5.1	Pipeline-Operator	87
5.2	.NET-Objekte in der Pipeline	88
5.3	Pipeline Processor	90
5.4	Pipelining von Parametern	91
5.5	Pipelining von klassischen Befehlen	94
5.6	Anzahl der Objekte in der Pipeline	95
5.7	Zeilenumbrüche in Pipelines	96
5.8	Zugriff auf einzelne Objekte aus einer Menge	96
5.9	Zugriff auf einzelne Werte in einem Objekt	98
5.10	Methoden ausführen	99
5.11	Analyse des Pipeline-Inhalts	101
5.12	Filtern	113
5.13	Zusammenfassung von Pipeline-Inhalten	116
5.14	„Kastrierung“ von Objekten in der Pipeline	117
5.15	Sortieren	118
5.16	Duplikate entfernen	119
5.17	Gruppierung	120
5.18	Berechnungen	122
5.19	Zwischenschritte in der Pipeline mit Variablen	122
5.20	Verzweigungen in der Pipeline	123
5.21	Vergleiche zwischen Objekten	125
5.22	Zusammenfassung	126
5.23	Praxisbeispiele	127
6	PowerShell-Skripte	129
6.1	Skriptdateien	129

6.2	Start eines Skripts	131
6.3	Aliase für Skripte verwenden	132
6.4	Parameter für Skripte	133
6.5	Skripte dauerhaft einbinden (Dot Sourcing)	134
6.6	Das aktuelle Skriptverzeichnis	135
6.7	Sicherheitsfunktionen für PowerShell-Skripte	135
6.8	Anforderungsdefinitionen von Skripten	138
6.9	Skripte anhalten	138
6.10	Versionierung und Versionsverwaltung von Skripten	139
7	PowerShell-Skriptsprache	141
7.1	Hilfe zur PowerShell-Skriptsprache	141
7.2	Befehlstrennung	142
7.3	Kommentare	142
7.4	Variablen	143
7.5	Variablenbedingungen	153
7.6	Zahlen	154
7.7	Zeichenketten (Strings)	156
7.8	Reguläre Ausdrücke	165
7.9	Datum und Uhrzeit	171
7.10	Arrays	173
7.11	ArrayList	176
7.12	Assoziative Arrays (Hash-Tabellen)	177
7.13	Operatoren	178
7.14	Überblick über die Kontrollkonstrukte	182
7.15	Schleifen	183
7.16	Bedingungen	188
7.17	Unterroutinen (Prozedur/Funktionen)	190
7.18	Eingebaute Funktionen	196
7.19	Fehlerbehandlung	197
7.20	Objektorientiertes Programmieren mit Klassen	204
8	Ausgaben	209
8.1	Ausgabe-Commandlets	209
8.2	Benutzerdefinierte Tabellenformatierung	212
8.3	Benutzerdefinierte Listenausgabe	214
8.4	Mehrspaltige Ausgabe	214
8.5	Out-GridView	215
8.6	Standardausgabe	217
8.7	Einschränkung der Ausgabe	219

8.8	Seitenweise Ausgabe	219
8.9	Ausgabe einzelner Werte	220
8.10	Details zum Ausgabeoperator	222
8.11	Ausgabe von Methodenergebnissen und Unterobjekten in Pipelines	226
8.12	Ausgabe von Methodenergebnissen und Unterobjekten in Zeichenketten	226
8.13	Unterdrückung der Ausgabe	227
8.14	Ausgaben an Drucker	228
8.15	Ausgaben in Dateien	228
8.16	Umleitungen (Redirection)	229
8.17	Fortschrittsanzeige	229
8.18	Sprachausgabe	230
9	Das PowerShell-Navigationsmodell (PowerShell Provider)	233
9.1	Einführungsbeispiel: Navigation in der Registrierungsdatenbank	233
9.2	Provider und Laufwerke	234
9.3	Navigationsbefehle	237
9.4	Pfadangaben	237
9.5	Beispiel	239
9.6	Eigene Laufwerke definieren	240
10	Fernauführung (Remoting)	241
10.1	RPC-Fernabfrage ohne WS-Management	242
10.2	Anforderungen an PowerShell Remoting	243
10.3	Rechte für PowerShell-Remoting	244
10.4	Einrichten von PowerShell Remoting	245
10.5	Überblick über die Fernauführungs-Commandlets	247
10.6	Interaktive Fernverbindungen im Telnet-Stil	248
10.7	Fernauführung von Befehlen	249
10.8	Parameterübergabe an die Fernauführung	253
10.9	Fernauführung von Skripten	254
10.10	Ausführung auf mehreren Computern	255
10.11	Sitzungen	256
10.12	Implizites Remoting	261
10.13	Zugriff auf entfernte Computer außerhalb der eigenen Domäne	262
10.14	Verwaltung des WS-Management-Dienstes	265
10.15	PowerShell Direct für Hyper-V	267
10.16	Praxisbeispiel zu PowerShell Direct	269
11	PowerShell-Werkzeuge	273
11.1	PowerShell-Standardkonsole	273

11.2	PowerShell Integrated Scripting Environment (ISE)	282
11.3	PowerShell Script Analyzer	292
11.4	PowerShell Analyzer	298
11.5	PowerShell Tools for Visual Studio	299
11.6	PowerShell Pro Tools for Visual Studio	300
11.7	NuGet Package Manager	301
11.8	PowerShell-Erweiterung für Visual Studio Code	301
11.9	PowerShell Web Access (PSWA)	304
11.10	Azure Cloud Shell	310
11.11	ISE Steroids	310
11.12	PowerShellPlus	311
11.13	PoshConsole	314
11.14	PowerGUI	315
11.15	PrimalScript	316
11.16	PowerShell Help	318
11.17	CIM Explorer for PowerShell ISE	318
11.18	PowerShell Help Reader	319
11.19	PowerShell Remoting	320
12	Windows PowerShell Core 5.1 in Windows Nano Server	321
13	PowerShell Core 6.x für Windows, Linux und macOS	323
13.1	Geschichte der PowerShell Core	323
13.2	Motivation für den Einsatz der PowerShell Core auf Linux und macOS	324
13.3	Funktionsumfang der PowerShell Core	325
13.4	Entfallene Commandlets in PowerShell Core	327
13.5	Erweiterungsmodule nutzen in PowerShell Core	332
13.6	Geänderte Funktionen in PowerShell Core	337
13.7	Neue Funktionen der PowerShell Core	340
13.8	PowerShell Core-Konsole	342
13.9	VSCoDe-PowerShell als Editor für PowerShell Core	342
13.10	Verwendung auf Linux und macOS	347
13.11	PowerShell-Remoting via SSH	353
13.12	Dokumentation zur PowerShell Core	357
13.13	Quellcode zur PowerShell Core	359
Teil B:	PowerShell-Aufbauwissen	361
14	Verwendung von .NET-Klassen	363
14.1	.NET versus .NET Core	363
14.2	.NET-Bibliotheken	364

14.3	Microsoft Developer Network (MSDN)	365
14.4	Überblick über die Verwendung von .NET-Klassen	366
14.5	Erzeugen von Instanzen	367
14.6	Parameterbehaftete Konstruktoren	369
14.7	Initialisierung von Objekten	370
14.8	Nutzung von Attributen und Methoden	371
14.9	Statische Mitglieder in .NET-Klassen und statische .NET-Klassen	373
14.10	Generische Klassen nutzen	376
14.11	Zugriff auf bestehende Objekte	378
14.12	Laden von Assemblies	378
14.13	Verwenden von Nuget-Assemblies	381
14.14	Objektanalyse	383
14.15	Auflistungen (Enumerationen)	384
14.16	Verknüpfen von Aufzählungswerten	385
15	Verwendung von COM-Klassen	387
15.1	Erzeugen von COM-Instanzen	387
15.2	Nutzung von Attributen und Methoden	388
15.3	Liste aller COM-Klassen	389
15.4	Holen bestehender COM-Instanzen	390
15.5	Distributed COM (DCOM)	390
16	Zugriff auf die Windows Management Instrumentation (WMI) ..	391
16.1	Einführung in WMI	391
16.2	WMI in der PowerShell	418
16.3	Open Management Infrastructure (OMI)	420
16.4	Abruf von WMI-Objektmenen	420
16.5	Fernzugriffe	421
16.6	Filtern und Abfragen	421
16.7	Liste aller WMI-Klassen	425
16.8	Hintergrundwissen: WMI-Klassenprojektion mit dem PowerShell-WMI-Objektadapter	426
16.9	Beschränkung der Ausgabeliste bei WMI-Objekten	430
16.10	Zugriff auf einzelne Mitglieder von WMI-Klassen	432
16.11	Werte setzen in WMI-Objekten	432
16.12	Umgang mit WMI-Datumsangaben	434
16.13	Methodenaufrufe	435
16.14	Neue WMI-Instanzen erzeugen	436
16.15	Instanzen entfernen	437
16.16	Commandlet Definition XML-Datei (CDXML)	438

17	Dynamische Objekte	441
17.1	Erweitern bestehender Objekte	441
17.2	Komplett dynamische Objekte	443
18	Einbinden von C# und Visual Basic .NET	445
19	Win32-API-Aufrufe	447
20	Benutzereingaben	451
20.1	Read-Host	451
20.2	Benutzerauswahl	452
20.3	Grafischer Eingabedialog	453
20.4	Dialogfenster	454
20.5	Authentifizierungsdialg	454
20.6	Zwischenablage (Clipboard)	456
21	Fehlersuche	459
21.1	Detailinformationen	459
21.2	Einzelschrittmodus	460
21.3	Zeitmessung	461
21.4	Ablaufverfolgung (Tracing)	462
21.5	Erweiterte Protokollierung aktivieren	463
21.6	Script-Debugging in der ISE	465
21.7	Kommandozeilenbasiertes Script-Debugging	465
22	Transaktionen	467
22.1	Commandlets für Transaktionen	467
22.2	Start und Ende einer Transaktion	468
22.3	Zurücksetzen der Transaktion	469
22.4	Mehrere Transaktionen	470
23	Standardeinstellungen ändern mit Profilskripten	471
23.1	Profilpfade	471
23.2	Ausführungsreihenfolge	473
23.3	Beispiel für eine Profildatei	473
23.4	Starten der PowerShell ohne Profilskripte	474
24	Digitale Signaturen für PowerShell-Skripte	475
24.1	Zertifikat erstellen	475
24.2	Skripte signieren	477
24.3	Verwenden signierter Skripte	478
24.4	Mögliche Fehlerquellen	479

25	Hintergrundaufträge („Jobs“)	481
25.1	Voraussetzungen	481
25.2	Architektur	482
25.3	Starten eines Hintergrundauftrags	482
25.4	Hintergrundaufträge abfragen	483
25.5	Warten auf einen Hintergrundauftrag	484
25.6	Abbrechen und Löschen von Aufträgen	484
25.7	Analyse von Fehlermeldungen	485
25.8	Fernausführung von Hintergrundaufträgen	485
25.9	Praxisbeispiel	486
26	Geplante Aufgaben und zeitgesteuerte Jobs	489
26.1	Geplante Aufgaben (Scheduled Tasks)	489
26.2	Zeitgesteuerte Jobs	493
27	PowerShell-Workflows	499
27.1	Ein erstes Beispiel	499
27.2	Unterschiede zu einer Function bzw. einem Skript	504
27.3	Einschränkungen bei Workflows	504
27.4	Workflows in der Praxis	506
27.5	Workflows in Visual Studio erstellen	513
28	Ereignissystem	531
28.1	WMI-Ereignisse	531
28.2	WMI-Ereignisabfragen	531
28.3	WMI-Ereignisse seit PowerShell 1.0	533
28.4	Registrieren von WMI-Ereignisquellen seit PowerShell 2.0	534
28.5	Auslesen der Ereignisliste	535
28.6	Reagieren auf Ereignisse	537
28.7	WMI-Ereignisse ab PowerShell-Version 3.0	539
28.8	Registrieren von .NET-Ereignissen	539
28.9	Erzeugen von Ereignissen	540
29	Datenbereiche und Datendateien	543
29.1	Datenbereiche	543
29.2	Datendateien	545
29.3	Mehrsprachigkeit/Lokalisierung	546
30	Desired State Configuration (DSC)	549
30.1	Grundprinzipien	550
30.2	DSC für Linux	550

30.3	Ressourcen	551
30.4	Verfügbare DSC-Ressourcen	551
30.5	Eigenschaften einer Ressource	554
30.6	Aufbau eines DSC-Dokuments	554
30.7	Commandlets für die Arbeit mit DSC	555
30.8	Ein erstes DSC-Beispiel	555
30.9	Kompilieren und Anwendung eines DSC-Dokuments	556
30.10	Variablen in DSC-Dateien	558
30.11	Parameter für DSC-Dateien	559
30.12	Konfigurationsdaten	560
30.13	Entfernen einer DSC-Konfiguration	563
30.14	DSC Pull Server	566
30.15	DSC-Praxisbeispiel 1: IIS installieren	574
30.16	DSC-Praxisbeispiel 2: Software installieren	575
30.17	DSC-Praxisbeispiel 3: Software deinstallieren	577
30.18	Realisierung einer DSC-Ressource	578
30.19	Weitere Möglichkeiten	579
31	PowerShell-Snap-Ins	581
31.1	Einbinden von Snap-Ins	581
31.2	Liste der Commandlets	585
32	PowerShell-Module	587
32.1	Überblick über die Commandlets	587
32.2	Modulararchitektur	588
32.3	Aufbau eines Moduls	589
32.4	Module aus dem Netz herunterladen und installieren mit PowerShellGet	589
32.5	Module manuell installieren	596
32.6	Doppeldeutige Namen	596
32.7	Auflisten der verfügbaren Module	598
32.8	Importieren von Modulen	599
32.9	Entfernen von Modulen	602
33	Ausgewählte PowerShell-Erweiterungen	603
33.1	PowerShell-Module in Windows 7 und Windows Server 2008 R2	604
33.2	PowerShell-Module in Windows 8.0 und Windows Server 2012	605
33.3	PowerShell-Module in Windows 8.1 und Windows Server 2012 R2	607
33.4	PowerShell-Module in Windows 10 und Windows Server 2016	610
33.5	PowerShell Community Extensions (PSCX)	614
33.6	PowerShellPack	618
33.7	www.IT-Visions.de: PowerShell Extensions	619

33.8	Quest Management Shell for Active Directory	620
33.9	Microsoft Exchange Server	621
33.10	System Center Virtual Machine Manager	622
33.11	PowerShell Management Library for Hyper-V (pshyperv)	623
33.12	Powershell Outlook Account Manager	624
33.13	PowerShell Configurator (PSConfig)	625
33.14	Weitere Erweiterungen	626
34	Delegierte Administration/Just Enough Administration (JEA) ..	627
34.1	JEA-Konzept	627
34.2	PowerShell-Sitzungskonfiguration erstellen	627
34.3	Sitzungskonfiguration nutzen	631
34.4	Delegierte Administration per Webseite	632
35	Tipps und Tricks zur PowerShell	633
35.1	Alle Anzeigen löschen	633
35.2	Befehlsgeschichte	633
35.3	System- und Hostinformationen	634
35.4	Anpassen der Eingabeaufforderung (Prompt)	635
35.5	PowerShell-Befehle aus anderen Anwendungen heraus starten	636
35.6	ISE erweitern	637
35.7	PowerShell für Gruppenrichtlinienskripte	638
35.8	Einblicke in die Interna der Pipeline-Verarbeitung	640
	Teil C: PowerShell im Praxiseinsatz	643
36	Dateisystem	645
36.1	Laufwerke	646
36.2	Ordnerinhalte	651
36.3	Dateieigenschaften verändern	653
36.4	Eigenschaften ausführbarer Dateien	654
36.5	Kurznamen	656
36.6	Lange Pfade	656
36.7	Dateisystemoperationen	657
36.8	Praxisbeispiel: Zufällige Dateisystemstruktur erzeugen	658
36.9	Praxisbeispiel: Leere Ordner löschen	659
36.10	Einsatz von Robocopy	660
36.11	Dateisystemkataloge	663
36.12	Papierkorb leeren	664
36.13	Dateieigenschaften lesen	664
36.14	Praxisbeispiel: Fotos nach Aufnahmedatum sortieren	665

36.15	Datei-Hash	666
36.16	Finden von Duplikaten	667
36.17	Verknüpfungen im Dateisystem	669
36.18	Komprimierung	674
36.19	Dateisystemfreigaben	676
36.20	Überwachung des Dateisystems	687
36.21	Dateiversionsverlauf	688
36.22	Windows Explorer öffnen	689
36.23	Windows Server Backup	689
37	Festplattenverschlüsselung mit BitLocker	693
37.1	Übersicht über das BitLocker-Modul	694
37.2	Verschlüsseln eines Laufwerks	695
38	Dokumente	697
38.1	Textdateien	697
38.2	CSV-Dateien	698
38.3	Analysieren von Textdateien	701
38.4	INI-Dateien	704
38.5	XML-Dateien	705
38.6	HTML-Dateien	713
38.7	Binärdateien	713
39	Datenbanken	715
39.1	ADO.NET-Grundlagen	715
39.2	Beispieldatenbank	721
39.3	Datenzugriff mit den Bordmitteln der PowerShell	722
39.4	Hilfsroutinen für den Datenbankzugriff (DBUtil.ps1)	733
39.5	Datenzugriff mit den PowerShell-Erweiterungen	736
39.6	Datenbankzugriff mit SQLPS	740
39.7	Datenbankzugriff mit SQLPSX	740
40	Microsoft-SQL-Server-Administration	741
40.1	PowerShell-Integration im SQL Server Management Studio	742
40.2	SQL-Server-Laufwerk „SQLSERVER:“	743
40.3	Die SQLPS-Commandlets	746
40.4	Die SQL Server Management Objects (SMO)	748
40.5	SQLPSX	751
40.6	Microsoft-SQL-Server-Administration mit der PowerShell in der Praxis	759

