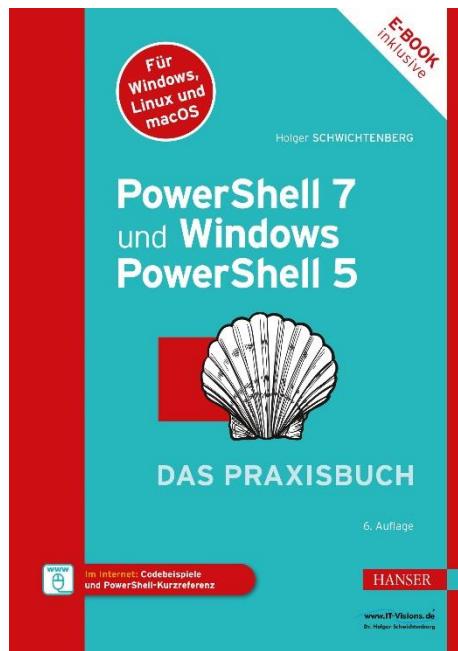


HANSER



Leseprobe

zu

PowerShell 7 und Windows PowerShell 5

von Holger Schwichtenberg

Print-ISBN: 978-3-446-48195-4

E-Book-ISBN: 978-3-446-48196-1

E-Pub-ISBN: 978-3-446-48244-9

Weitere Informationen und Bestellungen unter

<https://www.hanser-kundencenter.de/fachbuch/artikel/9783446481954>

sowie im Buchhandel

© Carl Hanser Verlag, München

Inhalt

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

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

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

8 Ausgaben	247
8.1 Ausgabe-Commandlets	247
8.2 Benutzerdefinierte Tabellenformatierung	250
8.3 Benutzerdefinierte Listenausgabe	252
8.4 Mehrspaltige Ausgabe	252
8.5 Out-GridView	253
8.6 Standardausgabe	255
8.7 Einschränkung der Ausgabe	257
8.8 Seitenweise Ausgabe	258
8.9 Ausgabe einzelner Werte	259
8.10 Details zum Ausgabeoperator	262
8.11 Ausgabe von Methodenergebnissen und Unterobjekten in Pipelines	265
8.12 Ausgabe von Methodenergebnissen und Unterobjekten in Zeichenketten	266
8.13 Unterdrückung der Ausgabe	267
8.14 Ausgaben an Drucker	267
8.15 Ausgaben in Dateien	268
8.16 Umleitungen (Redirection)	268
8.17 Fortschrittsanzeige	269
8.18 Sprachausgabe	269
9 Das PowerShell-Navigationsmodell (PowerShell Provider)	271
9.1 Einführungsbeispiel: Navigation in der Registrierungsdatenbank	271
9.2 Provider und Laufwerke	272
9.3 Navigationsbefehle	274
9.4 Pfadangaben	275
9.5 Beispiel	277
9.6 Eigene Laufwerke definieren	278
10 Fernausführung (Remoting)	279
10.1 RPC-Fernabfrage ohne WS-Management	280
10.2 Anforderungen an PowerShell Remoting	281
10.3 Rechte für PowerShell-Remoting	282
10.4 Einrichten von PowerShell Remoting	282
10.5 Überblick über die Fernausführungs-Commandlets	285
10.6 Interaktive Fernverbindungen im Telnet-Stil	285
10.7 Fernausführung von Befehlen	287
10.8 Parameterübergabe an die Fernausführung	291
10.9 Fernausführung von Skripten	292
10.10 Ausführung auf mehreren Computern	293
10.11 Sitzungen	294
10.12 Implizites Remoting	299