41	ODBC-Datenquellen	765
41.1	ODBC-Treiber und -Datenquellen auflisten	766
41.2	Anlegen einer ODBC-Datenquelle	767
41.3	Zugriff auf eine ODBC-Datenquelle	768
42	Registrierungsdatenbank (Registry)	771
42.1	Schlüssel auslesen	771
42.2	Schlüssel anlegen und löschen	772
42.3	Laufwerke definieren	772
42.4	Werte anlegen und löschen	773
42.5	Werte auslesen	774
42.6	Praxisbeispiel: Windows-Explorer-Einstellungen	774
42.7	Praxisbeispiel: Massenanlegen von Registry-Schlüsseln	775
43	Computer- und Betriebssystemverwaltung	777
43.1	Computerinformationen	777
43.2	Versionsnummer des Betriebssystems	779
43.3	Zeitdauer seit dem letzten Start des Betriebssystems	779
43.4	BIOS- und Startinformationen	780
43.5	Windows-Produktaktivierung	781
43.6	Umgebungsvariablen	781
43.7	Schriftarten	784
43.8	Computernamen und Domäne	784
43.9	Herunterfahren und Neustarten	785
43.10	Windows Updates installieren	786
43.11	Wiederherstellungspunkte verwalten	789
44	Windows Defender	791
45	Hardwareverwaltung	793
45.1	Hardwarebausteine	793
45.2	Plug-and-Play-Geräte	795
45.3	Druckerverwaltung (ältere Betriebssysteme)	795
45.4	Druckerverwaltung (seit Windows 8 und Windows Server 2012)	796
46	Softwareverwaltung	799
46.1	Softwareinventarisierung	799
46.2	Installation von Anwendungen	802
46.3	Deinstallation von Anwendungen	803
46.4	Praxisbeispiel: Installationstest	803
46.5	Installationen mit PowerShell Package Management („OneGet“)	804

46.6	Versionsnummer ermitteln	807
46.7	Servermanager	808
46.8	Windows-Features installieren auf Windows-Clientbetriebssystemen	819
46.9	Praxisbeispiel: IIS-Installation	822
46.10	Softwareeinschränkungen mit dem PowerShell-Modul „AppLocker“	824
47	Prozessverwaltung	831
47.1	Prozesse auflisten	831
47.2	Prozesse starten	832
47.3	Prozesse mit vollen Administratorrechten starten	833
47.4	Prozesse unter einem anderen Benutzerkonto starten	834
47.5	Prozesse beenden	835
47.6	Warten auf das Beenden einer Anwendung	836
48	Windows- Systemdienste	837
48.1	Dienste auflisten	837
48.2	Dienstzustand ändern	840
48.3	Diensteigenschaften ändern	840
48.4	Dienste hinzufügen	841
48.5	Dienste entfernen	842
49	Netzwerk	843
49.1	Netzwerkkonfiguration (ältere Betriebssysteme)	843
49.2	Netzwerkkonfiguration (ab Windows 8 und Windows Server 2012)	845
49.3	DNS-Client-Konfiguration	848
49.4	DNS-Namensauflösung	851
49.5	Erreichbarkeit prüfen (Ping)	853
49.6	Windows Firewall	854
49.7	Remote Desktop (RDP) einrichten	860
49.8	E-Mails senden (SMTP)	861
49.9	Auseinandernehmen von E-Mail-Adressen	863
49.10	Abruf von Daten von einem HTTP-Server	863
49.11	Praxisbeispiel: Linkprüfer für eine Website	865
49.12	Aufrufe von SOAP-Webdiensten	868
49.13	Aufruf von REST-Diensten	870
49.14	Aufrufe von OData-Diensten	872
49.15	Hintergrunddatentransfer mit BITS	873
50	Ereignisprotokolle (Event Log)	877

51	Leistungsdaten (Performance Counter)	881
51.1	Zugriff auf Leistungsindikatoren über WMI	881
51.2	Get-Counter	882
52	Sicherheitseinstellungen	885
52.1	Aktueller Benutzer	885
52.2	Grundlagen	886
52.3	Zugriffsrechtelisten auslesen	891
52.4	Einzelne Rechteinträge auslesen	892
52.5	Besitzer auslesen	894
52.6	Benutzer und SID	894
52.7	Hinzufügen eines Rechteintrags zu einer Zugriffsrechteliste	897
52.8	Entfernen eines Rechteintrags aus einer Zugriffsrechteliste	900
52.9	Zugriffsrechteliste übertragen	901
52.10	Zugriffsrechteliste über SDDL setzen	902
52.11	Zertifikate verwalten	903
53	Optimierungen und Problemlösungen	907
53.1	PowerShell-Modul „TroubleshootingPack“	907
53.2	PowerShell-Modul „Best Practices“	911
54	Active Directory	913
54.1	Benutzer- und Gruppenverwaltung mit WMI	914
54.2	Einführung in System.DirectoryServices	915
54.3	Basiseigenschaften	926
54.4	Benutzer- und Gruppenverwaltung im Active Directory	928
54.5	Verwaltung der Organisationseinheiten	936
54.6	Suche im Active Directory	937
54.7	Navigation im Active Directory mit den PowerShell Extensions	944
54.8	Verwendung der Active-Directory-Erweiterungen von www.IT-Visions.de	945
54.9	PowerShell-Modul „Active Directory“ (ADPowerShell)	947
54.10	PowerShell-Modul „ADDSDeployment“	975
54.11	Informationen über die Active-Directory-Struktur	978
55	Gruppenrichtlinien	981
55.1	Verwaltung der Gruppenrichtlinien	982
55.2	Verknüpfung der Gruppenrichtlinien	983
55.3	Gruppenrichtlinienberichte	985
55.4	Gruppenrichtlinienvererbung	986
55.5	Weitere Möglichkeiten	987

56	Lokale Benutzer und Gruppen	989
56.1	Modul „Microsoft.PowerShell.LocalAccounts“	989
56.2	Lokale Benutzerverwaltung in älteren PowerShell-Versionen	991
57	Microsoft Exchange Server	993
57.1	Daten abrufen	993
57.2	Postfächer verwalten	994
57.3	Öffentliche Ordner verwalten	995
58	Internet Information Services (IIS)	997
58.1	Überblick	997
58.2	Navigationsprovider	999
58.3	Anlegen von Websites	1001
58.4	Praxisbeispiel: Massenanlegen von Websites	1002
58.5	Ändern von Website-Eigenschaften	1005
58.6	Anwendungspool anlegen	1005
58.7	Virtuelle Verzeichnisse und IIS-Anwendungen	1006
58.8	Website-Zustand ändern	1007
58.9	Anwendungspools starten und stoppen	1007
58.10	Löschen von Websites	1008
59	Virtuelle Systeme mit Hyper-V	1009
59.1	Das Hyper-V-Modul von Microsoft	1010
59.2	Die ersten Schritte mit dem Hyper-V-Modul	1012
59.3	Virtuelle Maschinen anlegen	1016
59.4	Umgang mit virtuellen Festplatten	1022
59.5	Konfiguration virtueller Maschinen	1025
59.6	Dateien kopieren in virtuelle Systeme	1029
59.7	PowerShell Management Library for Hyper-V (für ältere Betriebssysteme)	1031
60	Windows Nano Server	1035
60.1	Das Konzept von Nano Server	1035
60.2	Einschränkungen von Nano Server	1037
60.3	Varianten des Nano Servers	1039
60.4	Installation eines Nano Servers	1039
60.5	Docker-Image	1041
60.6	Fernverwaltung mit PowerShell	1041
60.7	Windows Update auf einem Nano Server	1044
60.8	Nachträgliche Paketinstallation	1044
60.9	Abgespeckter IIS unter Nano Server	1046
60.10	Nano-Serververwaltung aus der Cloud heraus	1047

61	Docker-Container	1049
61.1	Docker-Varianten für Windows	1050
61.2	Docker-Installation auf Windows 10	1051
61.3	Docker-Installation auf Windows Server 2016	1053
61.4	Installation von „Docker for Windows“	1054
61.5	Docker-Registries	1056
61.6	Docker-Images laden	1056
61.7	Container starten	1057
61.8	Container-Identifikation	1058
61.9	Container mit Visual Studio	1059
61.10	Befehle in einem Container ausführen	1061
61.11	Ressourcenbeschränkungen für Container	1063
61.12	Dateien zwischen Container und Host kopieren	1063
61.13	Dockerfile	1063
61.14	Docker-Netzwerke	1064
61.15	Container anlegen, ohne sie zu starten	1065
61.16	Container starten und stoppen	1065
61.17	Container beenden und löschen	1065
61.18	Images löschen	1066
61.19	Images aus Containern erstellen	1066
61.20	.NET Core-Container	1066
61.21	Images verbreiten	1069
61.22	Azure Container Service (ACS)	1071
62	Grafische Benutzeroberflächen (GUI)	1073
62.1	Einfache Nachfragedialoge	1073
62.2	Einfache Eingabe mit Inputbox	1075
62.3	Komplexere Eingabemasken	1076
62.4	Universelle Objektdarstellung	1078
62.5	WPF PowerShell Kit (WPK)	1079
62.6	Direkte Verwendung von WPF	1087
Teil D: Profiwissen – Erweitern der PowerShell		1089
63	Entwicklung von Commandlets in der PowerShell-Skriptsprache	1091
63.1	Aufbau eines skriptbasierten Commandlets	1091
63.2	Verwendung per Dot Sourcing	1093
63.3	Parameterfestlegung	1094
63.4	Fortgeschrittene Funktion (Advanced Function)	1100

63.5	Mehrere Parameter und Parametersätze	1102
63.6	Unterstützung für Sicherheitsabfragen (-Whatif und -Confirm)	1104
63.7	Kaufmännisches Beispiel: Test-CustomerID	1106
63.8	Erweitern bestehender Commandlets durch Proxy-Commandlets	1109
63.9	Dokumentation	1115
64	Entwicklung eigener Commandlets mit C#	1119
64.1	Technische Voraussetzungen	1120
64.2	Grundkonzept der .NET-basierten Commandlets	1121
64.3	Schrittweise Erstellung eines minimalen Commandlets	1123
64.4	Erstellung eines Commandlets mit einem Rückgabeobjekt	1131
64.5	Erstellung eines Commandlets mit mehreren Rückgabeobjekten	1133
64.6	Erstellen eines Commandlets mit Parametern	1137
64.7	Verarbeiten von Pipeline-Eingaben	1139
64.8	Verkettung von Commandlets	1142
64.9	Fehlersuche in Commandlets	1146
64.10	Statusinformationen	1149
64.11	Unterstützung für Sicherheitsabfragen (-whatif und -confirm)	1154
64.12	Festlegung der Hilfeinformationen	1156
64.13	Erstellung von Commandlets für den Zugriff auf eine Geschäftsanwendung ..	1160
64.14	Konventionen für Commandlets	1161
64.15	Weitere Möglichkeiten	1163
65	PowerShell-Module erstellen	1165
65.1	Erstellen eines Skriptmoduls	1165
65.2	Praxisbeispiel: Umwandlung einer Skriptdatei in ein Modul	1167
65.3	Erstellen eines Moduls mit Binärdateien	1167
65.4	Erstellen eines Moduls mit Manifest	1168
65.5	Erstellung eines Manifest-Moduls mit Visual Studio	1175
66	Hosting der PowerShell	1177
66.1	Voraussetzungen für das Hosting	1178
66.2	Hosting mit PSHost	1179
66.3	Vereinfachtes Hosting seit PowerShell 2.0	1182
Anhang A: Crashkurs „Objektorientierung“	1185	
Anhang B: Crashkurs .NET	1193	
B.1	Was ist das .NET Framework?	1195
B.2	Was ist .NET Core?	1196
B.3	Eigenschaften von .NET	1197

B.4	.NET-Klassen	1198
B.5	Namensgebung von .NET-Klassen (Namensräume)	1198
B.6	Namensräume und Softwarekomponenten	1200
B.7	Bestandteile einer .NET-Klasse	1201
B.8	Vererbung	1202
B.9	Schnittstellen	1203
Anhang C: Literatur		1204
Anhang D: Weitere Informationen im Internet		1207
Anhang E: Abkürzungsverzeichnis		1209
Stichwortverzeichnis		1235

Vorwort

Liebe Leserin, lieber Leser,

willkommen zur aktuellen Auflage des PowerShell-Buchs! Es handelt sich hierbei um die dritte Auflage des Windows PowerShell 5-Buches und die siebte Auflage des PowerShell-Buches insgesamt, das erstmalig 2007 bei Addison-Wesley erschienen ist. Das vor Ihnen liegende Buch behandelt die Windows PowerShell in der Version 5.1 sowie die PowerShell Core in der Version 6.0/6.1 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 45 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 68. 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 Core 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.

Die PowerShell Core 6.0, die zum letzten Redaktionsschluss noch auf dem Beta-Stand war, ist inzwischen erschienen. Das Buch wurde auf die RTM-Version 6.0 sowie die Release Candidate-Version von PowerShell Core 6.1 aktualisiert. Zudem wurde ein Unterkapitel zur Installation von Windows-Features in Client-Betriebssystemen im Kapitel „Teil C/Softwareverwaltung“ sowie ein Praxisbeispiel zur Installation des Webservers „IIS“ ergänzt. Außerdem wurde das Kapitel „Verwendung von .NET-Klassen“ erweitert.

■ 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 Einflüssen steht natürlich auch dieses etablierte Buch. Leider kann man ein Buch nicht so schnell ändern wie Software.

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         PowerShell00P
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/.NET Core, sondern zu fast allen modernen Techniken der Entwicklung und des Betriebs von Software in großen Unternehmen. Wir besuchen Sie gerne in Ihrem Unternehmen an einem beliebigen Standort.

■ Zum Schluss des Vorworts ...

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

Dr. Holger Schwichtenberg

Essen, im Oktober 2018

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.

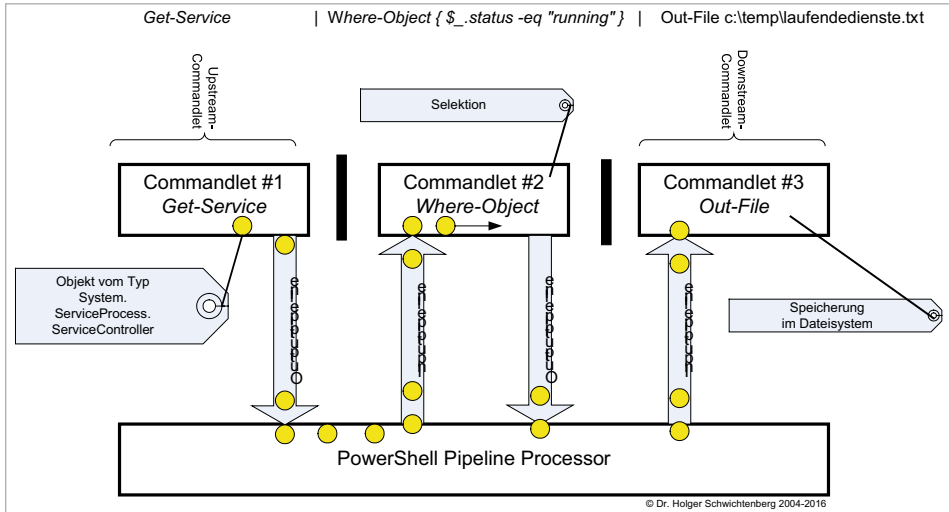


Bild 5.1 Der Pipeline Processor befördert die Objekte vom Downstream-Commandlet zum Upstream-Commandlet. Die Verarbeitung ist in der Regel asynchron.

Wie das obige Bild schon zeigt, beginnt ein nachfolgendes Commandlet mit seiner Arbeit, sobald es ein erstes Objekt aus der Pipeline erhält. Das Objekt durchläuft die komplette Pipeline. Erst dann wird das nächste Objekt vom ersten Commandlet abgeholt. Man nennt dies „Streaming-Verarbeitung“. Streaming-Verarbeitung ist schneller als die klassische sequentielle Verarbeitung, weil die folgenden Commandlets in der Pipeline nicht auf vorhergehende warten müssen. Intern arbeitet die PowerShell aber nur mit einem Thread, d. h. es findet keine parallele Verarbeitung mehrerer Befehle statt.

Aber nicht alle Commandlets beherrschen die asynchrone Streaming-Verarbeitung. Commandlets, die alle Objekte naturgemäß erst mal kennen müssen, bevor sie überhaupt ihren Zweck erfüllen können (z. B. `Sort-Object` zum Sortieren und `Group-Object` zum Gruppieren), blockieren die asynchrone Verarbeitung.



HINWEIS: Es gibt auch einige Commandlets, die zwar asynchron arbeiten könnten, aber leider nicht so programmiert wurden, um dies zu unterstützen.

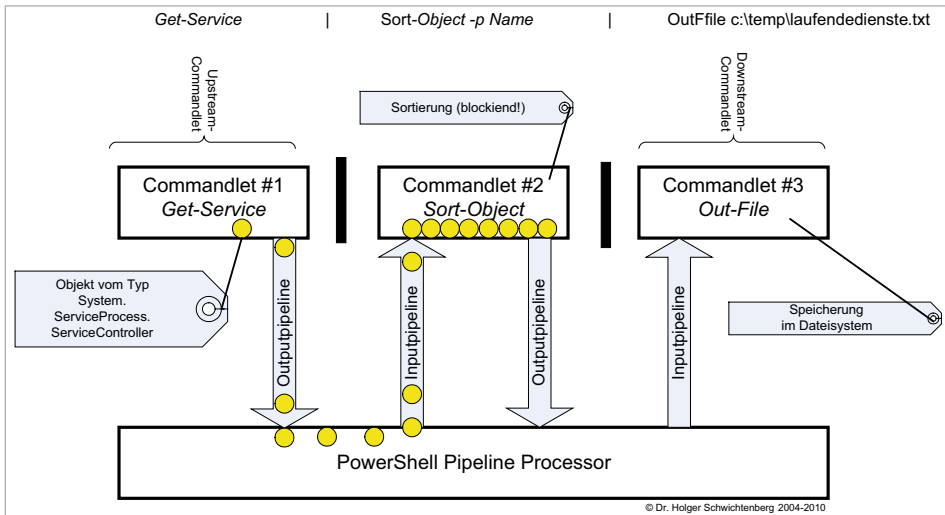


Bild 5.2 Sort-Object blockiert die direkte Weitergabe. Erst wenn alle Objekte angekommen sind, kann das Commandlet sortieren.

Auch bei Commandlets, die Streaming-Verarbeitung unterstützen, kann der PowerShell-Nutzer mit dem allgemeinen Parameter `-OutBuffer` (abgekürzt `-ob`), den jedes Commandlet anbietet, dafür sorgen, dass eine bestimmte Anzahl von Objekten angesammelt wird, bevor eine Weitergabe an das nachfolgende Commandlet erfolgt.

Im Standard beginnt die Ausgabe der Ordner- und Dateinamen sofort:

```
dir c:\ -Recurse | ft name
```

In diesem Fall passiert lange nichts, bevor die Ausgabe beginnt:

```
dir c:\ -Recurse -OutBuffer:100000 | ft name
```

■ 5.4 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 <scriptblock>
  Specifies the script block that is used to filter the objects. Enclose the
  script block in braces < > .

  Required?                true
  Position?                1
  Default value
  Accept pipeline input?   false
  Accept wildcard characters? false

-InputObject <psobject>
  Specifies the objects to be filtered. You can also pipe the objects to Where-Object.

  Required?                false
  Position?                named
  Default value
  Accept pipeline input?   true <ByValue>
  Accept wildcard characters? false

PS P:\> _
```

Bild 5.4 Parameter des Commandlets `Where-Object`

Leider geht es nicht bei allen Commandlets so einfach mit der Parameterübergabe. Man nehme zum Beispiel das Commandlet `Test-Connection`, das prüft, ob ein Computer per Ping erreichbar ist.

Der normale Aufruf mit Parameter ist:

```
Test-Connection -computername Server123
```

oder ohne benannten Parameter

```
Test-Connection Server123
```

Nun könnte man auf die Idee kommen, hier den Computernamen genau so zu übergeben, wie den Namen bei `Get-Service`. Allerdings liefert `"Server123" | Test-Connection` den Fehler: *„The input object cannot be bound to any parameters for the command either because the command does not take pipeline input or the input and its properties do not match any of the parameters that take pipeline input.“*

Warum das nicht geht, kann man in der Hilfe zum Parameter `ComputerName` des Commandlets `Test-Connection` erkennen. Dort steht, dass `ComputerName` nur als „ByPropertyName“ akzeptiert wird und nicht wie beim Parameter `Name` beim Commandlet `Get-Service` auch „ByValue“. Das bedeutet also, dass man erst ein Objekt mit der Eigenschaft `ComputerName` konstruieren und dann übergeben muss:

```
New-Object psobject -Property @{ComputerName="Server123"} | Test-Connection
```


Das funktioniert zwar, ist aber hässlich und umständlich. Warum Test-Connection und einige andere Commandlets die Eingaben nicht „ByValue“ unterstützen, wusste übrigens das PowerShell-Entwicklungsteam auf Nachfrage auch nicht zu beantworten. Die Schuld liegt hier vermutlich bei dem einzelnen Entwickler bei Microsoft, der die Commandlets implementiert hat.

```

-ComputerName <string[]>

Required?                true
Position?                0
Accept pipeline input?   true (ByPropertyName)
Parameter set name       (All)
Aliases                  CN, IPAddress, __SERVER, Server, Destination
Dynamic?                 false
  
```

Bild 5.5 Hilfe zum Parameter ComputerName des Commandlets Test-Connection

■ 5.5 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.

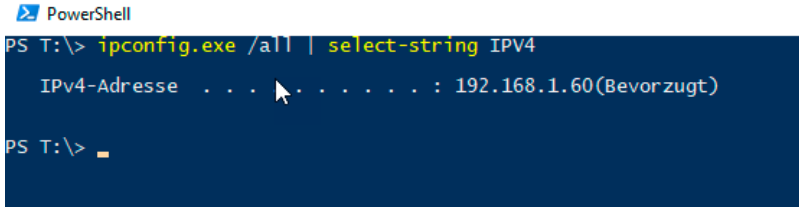
```

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:http 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 Klammernpaars

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	pagererror.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.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 5.20 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;}
PeakPageMemorySize   Property           System.Int32 PeakPageMemorySize {get;}
PeakPageMemorySize64 Property          System.Int64 PeakPageMemorySize64 {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;}
StartInfo              Property           System.DateTime 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;}
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...

PS C:\> _
  
```

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 sogenannten 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 = PagedMemorySize
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()
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.


```

PS T:\> Get-Process | Select-Object psResources | Format-Table
Name                                     Id HandleCount WorkingSet PagedMemorySize PrivateMemorySize VirtualMemorySize TotalProcessorTime
-----
AcroRd32                                7264      696  146747392      123883520      123883520      380379136 00:00:03.5312500
AcroRd32                                13724     449  26161152      11595776      11595776      150568960 00:00:00.1562500
armsvc                                  4572       143   6991872      1921024      1921024      65519616
atetclxx                                 3252      225  10227712      2854912      2854912      109641728
atiesrxx                                 3084       140   6004736      1875968      1875968      42979328
audiogdg                                 7244      210  14209024      8101888      8101888      64712704 00:00:00.6250000
AVKProxy                                 4652       368   4210688      7131136      7131136      127819776
AVKNet1x64                               2880     1649  186978304      171806720      171806720      413323264
backgroundTaskHost                       11324    214  16220160      4587520      4587520      111869952 00:00:00.0468750
ccc                                       2104      924   7221248      80384000      80384000      908488704 00:00:02.8125000
chrome                                   608       401  94724096      65105920      65105920      1025171456 00:00:01.8593750
chrome                                   1840      204   9650176      2387968      2387968      105435136 00:00:00.0468750
chrome                                   2024     1578  181137408      117514240      117514240      524996608 00:00:09.1718750
chrome                                   10272    289   37580800      24600576      24600576      810311680 00:00:00.1718750
chrome                                   11124    285   33673216      21262336      21262336      799825920 00:00:00.2968750
chrome                                   11532    144  10096640      2252800      2252800      98193408 00:00:00.0156250
chrome                                   12232    318   39460864      24760320      24760320      814440448 00:00:00.3906250
chrome                                   13792    578   66674688      72155136      72155136      483831808 00:00:01.5468750
chrome                                   13796    290   40976384      29523968      29523968      812408832 00:00:00.3593750
chrome                                   14252    287   37462016      24379392      24379392      806641664 00:00:00.3750000
conhost                                  9420     229  15302656      4157440      4157440      113057792 00:00:00.2656250
csrss                                    628       746   5365760      2158592      2158592      61370368
csrss                                    736       606   5709824      10141696      10141696      72417280
dasHost                                  5632     271  14905344      4575232      4575232      69120000
dllhost                                  9320     247  33275904      24334336      24334336      393596928 00:00:00.2656250
dllhost                                  10648    172  11730944      4620288      4620288      354037760 00:00:00.1093750
PSAService                               4556      614  41193472      26361856      26361856      237490176
OSATray                                  604       493  44281856      38748160      38748160      320065536 00:00:00.4687500
dwm                                       1388      712  135815168      148025344      148025344      429637632
explorer                                  7380     2580  130813952      61431808      61431808      563249152 00:00:10.7812500
ExpressTray                              12992    988   70295552      56795136      56795136      409272320 00:00:01.0625000
fontdrvhost                              536       46   4747264      2023424      2023424      63488000
fontdrvhost                              548       46  11591680      4321280      4321280      148639744
garminservice                             4564     1274  70078464      45420544      45420544      318988288
gdAgentSvc                                4696      616   3436544      7983104      7983104      141025280
gdAgentUI                                 10968    295   1474560      3985408      3985408      116932608 00:00:00.0781250
GDScan                                    2008      735  43450368      692846592      692846592      831651840
GoodSync-v10                              12896   391  279642112      267563008      267563008      439402496 00:00:48.9218750
GoogleCrashHandler                       6268     154   995328      2105344      2105344      67391488
GoogleCrashHandler64                     1420     136   741376      1953792      1953792      70037504
ps-server                                  6692     381  17182720      9486336      9486336      106037248
Idle                                       0         0         8192      53248      53248      65536
jexplore                                  2456     630  39108608      13930496      13930496      211484672 00:00:00.5000000
jexplore                                  4536     649  62017536      35028992      35028992      305123328 00:00:00.3281250
IpOverUsbSvc                              4444     262  13094912      8650752      8650752      128581632

```

Bild 5.23 Verwendung des Eigenschaftssatzes „psResources“