10.13 Zugriff auf entfernte Computer außerhalb der eigenen Domäne	300
10.14 Verwaltung des WS-Management-Dienstes	303
10.15 PowerShell Direct für Hyper-V	304
10.16 Praxislösung zu PowerShell Direct	306
11 PowerShell-Werkzeuge	309
11.1 PowerShell-Standardkonsole	309
11.2 Windows Terminal	324
11.3 Erweiterung der Konsolen	329
11.4 PowerShell Integrated Scripting Environment (ISE)	331
11.5 PowerShell Script Analyzer	342
11.6 PowerShell Analyzer	347
11.7 PowerShell Tools for Visual Studio	348
11.8 PowerShell Pro Tools for Visual Studio	350
11.9 Visual Studio Developer PowerShell	350
11.10 NuGet Package Manager Console (PMC)	353
11.11 Visual Studio Code mit PowerShell-Erweiterung	354
11.12 PowerShell-Erweiterungen für andere Editoren	356
11.13 PowerShell Web Access (PSWA)	357
11.14 Azure Cloud Shell	362
11.15 ISE Steroids	362
11.16 PowerShellPlus	363
11.17 PoshConsole	366
11.18 PowerGUI	367
11.19 PrimalScript	367
11.20 CIM Explorer for PowerShell ISE	369
12 Windows PowerShell Core 5.1 in Windows Nano Server	371
12.1 Installation	371
12.2 PowerShell-Skriptsprache	371
12.3 Werkzeuge	371
12.4 Fehlende Funktionen	372
13 PowerShell 7 für Windows, Linux und macOS	373
13.1 Motivation für den Einsatz der PowerShell 7 auf Linux und macOS	373
13.2 Basis der PowerShell 7	374
13.3 Identifizierung der PowerShell 7	375
13.4 Funktionsumfang der PowerShell 7	375
13.5 Entfallene Befehle in PowerShell 7	378
13.6 Erweiterungsmodule nutzen in PowerShell 7	384
13.7 Geänderte Funktionen in PowerShell 7	389

13.8	Neue Funktionen der PowerShell 7	391
13.9	PowerShell 7-Konsole	394
13.10	Praxislösung: Fallunterscheidung für PowerShell-Varianten	395
13.11	VSCode-PowerShell als Editor für PowerShell 7	396
13.12	Verwendung von PowerShell 7 auf Linux und macOS	400
13.13	PowerShell-Remoting via SSH	406
13.14	Performance-Vorteile der PowerShell 7	409
13.15	Dokumentation zur PowerShell 7	410
13.16	Quellcode zur PowerShell 7	412
Teil B: PowerShell-Aufbauwissen		415
14	Verwendung von .NET-Klassen	417
14.1	.NET versus .NET Core	417
14.2	Ermitteln der verwendeten .NET-Version	418
14.3	.NET-Bibliotheken	419
14.4	Microsoft Docs	421
14.5	Überblick über die Verwendung von .NET-Klassen	422
14.6	Erzeugen von Instanzen	422
14.7	Parameterbehaftete Konstruktoren	424
14.8	Initialisierung von Objekten	425
14.9	Nutzung von Attributen und Methoden	426
14.10	Statische Mitglieder in .NET-Klassen und statische .NET-Klassen	428
14.11	Generische Klassen nutzen	431
14.12	Zugriff auf bestehende Objekte	433
14.13	Laden von Assemblies	433
14.14	Liste der geladen Assemblies	435
14.15	Verwenden von NuGet-Assemblies	436
14.16	Objektanalyse	438
14.17	Aufzählungstypen (Auflistungen/Enumerationen)	439
15	Verwendung von COM-Klassen	443
15.1	Unterschiede zwischen COM und .NET	443
15.2	Erzeugen von COM-Instanzen	444
15.3	Abruf der Metadaten	444
15.4	Nutzung von Attributen und Methoden	445
15.5	Liste aller COM-Klassen	446
15.6	Holen bestehender COM-Instanzen	447
15.7	Distributed COM (DCOM)	447