```

<PropertySet>
  <Name>PSCConfiguration</Name>
  <ReferencedProperties>
    <Name>Name</Name>
    <Name>Id</Name>
    <Name>PriorityClass</Name>
    <Name>FileVersion</Name>
  </ReferencedProperties>
</PropertySet>
<PropertySet>
  <Name>PSResources</Name>
  <ReferencedProperties>
    <Name>Name</Name>
    <Name>Id</Name>
    <Name>HandleCount</Name>
    <Name>WorkingSet</Name>
    <Name>NonPagedMemorySize</Name>
    <Name>PagedMemorySize</Name>
    <Name>PrivateMemorySize</Name>
    <Name>VirtualMemorySize</Name>
    <Name>Threads.Count</Name>
    <Name>TotalProcessorTime</Name>
  </ReferencedProperties>
</PropertySet>

```

Bild 5.24

Definition der Eigenschaftssätze für die Klasse System.Diagnostics.Process in types.ps1ml

Notizeigenschaften (NoteProperty)

Notizeigenschaften (NoteProperties) sind zusätzliche Datenelemente, die nicht dem .NET-Objekt entstammen, sondern welche die PowerShell-Infrastruktur hinzugefügt hat. Im Beispiel der Ergebnismenge des Commandlets `Get-Process` ist dies `__NounName`, der einen Kurznamen der Klasse liefert. Andere Klassen haben zahlreiche Notizeigenschaften. Notizeigenschaften gibt es nicht im .NET Framework; sie sind eine Eigenart der PowerShell.

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: Adapted Type System (ATS)/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.psxml*-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 { $_.DisplayName -match "^i\w{3}$" }
-----
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 zwei 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=0EAD1870
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.

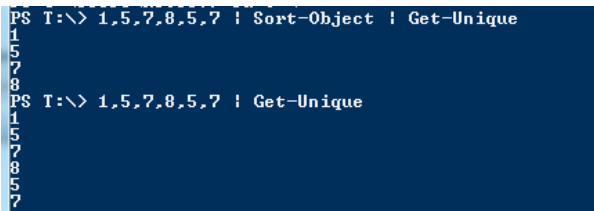
Achtung: Die Liste muss vorher sortiert sein!

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...}
18 .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, mqa.tlb, mqa10.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, kanji1.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 {AppVStreamingUX.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

\$PSVersionTable 11
\$_ 89, 99, 151, 444
& 66, 181
> 229
>> 229
-as 148
-Bit 5
-cmatch 165
-cnotmatch 165
-expression 640
-imatch 165
-inotmatch 165
-ItemsSource 1087
-Join 164
-match 165
-notmatch 165
-Parameter 73
-Split 164
-Verbose 52
.cat 663
.dll 47, 139, 378, 654, 1169 f.
.exe 139, 300
.NET 3, 40, 88, 104, 151, 173, 369,
454, 1073, 1119, 1146, 1160, 1193
- Bibliothek 363
- Klasse 363, 1198
- Runtime Host 37
.NET API Portability Analyzer 1038
.NET Core 3, 19, 40, 327, 1056,
1066, 1193, 1196
.NET Data Provider 715 f.
.NET Framework 31, 37, 243, 499,
581, 809, 1193, 1195, 1199
- 4.0 5
.NET Standard 364, 1197
.nupkg 382
.pfx 477
.pkg 22
.ps1 13, 62, 129, 1170

.psd1 547, 589, 1094, 1170, 1175
.psml 1094, 1165 ff., 1170, 1175
.psproj 299
.yaml 1068
32-Bit 5, 283, 721, 778
64-Bit 283, 721, 778
[Type] 374

A

Ablaufverfolgung 3, 462 f.
About 141
Absent 564
AbsoluteTimerInstruction 408
abstract 1191
Accelerator 146
Accent Grave
- Gravis 53
Access Control Entry 886
Access Control List 886, 897
Access Control Type 887
Access Mask 886
AccessControl 885, 890
AccessMask 678
Account Manager 603, 624
AccountDisabled 932
AceFlags 887
ACL 889, 902
ACS 1071
Active Directory 3, 235, 394, 553,
603 f., 620, 922, 928, 932, 947,
977 f., 1095
- PowerShell 947
- Struktur 978
- Suche 937
Active Directory Application Mode
978
Active Directory Domain Services
976
Active Directory Service Interface
siehe ADSI

Active-Scripting 135
ActiveScriptEventConsumer 409
ActiveX Data Objects 715, 726, 918
ADAccount 953
ADComputer 953
Add() 377
Add-ADGroupMember 957, 974
Add-Computer 785
Add-Content 697, 713
Add-DirectoryEntry 619
Add-DistributionGroupMember
994
Add-Feature 693
Add-JobTrigger 493
Add-LDAPObject 945, 1096 f.
Add-LocalGroupMember 327, 990
Add-Member 113, 441, 444, 1104
Add-ODBCDSN 765
Add-PSSnapin 328, 1123, 1128
Add-Type 379 f., 445, 453, 665, 749
Add-VirtualHardDisk 623
Add-VMDisk 1033
Add-VMDrive 1033
Add-VMHardDiskDrive 1011, 1023
Add-VMNIC 1033
Add-VMSwitch 1011
Add-WBSystemState 690
Add-WindowsCapability 820
Add-WindowsFeature 690, 808 f.,
813 ff., 976
AddCommand() 1183
AddScript() 1180, 1183
ADSDeployment 975, 977 f.
Administration
- delegiert 627
- webbasiert 304, 632
Administrator 267
Administratorrechte 244, 264, 267,
279 ff., 433, 592, 683, 833, 885,
1053, 1056
ADO.NET 715, 723, 918, 937

ADODB.Connection 919
 ADPowerShell 947, 953
 ADRMS 604
 ADSI 915, 919 f., 923, 925, 991
 – .NET 913, 915, 919
 – Bindung 921 f.
 – COM 919, 925
 – Container 927
 – Pfad 921
 AdsPath 937
 ADUser 953
 Advanced Function 1091, 1100
 ADWS 948
 AgentPC 915
 Akte X 914
 Aktivierung 625
 Aktivität 506, 510
 Alias 43, 56, 233, 645
 Aliaseigenschaft 107, 112
 AliasInfo 57
 AllNodes 561
 AllowClobber 25, 592
 AllowEmptyCollection 1100
 AllowEmptyString 1100
 AllowNull 1100
 AllSigned 136
 Alvin Kersh 914
 Änderungshistorie 720
 –and 115
 –force 51
 –or 115
 Animation 1087
 Ankerelement 167
 Anwendungspool 1005, 1007
 Anzeigesprache 546
 Apache 1056
 AppDomain 380, 749
 AppendChild(). 709
 Apple Software Package 22
 AppLockerPolicy 825 f.
 appSettings 307
 Architektur 31
 Args 133, 151
 args 253
 Array 173, 176 ff., 1202
 ArrayList 176
 AsJob 482, 503
 ASP.NET 394, 1059
 ASP.NET Core 324
 Assembly 364, 366, 378, 581, 587,
 594, 655, 1128, 1157
 – verbreiten 1200
 AssocClass 839
 ASSOCIATORS OF 410
 Assoziation 402
 – WMI 398, 402
 Asynchronous 874

Attribut 1185, 1190, 1201
 – indiziert 1202
 Audio 373
 Aufgabe
 – geplant 489
 Aufzählung 385
 Ausdruck 64
 – regulär 165
 Ausdruckauflösung 158
 Ausdrucksmodus 64
 Ausführungsrichtlinie 135
 Ausgabe
 – mehrspaltig 214
 – unterdrücken 227
 Ausgabeobjekt 1131
 Auslagerungsdatei 625
 Authentifizierung 454, 932, 975
 AuthorizationRuleCollection 891 f.
 AutoUpdate 786
 Azure 324
 Azure Cloud Shell 310
 Azure Container Registry 1056
 Azure Container Service *siehe* ACS

B

Background Intelligent Transfer
 Service *siehe* BITS
 BackgroundColor 152, 289, 309
 Backspace 160
 Backup 689, 691, 748
 Backup-GPO 983
 Backup-SqlDatabase 746, 748
 Base 937, 959
 bash 277, 310, 324, 345, 347
 Basisauthentifizierung 241
 Basisimage 1066, 1068
 Basisklasse 719
 Batterie 795
 Bedingung 188
 Beep 160
 Beep() 375
 Befehl
 – extern 43, 65
 Befehls-Add-On 76
 Befehlseingabefenster 26
 Befehlsgeschichte 633
 Befehlsmodus 64
 Befehlsobjekt 724
 Begin 667
 BeginProcessing() 1122, 1126
 Benutzer 393, 942, 1186, 1189
 – Active Directory 928
 – anlegen 931
 – lokal 989
 – löschen 933
 – umbenennen 933

– verschieben 934
 Benutzer-DSN 767
 Benutzerabmeldung 638
 Benutzeranmeldung 638
 Benutzerdaten lesen 962
 Benutzereingabe 451
 Benutzergruppe 974
 Benutzerkennwort 932
 Benutzerkontensteuerung *siehe*
 UAC
 Benutzerkonto 327, 962, 1205
 Benutzername 454
 Benutzerschnittstelle 392
 Berechnung 122
 Best Practice 604, 911
 Betriebssystembasis-Image 1035
 Bezeichner 1199
 Beziehung 1189
 Bibliothek 1196
 Big Endian 937
 Bild 618
 Bildschirmschoner 447, 915
 Bindung
 – ADSI 921
 – serverlos 922
 – WMI 417
 Bing 1082
 Binärdatei 713
 Binärmodul 1165
 BIOS 393
 BitLocker
 – Überblick 693
 Bitmap 665 f.
 BITS 45, 873, 876
 Blatt 919
 BMC 392
 Boot-Konfiguration 393
 Bootstrap 383
 break 182, 184, 194, 197
 bxor 654, 991
 Bypass 136
 ByPropertyName 92 f.
 Byte 154
 ByVal 92 f.
 BZIP2 675

C

C# 31, 141, 445, 447, 1091, 1119 f.,
 1126, 1193
 C++ 1197 f.
 C++/CLI 1119
 CAB 80
 Canvas 1084
 Carriage Return 160
 cat 347, 349
 CATID 411

- CD 378
- Certificate
 - Zertifikat 903
- ChangeAccess 683
- Checkpoint-Computer 789
- Checkpoint-VM 1010, 1026
- Children 918
- ChildSession 258
- CHKDSK 394
- Chkdisk() 435
- chmod 347
- Chocolatey 806 f.
- chown 352
- Chrome 553, 806
- CIL 37, 1195
- CIM 33, 392, 395
 - Repository 403
- CIM Explorer 318
- CIM Query Language *siehe* CQL
- CimClass 416 f., 419, 428
- CimClassProperties 428
- CimInstance 416 f., 419, 428 f.
- CimInstanceProperties 428
- CimProperty 428
- Cisco 392
- City 954 f.
- class 204
- ClassCreationEvent 532
- ClassDeletionEvent 532
- ClassModificationEvent 532
- Clear-BitLockerAutoUnlock-Funktion 694
- Clear-Content 698
- Clear-DnsClientCache 852
- Clear-EventLog 242, 879
- Clear-History 634
- Clear-Host 196, 633
- Clear-Item 645
- Clear-RecycleBin 664
- Clear-Variable 150
- ClearCase 140
- Click 1085
- Clipboard *siehe* Zwischenablage
- CliXml 709
- Close() 1084
- CLR 11, 37, 1169, 1195
- cmd.exe 88
- Cmdlet 1154
- Cmdlet Help Editor 1157
- CmdletBinding 1100, 1104
- cn 929
- Codeausschnitt 284, 302
- Codeeigenschaft 107, 112
- Color 797
- COM 40, 387, 664, 1198
 - Kategorie 411
 - Klasse 389
 - Komponente 393
 - Moniker 921
 - Sicherheit 407
- Comma-Separated Values 701
- Command Line Event Consumer 409
- Command Mode 64
- Commandlet 3, 43, 55, 65, 69, 89, 239
 - binär 1119
 - erstellen 1089, 1091, 1119
 - Klasse 1121
 - Konvention 1144, 1161
 - Moniker
 - Provider 234
 - Proxy 1109
 - Verkettung 1142
- CommandNotFoundException 202
- CommitChanges() 372, 916, 925 f., 932
- Common Information Model *siehe* CIM
- Common Intermediate Language *siehe* CIL
- Common Language Runtime *siehe* CLR
- Common Language Specification 1195
- Common Management Information Protocol 392
- Common Parameter 49
- Common Type System 1195
- compare 126
- Compare-Object 125 f.
- Compare-VM 1010, 1028
- Complete-BITSTransfer 874
- Complete-Transaction 328, 467 f., 470
- Component Object Model 31
- Component Object Model *siehe* COM
- Compress-Archive 674
- Computer 401, 942, 1189
- Computergruppe 307
- ComputerInfo 777
- ComputerName 133
- Computername 339, 784
- ComputerName 797
- Computerrichtlinie 464
- Computerverwaltung 777, 833
- configuration 554
- ConfigurationData 561
- ConfigurationID 571
- ConfigurationNamingContext 951
- Confirm 50
- confirm 51
- Confirm 52
- confirm 798, 847, 959, 962, 1104, 1106, 1154
- ConfirmPreference 52, 1106
- Connect-VMNetworkAdapter 1011
- Connection 722
- Console.WriteLine() 1149, 1162
- ConsolePaneBackgroundColor 289
- Container 927, 983, 1039, 1049
- Container-Klasse 919
- continue 51
- Continue 182, 184
- continue 194
- Continue 197, 199
- Convert-Html 713
- Convert-String 702
- Convert-VHD 1011, 1022, 1025
- Convert-Xml 711
- ConvertFrom-String 703
- ConvertFrom-StringData 544
- ConvertTo-ContainerImage 1066
- ConvertTo-CSV 701
- ConvertTo-DataTemplate 1087
- ConvertTo-SecureString 456, 975
- ConvertTo-WebApplication 1007
- ConvertTo-XML 711
- copy 657
- Copy-ContainerFile 1063
- Copy-GPO 982
- Copy-Item 197, 258, 645, 657, 660 f., 772, 904
- Copy-NetFirewallRule 855
- Copy-ToZip 675
- Copy-VMFile 1029
- Copy/Paste 275
- CORBA 1198
- Count 95, 175, 194
- Country 955
- CPU 127
- CQL 410, 425
- Create() 436, 1110
- CreateCommand() 724
- CreateElement() 709
- CreateInstance() 796
- CreateObject() 390
- CreationTime 665, 1190
- Creators Update 24, 70
- Credential 975
- Credentials 957
- CSV 62, 451, 698, 700, 775, 971, 1002
- CSV-Datei 127
- CultureInfo 1131
- Cursor 719 f.
- CustomerId 1108
- CVS 140

D

- Dana Scully 914
- Data Source Name *siehe* DSN
- DataReader 718 ff., 724, 726 ff.
- DataRow 112 f.
- DataSet 718 f., 726, 728 ff.
- DataTable 728 f.
- Datei 209, 394, 1186
 - Eigenschaft 653, 664
 - kopieren 657
 - löschen 45
 - Rechte 394
 - verschieben 657
- Dateiname 43
- Dateinamenerweiterung 127, 129
- Dateisystem 3, 659, 889, 1189, 1200
- Dateisystemfreigabe 553, 677
- Dateisystemkatalog 663
- Dateisystemstruktur 658
- Dateiversionsverlauf 688
- Datenabfrage 411
- Datenbank 403, 620, 715
- Datenbankmanagementsystem 722
- Datenbankverbindung 722
- Datenbankzeile 112
- Datenbankzugriff 715
- Datenbereich 543
- Datenbindung 1087
- Datendatei 543
- Datenmenge 233
- Datenquelle 765 f.
- Datenquellensteuerelement 718
- Datentyp 144, 151, 177, 923, 1142, 1201
 - .NET 89
 - PowerShell 144
 - WMI 397
- Datenzugriff 723
- DateTime 98, 101, 103, 374
- Datum 171
- Day 98
- DB2 717, 765
- dBase 765
- DbCommand 724
- DbDataReader 726
- DBG 465
- DbProviderFactories 717
- DCOM 241, 390, 403, 419, 786, 1137, 1150
 - Konfiguration 393
- dcpromo 975
- DDL 410
- Debug 50, 52
- debug 1149
- Debug-Modus 465
- Debugger 29
- Debugging 3, 28, 302, 465, 513
- DebugPreference 52, 1149
- Decimal 154
- Deep Throat 914
- Default Domain Policy 987
- DefaultNamingContext 951
- Deinstallation 803
- Delete() 1190
- Deleting 797
- Delimiter 698
- Deployment 1200
- Deployment Image Servicing and Management *siehe* DISM
- Description 929, 955
- DESCRIPTION 1108
- Description 1115
- Deserialisierung 250
- Desired State Configuration *siehe* DSC
- Desktop 393
- Desktop Management Task Force *siehe* DMTF
- Desktop-Anwendungen 1196
- Destruktor 1186, 1202
- Deutsche Telekom 475
- Dezimalzahl 154
- DHCP 843 f., 846
- Diagnose *siehe* Fehlersuche
- Dialogfenster 454, 1073
- diff 126
- Digest 241
- dir 351
- Directory 889
- Directory Management Objects 978
- DirectoryEntry 112 f., 147, 369, 917 ff., 923 ff., 928, 931, 934
- DirectoryInfo 104, 116, 441, 653, 889, 1190 f.
- DirectorySearcher 147, 938
- DirectorySecurity 891
- DirectoryString 929
- Disable-ComputerRestore 789
- Disable-JobTrigger 493
- Disable-Mailbox 994
- Disable-NetFirewallRule 855
- Disable-PnpDevice 795
- Disable-PSRemoting 247
- Disable-PSSessionConfiguration 257, 259
- Disable-VMIntegrationService 1014
- Disable-WindowsOptionalFeature 820, 822
- Disk Quotas 394
- DISM 820
- Dismount-VHD 1022
- DisplayName 955
- Distinguished Name 921, 926, 929, 951, 955
- Distributed COM *siehe* DCOM
- Distributed Component Object Model *siehe* DCOM
- Distributed File System 394
- Distributed Managements Objects *siehe* DMO
- DML 410
- DMO 749
- DMTF 241, 392
- DNS 269, 851, 976, 1038 f.
- DNS-Client 848
- DNS-Konfigurationseinstellungen
 - per WMI abfragen 849
- DNS-Server 394, 846, 850
- DnsClient 845
- DnsClient 846, 848
- DnsClient 852
- do 182
- Docker 324, 1035, 1038, 1041, 1049, 1051, 1053 f., 1068
- Docker Compose File 1059
- Docker Hub 1056
- Docker-Image 1056
- docker.exe 1049
- Dockerfile 1063, 1068
- DockPanel 1084
- Dokument 697
- Dokumentation
 - .NET 83
 - Active Directory 931
- Dollarzeichen 122, 158
- Domain 855
- Domain Controller 977
- Domäne 784, 922, 978, 1186, 1189
 - Beitritt 269, 785
 - hinzufügen 785
- DOS 3
- Dot Sourcing 62, 134 f., 474, 581, 1093 f., 1097
- DotNetTypes.Format.ps1xml 209, 217
- Double 154
- DownloadString() 371, 863
- DriveInfo 374
- Driver 768
- DriveType 384
- Druckauftrag 796
 - löschen 796
- Drucker 209, 228, 393, 797
 - verwalten 795 f.
- Druckerport 796
- Druckerverwaltung 795 f., 843
- DSC 323, 549, 555
 - Linux 323
- DSC Pull Server 566, 571

DSN 765, 767f.
DuplexingMode 797
DVD 378, 1028, 1032

E

E-Mail 862, 869
– Adresse 863
– EmailEvent 413
– senden 861
echo 64
Edit-NanoServerImage 1039
Eigenschaft 107
Eigenschaftenzwischenspeicher
 916
Eigenschaftssatz 107, 109
Eingabe 451
Eingabeaufforderung 635 f.
Eingabedialog 453
Eingabemaske 1076
Eingabeobjekt 1139
Eingabesteuerelement 1087
Eingabeunterstützung 283, 302
Einzelschrittmodus 460
Elevated 137f., 833, 885, 1056
Else 188
Emacs 130, 277
EmailAddress 955
EmailEvent 413
Enable-BitLocker 695
Enable-ComputerRestore 789
Enable-JobTrigger 493
Enable-NetFirewallRule 855, 861
Enable-ODBCPerfCounter 765
Enable-PnpDevice 795
Enable-PSRemoting 245, 283, 797,
 860
Enable-PSSessionConfiguration
 257, 259
Enable-VMIntegrationService
 1014
Enable-WindowsOptionalFeature
 820, 822
Encoding 698
End 667
EndProcessing() 1122, 1126
endregion 291
Enter-PSSession 247f., 257, 262,
 355, 465, 1041
Enum 385
EnumerateCollection 1134
Enumerationsklasse 384
env 347, 782
Environment 551
Ereignis 396, 1185, 1202
– PowerShell 531, 540
– WMI 408, 531

Ereignisabfrage 412
Ereigniskonsumment 408 f.
– permanent 408
– temporär 408
Ereignisprotokoll 127, 328, 393 f.,
 404, 410, 463, 553, 877
– Überwachung 413, 533
Ereignisprovider 408
Ereignissystem 531
Error 151, 202, 797
ErrorAction 50 ff., 199, 201, 660
ErrorActionPreference 52, 152, 201,
 963
ErrorBackgroundColor 17
ErrorRecord 197, 201, 203
ErrorVariable 51, 201
Ethernet 845
ETS 89, 107, 112, 427, 920
Event *siehe* Ereignis
EventConsumer 396
EventViewerConsumer 409
EXAMPLE 1108
Example 1115
Exception 197, 202f., 1145, 1151
Exchange Management Shell 621,
 993
Exchange Server 83, 394, 404, 993
ExecuteNonQuery() 724
ExecuteReader() 724, 726
ExecuteRow() 724
ExecuteScalar() 724
ExecutionPolicy 14 ff.
EXIF 665
Exists() 924
exit 182
Exit-PSSession 249, 257
Expand-Archive 674
explorer.exe 689
Export-Alias 62
Export-CliXml 451, 709
Export-Console 583, 1129
Export-Counter 883
Export-CSV 116, 339, 451, 701
Export-ModuleMember 588
Export-PfxCertificate 905
Export-VM 1011, 1027
Export-VMSnapshot 1026
Express 742
Expression 212
Expression Mode 64
Extended Reflection 89
Extended Type System *siehe* ETS
Extensible Application Markup
 Language *siehe* XAML
Extrinsic Event 408

F

facsimileTelephoneNumber 955
Failover Cluster 604
false 46, 51, 146, 151
Fax 955
FBI 914, 934
Feature 809
FeatureOperationResult 815
Fehler 51
Fehlerbehandlung 197, 1197
Fehlerklasse 182, 203
Fehlermeldung 46
Fehlersuche 459, 1146
Fehlertext 182
Fernaufruf 242
Fernausführung 133, 241
– Hintergrundauftrag 485
Fernverwaltung 241
Fernzugriff 241
Festplatte
– virtuell 1022
Festplattenverschlüsselung 693
Fibre-Channel 1010
Field 108
File 889, 1188
File History 688
FileInfo 104, 116, 441, 653f., 889,
 1190 f.
FileInformation 828
FileSecurity 889, 891
FileSystem 618, 645
FileSystemAccessRule 892
FileSystemInfo 1190
FileSystemObject 1188
FileSystemRights 385
FileSystemWatcher 539
FileVersionInfo 807
filter 651
Find() 918
Find-Module 335, 593 ff.
Find-Package 806
Firefox 553
Firewall 350, 625
Firewall-Regel 858
First 97
fish 324
For 184, 186
Force 49
force 51
Force 652, 959
ForEach 100, 121, 127f., 182, 186
foreach 511
ForEach 918, 1135
ForEach-Object 99 f., 109, 121f., 126,
 175, 187, 193, 461, 510, 697, 1134
ForegroundColor 74

- Forest 978
 - Form Feed 160
 - Format 212
 - Format-List 87, 210, 891
 - Format-Table 98, 109, 127, 210, 212, 218 f., 891
 - Format-Wide 209 f., 214 f.
 - Format-Xml 706 f.
 - Formatkennzeichner 222
 - Fortschrittsanzeige 230
 - Fox Mulder 914
 - FoxPro 765
 - Framework Class Library 363
 - Freigabe 686
 - FTP 339
 - FullAccess 683
 - FullName 1190
 - function 43, 182, 190, 196, 233
 - Funktion 43, 190, 233
 - eingebaut 196
 - fortgeschritten 1100
- G**
- GAC 378 f.
 - Ganzzahl 154
 - Gast 1009
 - Gateway 846
 - GeneralizedTime 929
 - Geplante Aufgabe 489
 - Gesamtstruktur 978 f.
 - Geschäftsanwendung 1160
 - Get-Acl 885, 889, 891, 901
 - Get-ADComputer 957
 - Get-ADDomain 979
 - Get-ADDomainController 979
 - Get-ADForest 979
 - Get-ADGroup 957, 974
 - Get-ADGroupMember 957, 974
 - Get-ADObject 47, 913, 944, 957ff.
 - Get-ADOptionalFeature 979
 - Get-ADOrganizationalUnit 957, 961
 - Get-ADPrincipalGroupMembership 974 f.
 - Get-ADRootDSE 979
 - Get-ADUser 957, 962 f.
 - Get-Alias 57
 - Get-AppLockerFileInformation 825
 - Get-AppLockerPolicy 825 f.
 - Get-AuthenticodeSignature 478
 - Get-BitLockerVolume 694 f.
 - Get-BITSTransfer 874 f.
 - Get-BPAModel 911
 - Get-BPAResult 911
 - Get-CDRomDrive 619, 793
 - Get-ChildItem 44 ff., 48, 87 f., 116, 120 f., 127 f., 233, 651, 674, 771, 801
 - BitLocker 695
 - Get-CimAssociatedInstance 418
 - Get-CimClass 391, 418, 425
 - Get-CimInstance 327, 391, 413, 418 f., 421 f., 436, 794, 797
 - Get-Clipboard 328, 456 f., 597, 614
 - Get-Command 55, 69 ff., 600
 - Get-ComputerInfo 777
 - Get-ComputerInfo 784
 - Get-ComputerInfo 1131
 - Get-Computersname 1124
 - Get-ComputerRestorePoint 789
 - Get-Container 1065 f.
 - Get-ContainerImage 1056
 - Get-ContainerImage 1058, 1066, 1070
 - Get-Content 645, 660, 697, 713, 775, 800, 1088
 - Get-Counter 242, 327, 881 f.
 - Get-Credential 75, 265, 454, 628, 834, 862
 - Get-Culture 163
 - Get-DataRow 620, 736
 - Get-DataTable 620, 736
 - Get-Date 101, 171 f.
 - Get-DHCPsServer 25, 844
 - Get-DirectoryChildren 619
 - Get-DirectoryEntry 158, 619, 1144
 - Get-DirSize 1091
 - Get-Disk 646, 648, 793, 1133 f., 1137, 1139, 1141 f., 1150
 - Get-DisplaySetting 795
 - Get-DnsClient 848 f.
 - Get-DnsClient-Funktion
 - Beispiel 849
 - Get-DnsClientCache 852
 - Get-DnsClientServerAddress 848 f.
 - Get-DomainController 25 f., 922
 - Get-DSCConfiguration 565
 - Get-DVDDrive 623
 - Get-Event 535
 - Get-EventLog 10, 127, 242, 328, 877 f.
 - Get-ExCommand 993
 - Get-ExecutionPolicy 14
 - Get-ExportedType 655
 - Get-Filecatalog 83
 - Get-FileHash 666 f.
 - Get-FileVersionInfo 654, 807
 - Get-FirewallRule-Funktion 856
 - Get-FloppyDrive 623
 - Get-Flug 1160
 - Get-Flugziele 1160
 - Get-Font 784
 - Get-FreeDiskSpace 648, 650
 - Get-GPIInheritance 986
 - Get-GPO 982 f.
 - Get-GPOReport 985
 - Get-GPPermissions 988
 - Get-GPPrefRegistryValue 987
 - Get-GPRegistryValue 988
 - Get-GPResultantSetOfPolicy 985
 - Get-GPStarterGPO 982
 - Get-Help 71 ff., 76, 78 f., 82, 141, 339, 342, 1091, 1108, 1116, 1156
 - Get-History 342, 633
 - Get-Host 342, 634
 - Get-HotFix 242
 - Get-Item 664, 771, 1000, 1005
 - Get-ItemProperty 665, 771
 - Get-Job 481, 483
 - Get-JobTrigger 494
 - Get-Keyboard 794
 - Get-LDAPChildren 945, 1096
 - Get-LDAPObject 945, 1096 f.
 - Get-LocalGroupMember 990
 - Get-LocalUser 327, 989
 - Get-Location 56, 645
 - Get-LogicalDiskInventory 646
 - Get-LoremIpsum 658
 - Get-Mailbox 993 f.
 - Get-Mailboxdatabase 993
 - Get-Member 102, 105, 107 f., 117, 126, 374, 383, 428, 920, 1134
 - Get-Members 934
 - Get-MemoryDevice 619, 793
 - Get-Methode 108
 - Get-Module 213, 587, 598, 1165
 - Get-MountPoint 25
 - Get-MPCComputerStatus 791
 - Get-MultiTouchMaximum 795
 - Get-NanoServerPackage 1039
 - Get-NetAdapter 845
 - Get-NetAdapterBinding 845
 - Get-NetFirewallAddressFilter 855
 - Get-NetFirewallAddressFilter-Funktion 857
 - Get-NetFirewallApplicationFilter 855
 - Get-NetFirewallInterfaceFilter 855
 - Get-NetFirewallInterfaceTypeFilter 855
 - Get-NetFirewallPortFilter 855
 - Get-NetFirewallProfile 855 f.
 - Get-NetFirewallRule 855, 857 f.
 - Get-NetIPInterface 846
 - Get-NetworkAdapter 619, 794
 - Get-ODBCDriver 765
 - Get-ODBCDSN 765
 - Get-OSVersion 779
 - Get-Package 806 f.
 - Get-PackageProvider 805

- Get-PackageSource 806
 - Get-Passagier 1160
 - Get-PipelineInfo 102f., 116, 1143
 - Get-PnpDevice 795
 - Get-PnpDeviceProperty 795
 - Get-PointingDevice 794
 - Get-PowerShellDataSource 1087
 - Get-Printer 797
 - Get-PrintJob 797
 - Get-Process 26, 44f., 47f., 56, 62, 87, 92, 99, 101, 105, 107, 116f., 122, 127, 219, 228, 242, 339, 347, 461, 831, 1142, 1144
 - Get-Processor 619, 793f.
 - Get-PSBreakpoint 466
 - Get-PSDrive 646
 - Get-PSProvider 235
 - Get-PSRepository 595
 - Get-PSSession 257
 - Get-PSSessionConfiguration 257, 259, 630
 - Get-PSSnapIn 584
 - Get-PswaAuthorizationRule 307
 - Get-Random 155
 - Get-ReparsePoint 673
 - Get-Service 74f., 92, 95, 101, 116, 133, 228, 242, 339, 341, 358, 837f., 1109f.
 - Get-SHA1 667
 - Get-ShortPath 656
 - Get-SmbShare 678, 683
 - Get-SmbShareAccess 684
 - Get-SoundDevice 793
 - Get-SqlData 754
 - Get-Storagegroup 993
 - Get-Tapedrive 793
 - Get-TargetResource 579
 - Get-TerminalSession 25
 - Get-TraceSource 462
 - Get-Transaction 328, 467, 470
 - Get-Unique 119
 - Get-Uptime 340, 359, 779
 - Get-USB 794
 - Get-USBController 619, 794
 - Get-Variable 144, 147, 152
 - Get-VHD 1022
 - Get-VideoController 619, 793
 - Get-VirtualHardDisk 623
 - Get-VM 1011f., 1022
 - Get-VMBIOS-VM 1013
 - Get-VMBuildScript 1032f.
 - Get-VMHost 623, 1011
 - Get-VMMemory 1013
 - Get-VMProcessor 1013
 - Get-VMSnapshot 1026
 - Get-VMSummary 1033
 - Get-VMThumbnail 1032f.
 - Get-WBBackupSet 691
 - Get-WBPolicy 690
 - Get-WBSummary 690
 - Get-WebApplication 1000
 - Get-WebitemState 1007
 - Get-Website 1000, 1005
 - Get-WebvirtualDirectory 1000
 - Get-WindowsCapability 821
 - Get-WindowsEdition 779
 - Get-WindowsFeature 808, 811, 813
 - Get-WindowsOptionalFeature 820
 - Get-WinEvent 242
 - Get-WmiObject 39, 242, 327, 378, 391, 418f., 421f., 424, 650, 793, 799, 838, 840, 843, 881, 915
 - Get-xDscOperation 573
 - GetAccessRules() 891ff.
 - GetDrives() 373
 - GetFactoryClasses() 717
 - GetLongDateString() 101
 - GetLongTimeString() 101
 - GetNames() 385
 - GetObject() 390
 - GetRelated() 839
 - GetShortDateString() 101
 - GetShortTimeString() 101
 - GetTempName() 388
 - Getter 108, 1201
 - GetTimestamp() 359
 - GetType() 102, 151, 226
 - Gigabyte 154
 - Git 140
 - GitHub 359
 - Gitternetz 1084
 - GivenName 954f.
 - Gleichheitszeichen 179
 - global 149, 1084
 - Global Assembly Cache *siehe* GAC
 - Global Unique Identifier 926, 1198
 - GlobalSign 475
 - gm 126
 - Google 1082
 - GPMC 981
 - Grafikkarte 393, 420
 - Grant-SmbShareAccess 684
 - Gravis 53f., 96, 142, 160
 - grep 324
 - Grid 1084f.
 - GridView 215
 - Group 120, 551, 919
 - GROUP BY 410, 532
 - Group-Object 120, 126f., 878
 - Gruppe 942, 957, 971
 - Active Directory 928
 - anlegen 935
 - auflisten 934
 - lokal 989
 - Mitglied aufnehmen 935
 - Gruppenmitglieder 957
 - Gruppenmitgliedschaft 936
 - Gruppenrichtlinie 464, 604, 638, 981ff., 986
 - Vererbung 986
 - Gruppierung 120
 - GUID 650
 - Gültigkeitsbereich 145, 149
 - GZIP 675
- ## H
- Haltepunkt 28, 286, 465
 - Hardlink 671
 - Hardware 393, 619
 - Hardwareverwaltung 793
 - Hash-Tabelle 177f., 370, 436
 - Hashtable 177, 179, 370, 561, 579
 - Hashwert 663
 - HAVING 410, 532
 - Heimordner 151
 - Help-Info 80
 - HelpMessage 1100
 - Herausgeber 476, 479
 - Here-String 156
 - Herunterfahren 785
 - Hexadezimalzahl 154
 - hidden 206
 - Hilfe 69
 - Hilfetext 78
 - Hintergrundauftrag 481
 - Hintergrunddatentransfer 873
 - Hintergrundübertragungsdienst 45
 - History 633
 - Hit Refresh 324
 - HKCU 233
 - HKEY_CURRENT_USER 772
 - HKEY_LOCAL_MACHINE 772
 - HKLM 233
 - Home 151
 - HomeDirectory 955
 - HomeDrive 955
 - Host 152, 289, 309, 634, 1009
 - Hosting 1177
 - hostname.exe 784
 - Hotfix 393
 - HTML 713, 1004, 1076
 - HtmlWebResponseObject 864
 - HTTP 241, 339, 566
 - HTTPS 241, 339
 - Hyper-V 267, 603, 623, 1009, 1029, 1035, 1038f.
 - Überblick 1009
 - Hyper-V-Container *siehe* Docker

Hyper-V-Integrationsdienste 1014
 Hyper-V-Modul
 – Überblick 1010
 Hypervisor 1009

I

IADs 916 f., 919
 IADsComputer 919
 IADsContainer 917, 919
 IADsGroup 919
 IADsUser 919, 932
 idempotent 549 f.
 Identity 886
 IdentityReference 894
 Identität 1006
 IDL 404
 IEnumerable 918
 if 182, 188
 IIS 235, 304, 922, 1038 f.
 – 8.0 997
 – Internet Information Server 997
 – Nano 1046
 IIS Management Service 1046
 IIS-Anwendung 1006
 IISAdmin 839
 IISAdministration 997, 1046
 IISpy 383
 Impersonifizierung 454, 924
 – WMI 407
 Implizites Remoting 261
 Import-Alias 62
 Import-CliXml 451, 634, 710
 Import-Counter 883
 Import-CSV 127, 451, 653, 700,
 971
 Import-GPO 983
 Import-INIFile 704
 Import-Module 513, 588, 594, 600,
 1171 f.
 Import-PSSession 262
 Import-VM 1011, 1027 f.
 IncludeUserName 832
 Index 173
 Indexer 1202
 Informix 717, 765
 Ingres 717
 Inheritance Flags 887
 INI-Datei 704
 inlinescript 508
 Innertext 709
 Input 151
 InputBox 379 f., 453
 Inputbox 1075
 InputBox() 453, 1075
 InputObject 93, 383, 1144, 1146
 inquire 51, 199

Install-ADDSDomainController 977
 Install-ADDSTForest 977
 Install-Module 25, 311, 335, 591,
 593, 595
 Install-Package 381
 Install-PswaWebApplication 306
 Install-WindowsFeature 306, 567
 Installation 802
 Installationsordner 5, 112, 151
 Installationstechnologie 802
 installutil.exe 1123, 1128
 InstanceCreationEvent 532 f., 535
 InstanceDeletionEvent 396, 408,
 532
 InstanceModificationEvent 408,
 532
 Instanz 370, 1186
 Instanziierung 1187
 Instanzmitglied 1202
 int 145
 Int32 145
 Int64 154
 INTEGER 929
 Integrated Scripting Environment
 26
 Integrated Scripting Environment
siehe ISE
 IntelliSense 26, 44, 299
 Interface 1188
 Interface Definition Language 404
 InternalHost 634
 InternalHostUserInterface 452
 International .NET Association XXX
 Internet Control Message Protocol
 853
 Internet Information Server 394
 Internet Information Services
 393 f., 566, 574, 822, 839, 922,
 997
 Interpretermodus 277
 IntervallTimerInstruction 408
 Intrinsic Event 408
 InvalidOperationException 918
 Invoke-BPAModel 911
 Invoke-CimMethod 327, 418, 436
 Invoke-Command 247, 249, 251,
 253 f., 257, 264, 267, 465, 482,
 1041
 Invoke-ContainerImage 1057 f.
 Invoke-DbCommand 736
 Invoke-DBMaint 751
 Invoke-Expression 182
 Invoke-History 633
 Invoke-Query 754
 Invoke-SqlBackup 761
 Invoke-SqlCmd 740, 742, 746 ff.,
 759, 761

Invoke-SqlCommand 620
 Invoke-WebRequest 339, 864 f.,
 870
 Invoke-WmiMethod 327, 418, 435
 InvokeMethod() 417
 IP
 – Adresse 125, 147, 269, 411, 553,
 843, 846
 – Konfiguration 411
 – Routing 394
 IPAddress 147, 846
 ipconfig 65, 95
 IPHostEntry 851
 IRQ 393
 Is64BitOperatingSystem 778
 Is64BitProcess 778
 ISA 410
 IsCoreCLR 340
 ISE 130, 230, 282, 287, 1041, 1177
 – Debugging 286
 – Integrated Scripting Environment
 282
 ISE Steroids 310
 IsePack 618
 IsLinux 340
 IsMacOS 340
 ISO 1009, 1016, 1019, 1028
 IsWindows 340
 Item() 917

J

Java 1198
 JavaScript 383
 JEA 627
 Jeffrey Snover 131
 Job-Trigger 493
 – zeitgesteuerte Jobs 493
 John Doggett 914
 Join() 164
 Join-String 164
 JPEG 665
 JScript .NET 445
 JSON 870
 Junction 672
 Junction Point 671 f.
 Just-In-Time-Compiler 1195

K

Kennwort 454 f., 915, 932
 Kerberos 241, 407
 Kill() 100
 Kilobyte 154
 Klammer 47
 – rund 167
 Klammeraffe 64

Klasse 144, 373, 397, 401, 441, 1186, 1188 f., 1201
 – .NET 204, 363, 1121, 1198, 1200
 – CIM 395
 – COM 367, 387
 – Commandlet 1122, 1141, 1163
 – PowerShell 204 f.
 – statisch 375
 – WMI 393 f.
 Klassendiagramm 1190 f.
 Klassenhierarchie 1190
 Klassenmitglied 373, 1202
 Klassenname 369
 Known Host 357
 Kommandomodus 277
 Kommandozeilenbefehl 65
 Kommentar 142
 Komponentenorientierung 1195 f.
 Komposition 1087
 Komprimierung 674 ff.
 Konstruktor 1186, 1202
 Konstruktorfunktion 368 ff., 387
 Kontakt 942
 Konvention 1161
 Kopieren/Einfügen 275
 Kosinus 374
 Kreuzzuweisung 181
 ksh 324

L

Label 212
 LastAccessTime 1190
 LastExitCode 151
 Laufwerk 233 f., 240, 646, 772
 – virtuell 1022
 LDAP 913, 920, 922, 937
 – Suchanfrage 917, 937
 – Suche 943
 LDAP-Query 938
 Leaf 927
 Least Privilege 627
 Leistung 881
 Leistungsindikator 881 f.
 Length 95, 441
 Limit-EventLog 242, 877, 879
 LinearGradientBrush 1085
 Linie 1190
 LINK 1108
 Linux 3 f., 8, 18, 21, 23, 37, 236, 323 ff., 330, 347, 355, 387, 1050, 1054, 1198
 – Container 1066, 1068
 – Dateisystem 351
 Linux Foundation 324
 Literal 222
 Little Endian 937

Lizenzierung 625
 Load-ContainerImage 1069
 LoadFrom() 378
 LoadWithPartialName() 378
 Log File Event Consumer 410
 Logarithmus 374
 Logoff 638
 Logon 638
 Lokalisierung 400, 546
 Loopback 244
 ls 347

M

Machine.config 717
 MachineName 243, 1126
 MacOS 3 f., 8, 18, 37, 236
 macOS 323 ff., 330, 347
 MacOS 387, 1198
 macOS 351
 MailAddress 863
 MailMessage 861
 makecert.exe 476
 MAML 79, 1156
 man 347
 Manage-Bde 693
 Managed Code 916
 Managed Object 392
 Managed Object Format 404
 Managed Provider 715, 937
 Management Infrastructure API 416 f.
 ManagementBaseObject 416
 ManagementClass 112 f., 147, 416 f., 419, 422, 427, 1133
 ManagementEventWatcher 533
 ManagementObject 113, 147, 416 f., 419, 422, 427, 429 f., 435, 839, 1133 f.
 ManagementObjectCollection 427, 1133 f.
 ManagementObjectSearcher 147, 422, 1133
 ManagementScope 533
 Mandatory 1100, 1137
 Manifest 589
 Manifestmodul 1165
 Maschinencode 1195
 Match 113
 MaximumDriveCount 240
 MaximumErrorCount 151
 maxSessionsAllowedPerUser 307
 md 660, 666, 775
 measure 122
 Measure-Command 461
 Measure-Object 122, 126 f.
 Measure-VM 1011
 Megabyte 154
 Mehrsprachigkeit 546
 Mercurial 140
 Merge-VHD 1022
 Message 197
 MessageBox 454, 1073
 MessageBoxButtons 1074
 MessageBoxDefaultButton 1074
 MessageBoxIcon 1074
 Metamodell 402
 Metaobjekt 926
 Method *siehe* Methode
 Methode 107, 1185, 1190, 1202
 – Getter 1201
 – Setter 1201
 Microsoft Access 730
 – Treiber 721
 Microsoft Certified Solution Developer XXIX
 Microsoft Command Shell 31
 Microsoft Developer Network 83, 365
 Microsoft Developer Network *siehe* MSDN
 Microsoft Excel 971
 Microsoft Exchange 976
 Microsoft Exchange Server 553, 621, 993
 Microsoft Office 394
 Microsoft Outlook 836
 Microsoft Print Ticket XML 797
 Microsoft Shell 31
 Microsoft SQL Server 716, 740
 Microsoft System Center 413
 Microsoft Word 390
 Microsoft-Access 5
 Microsoft.ACE.OLEDB 721
 Microsoft.GroupPolicy 982
 Microsoft.Jet.OLEDB 721
 Microsoft.Management.Infrastructure.CimClass 428
 Microsoft.Management.Infrastructure.CimClassProperties 428
 Microsoft.Management.Infrastructure.CimInstance 428
 Microsoft.Management.Infrastructure.CimInstanceProperties 428
 Microsoft.Management.Infrastructure.CimProperty 428
 Microsoft.Update 786
 Microsoft.Vhd.PowerShell 1022
 Microsoft.VisualBasic 453, 1075
 Microsoft.VisualBasic.Interaction 390, 453, 1075
 Microsoft.Win32 801

- Microsoft.Win32.RegistryKey 771
 - Minute 98
 - Mitglied 1185
 - .NET 1201
 - statisch 1202
 - WMI 432
 - MMC 407
 - Modul 587, 596
 - Module Browser 592
 - Modulo 178
 - MOF 404, 559, 561
 - Monad 31
 - Monica Reyes 914
 - Moniker 921
 - Mono 1198
 - Month 98
 - more 196, 220
 - Most Valuable Professional XXIX
 - Mount-SpecialFolder 647
 - Mount-VHD 1022
 - move 657
 - Move-ADObject 957f.
 - Move-Item 645, 657, 666
 - Move-Mailbox 994
 - Move-VM 1011
 - MSCL 38
 - mscorlib.dll 1200
 - MSDN 179, 365, 1080
 - MSDN Library 83, 365
 - MSFT_Printer 797
 - MSFT_PrintJob 797
 - MSFT_SmbShare 683
 - MSFT_SmbShareAccessControl
Entry 684
 - MSFT_WUOperationsSession 786,
788, 1044
 - MSFT_WUSettings 786
 - MSFT_WUUpdate 786
 - MSH *siehe* Microsoft Shell
 - MSI 575, 799, 802 f., 1123
 - MTA 1079
 - Multithreading 1197
 - MySQL 717, 727, 740, 751, 1056
 - MySqlConnection 728
 - MySqlLib 740
- N**
- Nachkommastelle 154, 374
 - Name 92, 955
 - Namensauflösung 851
 - Namensraum 397, 399, 401 f., 889,
978, 1198
 - .NET 1198, 1200
 - ADSI 921
 - WMI 399, 402
 - Namensraumhierarchie 1199
 - Namespace-ID 921
 - NamespaceCreationEvent 532
 - NamespaceDeletionEvent 532
 - NamespaceModificationEvent 532
 - Nano Server 37, 321, 323, 1035,
1038
 - IIS 1046
 - Installation 1039
 - Paketinstallation 1039, 1044
 - NanoWbem 391
 - NativeObject 918, 920
 - Navigation 233
 - Navigation Provider 234
 - Navigationsbefehl 237
 - Navigationsmodell 645
 - Navigationsparadigma 233
 - Navigationsprovider 913
 - NetAdapter 845 f.
 - NetSecurity 855
 - NetSecurity-Modul
 - Überblick 854
 - Netsh 848, 850, 858 f.
 - Per PowerShell aufrufen 850
 - netstat 65
 - NetTCPIP 845 f.
 - Network Load Balancing 394
 - NetworkInterface 1131
 - Netzlaufwerk 393
 - Netzlaufwerksverbindung 394
 - Netzwerkadapter 1009
 - Netzwerkcenter 246
 - Netzwerkkarte 393, 411, 843, 1186
 - Geschwindigkeit 847
 - Netzwerkkartenprofil 860
 - Netzwerkkonfiguration 625, 843
 - Netzwerkmanagement 391
 - Netzwerkprofil
 - per PowerShell setzen 860
 - Netzwerkverbindung 394, 553,
845 f.
 - Neustart 269, 785, 815
 - New Line 160
 - new() 368, 370, 387
 - New-ADGroup 957, 974
 - New-ADObject 958
 - New-ADOrganizationalUnit 957, 961
 - New-ADUser 957, 962, 964
 - New-AppLockerPolicy 825, 828
 - New-Buchung 1160 f.
 - New-Button 73, 1083, 1085
 - New-CheckBox 1087
 - New-CimInstance 418, 436
 - New-CimSession 421
 - New-CimSessionOption 421
 - New-ComboBox 1087
 - New-Container 1065
 - New-DSCChecksum 572
 - New-Ellipse 1087
 - New-Event 540
 - New-EventLog 242, 877f.
 - New-FileCatalog 663
 - New-GLink 983
 - New-GPO 982
 - New-GPStarterGPO 982
 - New-Grid 1085
 - New-Guid 367
 - New-Hardlink 672
 - New-HardwareProfile 623
 - New-IISite 997, 1046
 - New-Image 1087
 - New-Int64Animation 1087
 - New-Item 234, 645, 669, 698, 771 f.,
775
 - New-ItemProperty 773, 775
 - New-JobTrigger 494, 496
 - New-Junction 672, 674
 - New-Label 1080, 1085
 - New-Line 1087
 - New-ListBox 1087
 - New-LocalUser 13
 - New-Mailbox 994
 - New-Mailboxdatabase 994
 - New-MediaElement 1087
 - New-Menu 1087
 - New-Module 587
 - New-NanoServerImage 1039 f.
 - New-NetFirewallRule 855, 858
 - New-NetIPAddress 846
 - New-Object 367ff., 375, 378, 387,
1104, 1203
 - New-PasswordBox 1085
 - New-ProgressBar 1087
 - New-PSDrive 240, 772, 801, 952
 - New-PSSession 245, 247, 257,
260 f., 265, 267, 631
 - New-PSSessionConfigurationFile
627
 - New-RadioButton 1087
 - New-Rectangle 1087
 - New-RichTextBox 1087
 - New-ScheduledJobOption 497
 - New-ScrollBar 1087
 - New-SelfSignedCertificate 476
 - New-Service 837, 841
 - New-Shortcut 670
 - New-Slider 1087
 - New-SmbShare 51, 683
 - New-StatusBar 1087
 - New-Storagegroup 994
 - New-Storyboard 1087
 - New-TextBlock 1087
 - New-TextBox 1085
 - New-TimeSpan 172
 - New-TreeView 1087

New-UrlShortcut 671
 New-VHD 1018, 1022 f.
 New-ViewBox 1087
 New-VirtualDVDDrive 623
 New-VirtualNetworkAdapter 623
 New-VM 623, 1011, 1016, 1033
 New-VMSwitch 1011
 New-WebApplication 1007
 New-WebAppPool 1005
 New-WebServiceProxy 868 ff.
 New-Website 997, 1001, 1046
 New-WebVirtualDirectory 1006
 New-Window 1085
 New-Zip 675
 NoAccess 683
 Node 709
 node.js 1056
 NoElement 120
 Non-Terminating Error 197, 1150, 1153
 NoProfile 474
 Northwind 742, 762
 Notation 1190
 – umgekehrt polnische 937
 Notepad 130
 NoteProperty 441, 704
 NOTES 1108
 Notes 1115
 Notizeigenschaft 107, 110, 117
 Novell 922
 Now 374
 Nslookup 848, 852
 NT Event Log Event Consumer 410
 NTAccount 890, 893
 NtAccount 894
 NTFS 404
 NTLM 241
 NTSecurityDescriptor 929
 Nuget 364, 381
 NuGet Package Manager 301
 null 123, 133, 187, 191, 227

O

Object 1192
 Object Linking and Embedding
 Database 726
 ObjectCategory 929, 941, 960
 ObjectClass 929, 941, 954 f., 960
 ObjectGUID 930, 955
 ObjectSecurity 889
 ObjectSecurityDescriptor 929
 ObjectSid 929
 ObjectVersion 930
 Object[] 194
 Objekt 703, 1162, 1185 f., 1188 f.
 – .NET 1078

– dynamisch 441, 1091
 – WMI 394
 Objekt-Pipeline 653
 Objektadapter 113, 426 f., 718
 Objektassoziation
 – WMI 402
 Objektbaum 1189
 Objektidentifikation
 – ADSI 921 f.
 Objektmenge 653
 Objektorientierte Programmierung
 siehe OOP
 Objektorientierung 88
 Objektorientierung *siehe* OO
 Objekttyp 1186
 OData 566
 ODBC 715 f., 765, 768
 Office 955
 OFS 151
 ogy 209
 OK 1074
 OKCancel 1074
 OLEDB 715 f., 726, 918
 – Provider 730, 918, 937
 OleDbCommand 724, 730
 OleDbConnection 722, 726, 730
 OleDbDataAdapter 730
 OMI 391, 420
 OMI *siehe* Open Management
 Infrastructure
 OneGet 804
 ONELEVEL 937
 OneLevel 959
 On_Click 1085
 OO 1185, 1195 f.
 OOP 1185
 OOP *siehe* COP
 Open Database Connectivity
 – Einstellung 394
 Open Management Infrastructure
 353
 – OMI 420
 Open Source 32
 Open() 722
 OpenSSH 353
 OpenView 413
 Operator 115, 164, 173, 178
 Optimize-VHD 1022
 Oracle 716, 727
 OracleCommand 724
 OracleConnection 722
 Ordner 45, 66, 237, 651, 656 f., 659
 – Dateisystem 394
 Ordnerstatistik 652
 Organisationseinheit 942, 971
 – anlegen 936

OSS
 – Open Source 32
 Out-Default 209, 217, 1180, 1182
 Out-File 209, 228
 Out-GridView 75, 209 f., 215 f.
 Out-Host 209, 219 f.
 Out-Null 209 f., 227
 Out-Printer 209, 228, 796
 Out-Speech 209, 230, 593
 Out-SqlScript 751
 OutBuffer 51, 91
 Outlook 119, 603, 624
 OUTPUTS 1108
 OutVariable 50, 124
ov siehe OutVariable

P

PackageManagement 590
 Page File 625
 Paketinstallation
 – Nano Server 1044
 Panel 1084
 PaperSize 797
 Papierkorb 664
 parallel 510
 Parameter 44 f., 93, 194, 1100
 PARAMETER 1108
 Parameter 1115, 1137, 1146
 – Abkürzung 48 f.
 – Skript 133
 Parameterliste 133
 ParentSession 258
 parsen 701
 Partition 957
 PascalCasing 1199
 PASH 323
 PassThru 124, 957
 passwd 349
 Pause 797
 PE 655
 PercentComplete 229
 Perforce 140
 Performance Counter Provider 881
 Performance Monitor 393, 404
 PERL 141
 Persistenz 512
 Pester 1175
 Petabyte 154
 Pfad 401
 – ADSI 921
 – Verzeichnisdienst 921
 – WMI 397, 399, 401
 Pfadangabe 237
 Pfeilspitze 1190
 Pflichtparameter 10
 PHP 141

- PhysicalDeliveryOfficeName 930, 955
- PIN 694
- Ping 65, 394, 853, 1150
- Ping-Host 25 f., 853
- Ping-VM 1033
- Pipe 88
- Pipeline 3, 38, 87, 89, 103, 122, 227, 325, 431
 - Ausgabe 1131
 - Eingabe 1139
- Pipeline Processor 90, 1122
- PipelineVariable 50, 125, 219
- Pipelining 87, 178, 233
- Plattform Invoke 447
- Plattformunabhängigkeit 1195
- Platzhalter 47, 222
- Plug-and-Play 795
- Polymorphismus 1192
- Port 350
- Portable-Executable-Format *siehe* PE
- PoshConsole 314
- Position 1100, 1137
- Postfach 994
- Postfix-Notation 937
- Power Management 404
- PowerGUI 130, 282
- PowerShell 3, 56, 87
 - Extension 619
 - Hosting 3
 - Konsole 273
 - Laufwerk 233, 240, 772
 - Remoting 320
 - Skriptsprache 141
 - Version 1.0 31
 - Version 2.0 31
 - Version 3.0 31
 - Version 4.0 32
 - Version 5.0 32
 - Version 5.1 32
 - Web Admin 632
- PowerShell Analyzer 298
- PowerShell Community Extensions 25
- PowerShell Community Extensions *siehe* PSCX
- PowerShell Core XXIII, , 3 f., 18, 32, 39, 323
 - Funktionsumfang 325
 - installieren 18
 - Konsole 342
 - Module 593
 - Version 5.1 37, 321, 323, 1038
 - Version 6.x 4, 18, 323
 - WMI 391
- PowerShell Direct 267, 269, 1041
- PowerShell Gallery 25, 551, 590
- PowerShell Management Library for Hyper-V 1010, 1031
- PowerShell Remoting *siehe* Remoting
- PowerShell Script Analyzer 292, 302 f.
- PowerShell Web Access *siehe* PSWA
- PowerShell-Remoting 304
- PowerShell Remoting Protocol 243, 353
- powershell.exe 55, 601, 636, 1079
- powerShellExePath 344
- PowerShellGet 590
- PowerShellPlus 130, 282, 311, 1144
- PowerStudio 299
- PrimalScript 316
- Principal 890
- Print Ticket XML 797
- Printer 209
- Printing 797
- PrintManagement 796
- Private 855
- Privileg 408
- Process 92, 117, 125, 551, 667, 1144
- ProcessRecord() 1122, 1126, 1137
- Professional Developer Conference 31
- profile 471
- profile.ps1 379, 1093, 1146
- ProfilePath 955
- Profilskript 280, 471, 474
- Programmcodeanalyse 292
- Programmgruppe 393
- Programmiersprache 188
- Programmiersprachenunabhängigkeit 1195
- Prompt 635
- PromptForChoice() 452
- Propagation Flags 887
- Property 108, 219, 1185, 1201
- Property Cache 925
- PropertyCollection 917
- PropertyDataCollection 417, 427 f.
- PropertyGrid 1078
- PropertyNames 917
- PropertyValueCollection 917, 924
- ProtectedFromAccidentalDeletion 955, 959
- Protokolldatei 410
- Protokollierung 463
- Provider 236
 - ADO.NET 715
 - Dateisystem 645
 - PowerShell 234
 - Verzeichnisdienst 913
- WMI 394
- Proxy 869
- Proxy-Commandlet 261, 1109
- ProxyCommand 1110
- Prozedur 191
- Prozess 44 f., 127, 393, 403
 - auflisten 228, 831
 - beenden 835
- ps 347
- PSBase 920, 926 f.
- PSCmdlet 1121
- PSCodeGen 618
- PSComputerName 255
- PSConfig 603
- PSCredential 454 f., 834, 975
- PSCustomObject 117, 443, 700, 704, 1104
- PSCX 26, 209, 456, 591, 593 f., 603, 614, 669, 675 f., 913
- PSCX *siehe* PowerShell Community Extensions
- psd1 545
- PSDiagnostics 604
- PSDriveInfo 646
- PSDSCRunAsCredential 554
- psedit 284, 291
- PSEdition 321
- PSHome 151
- PSHost 151, 1179
- PSHostRawUserInterface 1179 f.
- PSHostUserInterface 1179 f.
- PSImageTools 618
- psISE 289
- PSItem 88, 444
- PSModuleAutoLoadingPreference 151, 599
- PSModulePath 339, 347, 588, 1166
- PSObject 112, 1182 f.
- PSReadline 276 f., 342
- PSRemotingJob 482
- PSRSS 618
- PSScheduledJob 489
- PSScriptRoot 135
- PSSession 256
- PSSnapIn 1124
- PSSystemTools 618, 646, 794 f.
- psUnsupportedConsole Applications 287 f.
- PSUserTools 618
- PSVariable 144
- psversiontable 321
- PSWA 304, 306
- Public 855, 1137
- Public Network 246
- Pull Request 359
- Pull-ContainerImage 1056
- Punktnotation 98, 371, 432

- Push-ContainerImage 1070
- Put() 432 f.
- pv *siehe* PipelineVariable
- pwsh 19, 342
- Python 141

- Q**
- Quantifizierer 167
- Quantor 167
- QueryDialect 425
- Quest 603, 620

- R**
- RawUI 289
- RDP 269, 860
- Read-Host 451, 456, 505 f.
- ReadAccess 683
- Receive-Job 481, 483 f., 503
- Rechenleistung 127
- Recovery Console 1035 f.
- recurse 46, 651, 957
- recursive 957, 962
- Redirection *siehe* Umleitung
- Redstone 24, 32, 70
- REFERENCES OF 410
- Referenzkopie 179 f.
- Refresh() 1190
- RefreshCache() 925
- RefreshFrequencyMins 571
- Regel 824, 858
- region 291
- Register-CimIndicationEvent 418, 539
- Register-DnsClient 849
- Register-Event 537
- Register-ObjectEvent 687
- Register-Packagesource 594, 806
- Register-PSSessionConfiguration 257, 260, 627
- Register-ScheduledJob 495 f.
- Register-WmiEvent 534, 539
- Registrierungsdatenbank 3, 233, 239 f., 394, 564, 771
 - Schlüssel 771
- Registry 402, 404, 551, 771
- RegistryKey 801, 889
- RegistrySecurity 891
- RegistryValueChangeEvent 408, 532
- Regulärer Ausdruck 146, 165, 663
- Relative Distinguished Name 926 f.
- Remote Desktop Protocol *siehe* RDP
- Remote Desktop Service 604
- Remote Procedure Call *siehe* RPC
- Remote Server Administration
 - Tools 604, 913, 949
- Remoting 133, 241, 269, 355, 797, 1041
 - Implizit 261, 1109
 - Port 267
- Remove-ADGroup 974
- Remove-ADGroupMember 957, 974
- Remove-ADObject 957 ff.
- Remove-ADOrganizationalUnit 961
- Remove-ADUser 51, 962
- Remove-Alias 62, 87, 340
- Remove-Buchung 1160
- Remove-CimInstance 418, 438
- Remove-Computer 785
- Remove-Container 1065
- Remove-ContainerImage 1066, 1070
- Remove-DirectoryEntry 619, 1144
- Remove-Event 537
- Remove-EventLog 242, 877
- Remove-GLink 983
- Remove-GPO 982
- Remove-GPPrefRegistryValue 988
- Remove-GPRegistryValue 988
- Remove-Item 45, 49, 51, 62, 87, 645, 657, 772, 775
- Remove-ItemProperty 774
- Remove-Job 481, 484
- Remove-JobTrigger 494
- Remove-LDAPObject 945, 1096
- Remove-LocalUser 989
- Remove-Module 588, 602
- Remove-NetFirewallRule 855, 859
- Remove-NetFirewallRule-Funktion 859
- Remove-NetIPAddress 846
- Remove-NetRoute 846
- Remove-ODBCDsn 765
- Remove-PrintJob 797 f.
- Remove-PSBreakpoint 466
- Remove-PSSession 257 f.
- Remove-PSSnapin 328
- Remove-PswaAuthorizationRule 307
- Remove-Service 340, 842
- Remove-SmbShare 49, 51, 683
- Remove-Variable 150
- Remove-VM 1011, 1021 f.
- Remove-VMSnapshot 1026
- Remove-WebApplication 1008
- Remove-WebAppPool 1008
- Remove-Website 1008
- Remove-WebVirtualDirectory 1008
- Remove-WindowsCapability 820
- Remove-WindowsFeature 809, 814 f.
- Remove-WmiObject 418, 437
- Remove_DirectoryEntry 1154
- Rename-ADObject 957 f.
- Rename-Computer 784
- Rename-Drive 650
- Rename-GPO 982
- Rename-Item 657
- Rename-NetAdapter 848
- Rename-NetFirewallRule 855
- Rename-VM 1011
- Rename-VMSnapshot 1026
- Repair-VM 1011
- Replace 163
- Replikation 393
- Repository 403
 - requires 138
- Resize-VHD 1011, 1022
- Resolve-Assembly 379
- Resolve-DnsName 849, 852
- Resolve-Host 851
- Resolve-Path 238
- ResponseHeaders 371
- Ressource 551
- REST 870
- Restart-Computer 242, 785, 815
- Restart-PrintJob 797
- Restart-Service 52, 264, 267, 837, 840
- Restart-VM 1011
- Restore-ADObject 958
- Restore-Computer 789
- Restore-DscConfiguration 563
- Restore-GPO 983
- Restore-VMSnapshot 1026 f.
- Restricted 136
- Restricted Runspace 627
- Resume-PrintJob 797
- Resume-Service 837, 840
- Resume-VM 1011
 - return 182, 239, 1122
- Revoke-SmbShareAccess 684
- Richtlinienergebnisbericht 985
- Robocopy 660 ff.
- Rolle 809, 1038
- Rollendienst 809
- rootcimv2 401
- RPC 242
- RSAT 1010
- RSS 618, 863 f.
- Ruby on Rails 1056
- RuleCollection 826
- Run-ContainerImage 1057 f., 1061 ff.
- RunNow 495
- Runspace 298, 627
- RuntimeException 202
- Rückgabeobjekt 1131

S

- sa 750
- SAM 922
- SAMAccountName 929, 932, 937, 955
- SAPI.SPVoice 230, 389
- Sapien 316, 318 f., 624 f.
- Satya Nadella 324
- Save-ContainerImage 1069
- Save-Help 80
- Save-Module 591
- Save-VM 1011
- Schablone 1186 f.
- Schalter 46, 1163
- Schattenkopie 689
- Scheduled Task 489
- ScheduledJob 495
- Schema 926, 953
 - Active Directory 931
 - WMI 402
- Schemaabfrage 411
- SchemaNameCollection 918
- SchemaNamingContext 951
- Schleife 186
- Schlüssel 233
- Schlüsselattribut
 - WMI 396
- Schnittstelle 205, 719, 1188, 1192
 - .NET 1203
- Schriftart 784
- Schtasks.exe 490
- Scope *siehe* Gültigkeitsbereich
- script 149
- Script 565
- Script Analyzer 292
- Scripting.FileSystemObject 388
- ScriptMethod 441
- ScriptPaneBackgroundColor 289
- SDDL 630, 678, 902
- sealed 1191
- Searcher 786
- SearchScope 959
- Secure String 693 f.
- Security Descriptor 886
- Security Descriptor Definition Language 259
- Security Identifier 886, 891, 893 f.
- Security Service Provider 407
- Select
 - PowerShell 117
- SELECT
 - WQL 410 f., 413
- Select-Object 10, 87 f., 97, 109, 113, 117, 121, 126 f., 219, 430, 708, 1146
- Select-String 65, 94, 663, 698
- Select-Xml 707 f.
- SelectNodes() 707 f.
- SelectSingleNode() 707 f.
- Semantic Versioning 147
- Semaphore 889
- Semikolon 127, 833
- Send-MailMessage 861 f.
- Send-SmtpMail 861 f.
- sequence 507
- Serialisierung 104, 250
- Seriennummer 781
- Server 957
- Server Management Objects *siehe* SMO
- Server Manager 976
- ServerCertificateValidationCallback 870
- ServerRemoteHost 309
- ServerURL 571
- Service 551
- ServiceController 250, 837
- Serviceorientierung *siehe* SOA
- Session 390, 786
- sessionState 307
- Set-Acl 885, 898, 901
- Set-ADAccountPassword 957
- Set-ADGroup 974
- Set-ADObject 958 f.
- Set-ADOrganizationalUnit 961
- Set-ADUser 962, 964
- Set-Alias 61
- Set-AppLockerPolicy 825
- Set-AuthenticodeSignature 477
- Set-BPAREsult 911
- Set-CimInstance 418, 434
- Set-Clipboard 328, 456 f.
- Set-Content 645, 697, 713, 863
- Set-DataRow 736
- Set-DataTable 620, 736
- Set-Date 172
- Set-DistributionGroup 994
- Set-DnsClientServerAddress 846, 848 f.
- Set-ExecutionPolicy 14 ff., 129, 135 f., 478
- Set-FileTime 654
- Set-FirewallProfile 856
- Set-GPInheritance 987
- Set-GPLink 983
- Set-GPPermissions 988
- Set-GPPrefRegistryValue 987
- Set-GPRegistryValue 987
- Set-Item 264, 645
- Set-ItemProperty 654, 774, 861
- Set-JobTrigger 494
- Set-Location 56, 233, 645, 771
- Set-Mailbox 994
- Set-Methode 108
- Set-NetFirewallPortFilter 855
- Set-NetFirewallProfile 855
- Set-NetFirewallRule 855, 858
- Set-NetIPInterface 846
- Set-ODBCDriver 765
- Set-ODBCDsn 765
- Set-PrintConfiguration 797
- Set-PSBreakpoint 465 f.
- Set-PSDebug 147, 459 f.
- Set-PSReadlineOption 17 f., 277, 342
- Set-PSSessionConfiguration 257
- Set-ScheduledJob 495
- Set-Service 341, 837, 840 f.
- Set-StrictMode 147
- Set-TargetResource 578
- Set-TraceSource 463
- Set-Variable 144, 152, 1083 f.
- Set-VHD 1022
- Set-VM 1011, 1013
- Set-VMMemory 1033
- Set-VolumeLabel 650
- Set-VolumneLabel 25
- Set-WmiInstance 418, 434
- Set-WSManQuickConfig 246
- SetInfo() 916, 925 f.
- Setter 108, 1201
- SHA256 663
- Shell 3, 87
- Shell.Application 664, 676
- Shielded VM 1039
- ShouldProcess() 1154 f.
- Show() 1073
- Show-Command 75 f., 285
- Show-EventLog 242, 878
- Show-HyperVMenu 1032
- Show-NetFirewallRule 855
- Show-Service 242
- Show-VMMMenu 1032
- ShowDialog() 1088
- ShowWindow 76
- Shutdown 638
- Sicherheit
 - COM 406
 - Dateisystem 394, 404
 - PowerShell 135
 - WMI 406
- Sicherheitsabfrage 1104, 1154
- Sicherheitsbeschreibung 886
- SicherheitsEinstellung 885
- Sicherheitsmodell 3
- Sicherheitsrichtlinie 137
- SID 886
- Side-by-Side Executing 1197
- Signatur
 - digital 475
- Signieren 476
- SilentlyContinue 51, 199, 660, 963

- Simple Network Management
 - 392 f., 404
- Simple Object Access Protocol
 - siehe* SOAP
- Sitzung 256 ff.
- Sitzungskonfiguration 259, 627, 631
- Skip 97
- SkipNetworkProfileCheck 247, 860
- Skript 129, 131
 - PowerShell 129
- Skriptausführungsrechte 14
- Skriptausführungsrichtlinie 15
- Skriptblock 149, 249, 1083
- Skriptdatei 129
- Skripteigenschaft 107, 111
- Skriptfenster 26
- Skriptmodul 1165
- Skriptsprache 1089, 1091
- SMB 1038
- SMO 742, 748 f., 759 f., 762
- Smoking Man 914
- SMTP 861 f.
- SmtpClient 861
- SNA Server 404
- Snap-In 328, 581, 587, 601, 1123 f., 1146, 1157
- Snapshot
 - Hyper-V 1026
- Snippet 284
- SOA 1195 f.
- SOAP 241, 403, 868
- Software 393, 625
 - deinstallieren 803
 - installieren 575, 577, 802
 - inventarisieren 799
 - verwalten 799
- Software Restriction Policy 824
- Softwareentwickler 365
- Softwareentwicklungsplattform 1196
- Softwarekomponente 364, 378, 381
- Softwarepaket 806
- Softwarequelle 806
- Sort-Object 87 ff., 113, 118, 120 f., 126 ff., 193, 1122
- Sortieren 118
- Speech 209
- SpeechSynthesizer 230
- Speicher 95
- Speicherbereinigung 1197
- Speicherverbrauch 720
- Speicherverwaltung 1197
- Spitzname 1188
- Spoolerdienst 797
- Spooling 797
- Sprachausgabe 209, 230, 389
- Sprache 546
- Sprachkürzel 546
- SQL 410, 768
- SQL Server 324, 741
 - Agent 759
 - Laufwerk 743
- SQL Server Management Studio 743, 759
- SQLASCOMMANDETS 741
- Sqlcmd.exe 747
- SqlCommand 724, 747
- SqlConnection 370, 722, 726, 728
- SqlDataSourceEnumerator 718
- SQLPS 235, 740, 742, 746
- SQLPSX 740, 742, 751 f., 759
- SqlServerCe 716
- SqlServerCmdletSnapin100 741
- SSH 354 f.
- sshs 354
- SSL 306, 870
- STA 1079
- StackPanel 1084
- StackTrace 151
- Stammzertifizierungsstelle 476, 479
- Standarddrucker 228
- Standardkonsole 273
- Start-BITSTransfer 874
- Start-Container 1065
- Start-ContainerProcess 1061 f.
- Start-DscConfiguration 557
- Start-Job 481 ff.
- Start-Process 126, 489, 831 f., 834
- Start-PSSession 485
- Start-Service 251, 837, 840
- Start-Sleep 138
- Start-Transaction 467 ff.
- Start-Transcript 505
- Start-VM 1011, 1020, 1033
- Start-WBBackup 690
- Start-WBFileRecovery 691
- Start-WBHyperVRecovery 691
- Start-WBSystemStateRecovery 691
- Start-WBVolumeRecovery 691
- Start-Webitem 1007
- Start-Website 1007
- Startmenü 393
- Startup 638
- static 205
- Status 229, 797
- Stop 51, 199
- Stop-Computer 242, 785
- Stop-Container 1065
- Stop-Job 481, 484
- Stop-Process 100, 126, 505, 831, 835, 1144
- Stop-Service 45, 837, 840
- Stop-VM 1011
- Stop-WBJob 690
- Stop-Webitem 1007
- Stop-Website 1007
- StopProcessing() 1122
- Stopwatch 359
- Stored Procedure 729
- Streaming 90
- StreetAddress 955
- String 156, 163
- Subnetzmaske 846
- SUBTREE 937
- SubTree 959
- Subversion 140
- Suche
 - Active Directory 937
 - Assembly 655
 - LDAP 918
 - Verzeichniseintrag 927
 - XML 707
- SupportsShouldProcess 1154
- Surname 955
- Suse 37
- Suspend 50
- Suspend-PrintJob 797
- Suspend-Service 837, 840
- Suspend-VM 1011
- Switch 46, 182, 188, 1009
- SwitchParameter 1163
- Sybase 717
- Symbolic Link 671, 673
- SymLink 673 f.
- SYNOPSIS 1108
- Synopsis 1115
- Syntaxfarbhervorhebung 302
- System 1199 f.
- System ACL 890
- System Center Virtual Machine Manager 622
- System Management Server 404
- System-DSN 767
- System.ApplicationException 202
- System.Boolean 237
- System.Collections.Hashtable 177
- System.Console 375, 633
- System.Data 368
- System.Data.Odbc 716, 768
- System.Data.OleDb 716, 723
- System.Data.OracleClient 716, 723
- System.Data.SqlClient 716, 723, 747
- System.Data.SqlClient.
SqlConnection 370
- System.Data.SqlServerCe 716
- System.DateTime 171, 370 f., 374
- System.Diagnostics.EventLog 877
- System.Diagnostics.Process 89, 100, 104, 217, 831, 1142

System.DirectoryServices 368,
913 ff., 917, 919 f., 923, 928, 931,
937, 978, 991
System.DirectoryServices.Active-
Directory 978
System.Directoryservices.
DirectoryEntry 369, 372
System.dll 1200
System.Enum 385
System.Environment 248, 778 f.,
781, 783, 885, 1126 f., 1131
System.Globalization.CultureInfo
635
System.Int32 145, 154
System.IO.Compression 676
System.IO.Directory 889
System.IO.DirectoryInfo 441, 651 f.
System.IO.DriveInfo 373, 375, 378,
384, 646, 648
System.IO.DriveType 384
System.IO.File 889
System.IO.FileInfo 441, 651 f., 664
System.Management 368, 1133
System.Management.Automation
83, 1121, 1124, 1133, 1182
System.Management.Automation.
Cmdlet 1122
System.Management.Automation.
PathInfo 238 f.
System.Management.Automation.
PSCustomObject 700
System.Management.Automation.
PSDriveInfo 646
System.Management.Manage-
mentObject 173
System.Math 374
System.Media.SoundPlayer 373
System.Net.Mail 861, 863
System.Net.WebClient 371, 863,
870
System.Object 102, 104, 443, 1000,
1142, 1146
System.Random 156, 369 f.
System.Reflection 378
System.Security 890
System.Security.AccessControl
889
System.ServiceProcess.Service-
Controller 104, 837 f., 1142
System.String 156, 161, 349, 1126
System.TimeSpan 172
System.Type 102, 151, 226
System.ValueType 179
System.Windows 1079
System.Windows.FontStyle 1082
System.Windows.Forms 378, 665,
1073

System.Xml.Node 709
System32 121, 127 f.
Systemattribut
– WMI 397
Systemdienst 91, 393
– auflisten 411
– überwachen 413
Systemende 638
SystemEvent 532
Systemklassen
– WMI 395
Systemmanagement 391
SystemParametersInfo 447
Systemstart 638
Systemwiederherstellung 789
Sysvol 638

T

T-SQL 747
Tab Completion 276
Tabellenformatierung 212
TabPanel 1084
Tabulator 160
Tabulatorvervollständigung 275
Tag-ContainerImage 1069
TAR 675
TaskScheduler 618
TCP/IP 846
tcsh 324
Team Foundation Server *siehe* TFS
Tee-Object 123 f.
Telnet 248
Terminal Services 394
Terminating Error 197, 1150
Terrabyte 154
Test-32Bit 794
Test-64Bit 794
Test-AppLockerPolicy 825, 827
Test-Assembly 654
Test-Connection 26, 93 f., 853 f.
Test-CustomerID 1106
Test-DbConnection 736
Test-DscConfiguration 557
Test-FileCatalog 82, 663
Test-IsAdmin 885
Test-ModuleManifest 589
Test-Path 237
Test-PswaAuthorizationRule 307
Test-ServiceHealth 993
Test-SqlScript 751
Test-TargetResource 578
Test-UserGroupMembership 936
Test-VHD 1011, 1022
Test-Xml 706
Textanzeige 1087
Textdatei 127, 697
Texteingabefeld 453, 1075
TextInfo 163
Textpad 130
TFS 140
Thawte 475
this 444, 1084
Thread 510
Thread-Modell 1079
ThrottleLimit 255
throw 182 f., 202
ThrowTerminatingError() 1150
Thumbprint 903
TIFF 665
TimeSpan 461
Tivoli 413
TLS 870
ToLower() 163
Ton 375
ToString() 102, 104, 377, 1142, 1192
TotalProcessorTime 226
ToTitleCase() 163
ToUpper() 163
TPM 693
Trace-Command 640
Tracing *siehe* Ablaufverfolgung
Transaktion 328, 467
Transformation 1087
Translate() 894
trap 182, 197, 202 f.
Treiber 625
– ODBC 766
Trigger 493
Troubleshooting Pack 605, 907
true 146, 151, 794
Trusted Host 264
Trusted Platform Module *siehe*
TPM
Trustee 886
Try-Catch-Finally 197, 203
Try...Catch 964
Tuva 1035
Typ 145, 363, 1186, 1200
– Namensgebung 1200
Typadapter 145 f., 174
Typbezeichner 145
Type Cast *siehe* Typkonvertierung
types.ps1xml 62 f., 112 f.
Typisierung 144
Typkennzeichner *siehe*
Typbezeichner
Typkonvertierung 118, 148
Typname *siehe* Typbezeichner

U

UAC 132, 137, 279, 833
Überladung 196

Ubuntu 21, 37
ufw 350
Umgebungsvariable 393
– Linux 347
Umlaut 698
Umleitung 229
Undefined 136
Undo-Transaction 328, 467, 469 f.
UniformGrid 1084
Uninstall-Package 807
Uninstall-WindowsFeature 808 f.
Universal Coordinated Time 398, 434
Unix 3, 31, 87f., 141, 645, 671
Unlock-BitLocker 695 f.
Unregister-PSSessionConfiguration 257, 261, 630
Unrestricted 136
Unterbrechungsfreie Stromversorgung *siehe* USV
Unternehmensraum 1198
Unterordner 104, 651
Unterroutine 190
Unterschlüssel 233
until 182
Update 393, 625
– Einstellungen 788
– installieren 787
– suchen 786
Update-Help 80
UpdateColl 786
UsePropertyCache 925
User 551, 919, 928, 941
User Account Control *siehe* Benutzerkontensteuerung
User Settings 344
user32.dll 447
UserDomainName 885
UserName 885
UseTestCertificate 306
UseTransaction 468
using 508
USV 795
UTF8 698

V

Validate-CustomerID 1106
ValidateCount 1100
ValidateLength 153, 1101, 1163
ValidateNotNull 1100, 1163
ValidatePattern 153, 1101, 1163
ValidateRange 153, 1101
ValidateScript 153, 1101
ValidateSet 153
ValueFromPipeline 1100, 1139, 1141, 1146

ValueFromPipelineByPropertyName 1100, 1141
ValuesCollection 917
ValueType 179
Variable 29, 102, 124, 143, 151, 222, 233
– Auflösung 157
– eingebaut 151, 340
– vordefiniert 151, 340
– Workflow 509
Variablenuflösung 157f., 222
Variablenkennzeichner 124, 143
Variablentypisierung 144
VB 445, 447
Verbindungszeichenfolge 370, 722, 729
Verbose 50, 52, 557, 1149
VerbosePreference 52, 152, 1149
VerbsCommon 1162
VerbsCommunications 1162
VerbsData 1162
VerbsDiagnostic 1162
VerbsLifeCycle 1162
VerbsSecurity 1162
Vererbung 402, 1190, 1202
Vererbungsdiagramm 1190
Vererbungshierarchie 415, 953, 1190
– WMI 402
Vergleich 125
Vergleichsoperator 113
Verifikation 1197
VeriSign 475
Verknüpfung 670
Verzeichnisattribut 924
Verzeichnisdienst 112, 404, 619, 926, 938
Verzeichnisdienstklasse 926
Verzeichnisobjekt 923, 928
Verzweigung 123
VHD 1021 f., 1032, 1039
VHDX 1018, 1022, 1032
Video 1087
View 217
VirtualHardDisk 1022
Virtualisierung 1009
VirtualizingStackPanel 1084
Virtuelle Maschine *siehe* VM
Virtuelles System 1009
Virus 137
Visual Basic 445, 1193
Visual Basic .NET 31, 1119 f.
Visual Basic 6.0 1197
Visual Studio 299, 301, 513, 1059, 1066, 1068, 1089, 1119 f., 1146
– Container 1059
Visual Studio Code 130, 301, 342, 1063

Visual Studio Team Services *siehe* VSTS
Visual Web Developer Express 1120
VM 623, 1009, 1016
VMBus 267
VMGUID 267
VMName 267
void 227
Volume Shadow Copy Service *siehe* VSS
VolumeLabel 372
VSCode
– Visual Studio Code 301
VSCode-PowerShell 301, 342
VSS 688
VSTS 140

W

Wait 557
Wait-Job 481, 484
Wait-Process 836
WaitForAll 579
WaitForAny 579
WaitForSome 579
Walter Skinner 914
WarningAction 50 f., 199
WarningVariable 51
Warnung 51
WAS 839
WBEM 33, 392
WDAC 765
Web Administration 605
Web Based Enterprise Management 392
Web Service Description Language 869
Web Services Description Language *siehe* WSDL
WebAdministration 997, 999, 1046
Webanwendung 1196
Webdienst 868
Weblog 863, 1207
Webserver 235, 1002
Webservices 868, 870
Website 865, 1002, 1008
Well-Known GUID 922
Well-Known Object 922
Well-Known Security Principal 895
WellKnownSidType 895
Werkzeug 273
Wertemenge 173
Wertkopie 179 f.
whatif 50 ff., 152, 657, 798, 1104, 1154
WhatIfPreference 52
WHERE 410

- Where-Object 56 f., 87 ff., 100 f., 113, 115, 122, 126 f., 228, 838, 1122, 1134, 1146
- while 182
- Whistler 400
- whoami.exe 280
- Width 212
- Wiederherstellungspunkt 789
- Win32 395
- Win32_OpenSSH 353
- Win32-API 447
- Win32_Account 915
- Win32_ACE 680
- Win32_Battery 795
- Win32_Bios 780
- Win32_BootConfiguration 780
- Win32_CDRomDrive 431
- Win32_CDRomdrive 793
- Win32_CodecFile 801
- Win32_ComponentCategory 411
- Win32_ComputerShutdownEvent 408, 532
- Win32_Computersystem 778
- Win32_Currenttime 172
- Win32_Desktop 915
- Win32_Diskdrive 793
- Win32_Group 915
- Win32_Keyboard 794
- Win32_LocalTime 173
- Win32_LogicalDisk 401 f., 411, 435, 646, 648 f., 1133
- Win32_MappedLogicalDisk 650
- Win32_MemoryDevice 793
- Win32_NetworkAdapter 794
- Win32_NetworkAdapterConfiguration 411, 843, 849 f.
- Win32_NTLogEvent 411, 413, 533
- Win32_OperatingSystem 778 f.
- Win32_OSRecoveryConfiguration 781
- Win32_PerfRawData 881
- Win32_PerfRawData_PerfOS_Processor 881
- Win32_PerfRawData_PerfProc_Process 881
- Win32_PingStatus 853
- Win32_PointingDevice 794
- Win32_PowerManagementEvent 532
- Win32_Printer 795, 797, 843
- Win32_Printjob 795 f.
- Win32_Process 533
- Win32_Processor 793 f.
- Win32_ProcessStartTrace 532
- Win32_Product 799, 802 f.
- Win32_Quickfixengineering 801
- Win32_SecurityDescriptor 680
- Win32_Service 411, 413, 533, 837
- Win32_Share 678 f.
- Win32_SoundDevice 793
- Win32_SystemConfiguration-ChangeEvent 408, 532
- Win32_Tapedrive 793
- Win32_TCIPPrinterPort 795 f., 843
- Win32_Trustee 680
- Win32_USBController 794
- Win32_UserAccount 125, 401, 915
- Win32_VideoController 420, 431, 793, 795
- Win32_Volume 650
- Win32_WindowsProductActivation 781
- window 1084
- Windows 10 4, 32, 37, 267, 273, 276, 610
 - Anniversary Update 32
 - Rolle 1038
- Windows 2000 400
- Windows 7 604
- Windows 8 425, 997
- Windows 8.1 607
- Windows 9x 403
- Windows Activation Service *siehe* WAS
- Windows as a Service 32
- Windows Communication Foundation 394
- Windows Container *siehe* Docker
- Windows Data Access Components *siehe* WDAC
- Windows Defender 791, 1039
- Windows Driver Model 404
- Windows Explorer 689, 774
- Windows Firewall 625, 854, 859
 - per PowerShell konfigurieren 854
- Windows Forms 300, 1073, 1076, 1078
- Windows Installer 404, 824
- Windows Management Framework 5, 392, 797
- Windows Management Instrumentation 37
- Windows ME 403
- Windows Nano Server 37, 1035, 1051, 1056
- Windows Nano Server 1709 37
- Windows Nano Server 2016 37
- Windows PowerShell XXIII, 3 f.
- Windows PowerShell Community Extensions 614
- Windows Pre Installation Environment *siehe* WinPE
- Windows Presentation Foundation *siehe* WPF
- Windows Remote Management *siehe* WinRM
- Windows Script Host 31, 39, 135
- Windows Server 1709 37
- Windows Server 2003 31, 400, 403, 937
- Windows Server 2012 425, 605, 975, 997
- Windows Server 2012 R2 310, 607
- Windows Server 2016 4, 32, 37, 267, 273, 276, 610
- Windows Server Core 282, 1051, 1056
- Windows Server-Containern *siehe* Docker
- Windows Troubleshooting Platform 907
- Windows Update 786, 789
 - Agent API 1044
 - Nano Server 1044
- Windows Vista 1193
- Windows Workflow Foundation 327
- Windows XP 38, 400, 409
- Windows-Authentifizierung 750
- Windows-Firewall 860
 - per PowerShell konfigurieren
- Windows-Systemdienst 837
- WinMgmt.exe 403 f.
- WinPE 693
- WinRM 241, 243, 245 f., 403, 482, 785
- WITHIN 410, 532
- WKGUID 922
- WMI 3, 33, 37, 173, 241, 327, 391, 394, 402, 422, 1133, 1137, 1150
 - Class Explorer 415
 - Command Shell 38
 - Data Query 411
 - Ereignis 408
 - Event Query 410, 412
 - Klasse 415
 - Namespace 399
 - Object Browser 414 f.
 - Query Language 410, 424
 - Repository 403, 408, 426
 - Schema 402, 411
 - Schema-Query 411
 - Steuerung 403
- WMI API 416
- WMI Object Browser 414
- WMI Query Language *siehe* WQL
- WMIClass 391, 422
- WMISEARCHER 391, 422, 424
- Word 119
- Workflow 499, 505 ff., 513
 - Designer 514
 - Einschränkungen 504

- Persistenz 512
- verschachtelt 509
- WorkflowInfo 514
- WorkingSet 112
- WorkingSet64 89
- Workspace Settings 344
- World Wide Wings 1160
- WPF 73, 282, 300, 339, 380, 503, 1073, 1079
- WPF PowerShell Kit 618, 1079
- WPK 618, 1079
- WQL 410, 425, 531, 1133
- WrapPanel 1084
- Write-BZip2 675
- Write-Clipboard 457
- Write-Error 221
- Write-EventLog 242, 328, 877, 879
- Write-GZip 675
- Write-Host 74, 209, 221, 309, 506
- Write-Output 64
- Write-Progress 229 f., 440
- Write-Tar 25, 675
- Write-Warn 221
- Write-Zip 675
- WriteDebug() 1150
- WriteError() 1150, 1153
- WriteObject() 1122, 1127, 1134 f.
- WriteVerbose() 1150, 1153
- WriteWarning() 1150, 1153
- WS-Management 241 f., 245, 265, 403, 416, 419, 421
- WScript.Shell 670
- WSDL 868
- WSH 920
- WSMan 235, 266

- Wurzelnamensraum 1198
- www.IT-Visions.de 603, 619, 632, 646, 736
- Wörterbuch 119

X

- X-Files 914
- x64 1009
- x86 1009
- XAML 503, 513, 1087
- XamlReader 1088
- XCopy-Deployment 1197, 1200
- xDscDiagnostics 573
- xDscWebService 568
- XFilesServer 914
- XML 79, 451, 583, 686, 705 f., 709, 711, 870, 1156
- XML Application Markup Language *siehe* XAML
- XML-Schema 706
- XML-Webservice 1198
- XmlAttribute 708
- XmlDocument 71
- XmlElement 708
- XPathDocumentNavigator 708
- XslCompiledTransform 711

Y

- YAML 1059
- Year 98
- YesNo 1074
- YesNoCancel 1074

Z

- Zahl 154
- Zahlenliteral 154
- Zeichenkette 156 f., 164, 222, 1143, 1162
 - ersetzen 163
 - Operation 162
 - trennen 164
 - verbinden 164
- Zeichensatz 698
- Zeilenumbruch 53, 96
 - Pipeline 96
- Zeitmessung 461
- Zeitplandienst 393
- Zertifikat 306, 476, 903
 - selbst signiert 476
- Zertifikatsdatei 477
- Zertifikatsspeicher 3, 233, 903
- Zertifikatsverwaltung 475 f., 479
- ZIP 674 f.
- ZipFile 676
- zsh 324
- Zufallszahl 155 f.
- Zugriff verweigert 434
- Zugriffsrechtliste 885, 891
- Zuweisungsoperator 179
- Zwischenablage 456
- Zwischencode 1195
- Zwischenschritt 122
- Zwischenspeicher 720