16 Zugriff auf die Windows Management Instrumentation (WMI)	449
16.1 Einführung in WMI	449
16.2 WMI in der PowerShell	476
16.3 Open Management Infrastructure (OMI)	478
16.4 Abruf von WMI-Objektmengen	478
16.5 Fernzugriffe	479
16.6 Filtern und Abfragen	480
16.7 Liste aller WMI-Klassen	483
16.8 Hintergrundwissen: WMI-Klassenprojektion mit dem PowerShell-WMI-Objektadapter	484
16.9 Beschränkung der Ausgabeliste bei WMI-Objekten	488
16.10 Zugriff auf einzelne Mitglieder von WMI-Klassen	490
16.11 Werte setzen in WMI-Objekten	490
16.12 Umgang mit WMI-Datumsangaben	492
16.13 Methodenaufrufe	493
16.14 Neue WMI-Instanzen erzeugen	494
16.15 Instanzen entfernen	495
16.16 Commandlet Definition XML-Datei (CDXML)	495
17 Dynamische Objekte	499
17.1 Erweitern bestehender Objekte	499
17.2 Komplett dynamische Objekte	501
18 Einbinden von C# und Visual Basic .NET	503
19 Win32-API-Aufrufe	505
20 Benutzereingaben	508
20.1 Read-Host	508
20.2 Benutzerauswahl	509
20.3 Grafischer Eingabedialog	510
20.4 Dialogfenster	511
20.5 Authentifizierungsdialog	511
20.6 Zwischenablage (Clipboard)	513
21 Fehlersuche	514
21.1 Detailinformationen	514
21.2 Einzelschrittmodus	515
21.3 Zeitmessung	516
21.4 Ablaufverfolgung (Tracing)	517
21.5 Erweiterte Protokollierung aktivieren	519
21.6 Script-Debugging in der ISE	520
21.7 Kommandozeilenbasiertes Script-Debugging	520

22 Transaktionen	522
22.1 Commandlets für Transaktionen	522
22.2 Start und Ende einer Transaktion	523
22.3 Zurücksetzen der Transaktion	524
22.4 Mehrere Transaktionen	525
23 Standardeinstellungen ändern mit Profilskripten	526
23.1 Profilfade	526
23.2 Ausführungsreihenfolge	528
23.3 Beispiel für eine Profildatei	528
23.4 Starten der PowerShell ohne Profilskripte	530
24 Digitale Signaturen für PowerShell-Skripte	531
24.1 Zertifikat erstellen	531
24.2 Skripte signieren	533
24.3 Verwenden signierter Skripte	535
24.4 Mögliche Fehlerquellen	535
25 Hintergrundaufträge („Jobs“)	536
25.1 Voraussetzungen	536
25.2 Architektur	536
25.3 Starten eines Hintergrundauftrags	537
25.4 Hintergrundaufträge abfragen	538
25.5 Warten auf einen Hintergrundauftrag	539
25.6 Abbrechen und Löschen von Aufträgen	539
25.7 Analyse von Fehlermeldungen	539
25.8 Fernausführung von Hintergrundaufträgen	540
25.9 Praxislösung: Einen Job auf mehreren Computern starten	540
26 Geplante Aufgaben und zeitgesteuerte Jobs	542
26.1 Geplante Aufgaben (Scheduled Tasks)	542
26.2 Zeitgesteuerte Jobs	546
27 PowerShell-Workflows	552
27.1 Ein erstes Beispiel	552
27.2 Unterschiede zu einer Function bzw. einem Skript	556
27.3 Einschränkungen bei Workflows	557
27.4 Workflows in der Praxis	558
27.5 Workflows in Visual Studio erstellen	566

28 Ereignissystem	584
28.1 WMI-Ereignisse	584
28.2 WMI-Ereignisabfragen	584
28.3 WMI-Ereignisse seit PowerShell 1.0	586
28.4 Registrieren von WMI-Ereignisquellen seit PowerShell 2.0	587
28.5 Auslesen der Ereignisliste	588
28.6 Reagieren auf Ereignisse	590
28.7 WMI-Ereignisse seit PowerShell-Version 3.0	592
28.8 Registrieren von .NET-Ereignissen	592
28.9 Erzeugen von Ereignissen	593
29 Datenbereiche und Datendateien	595
29.1 Datenbereiche	595
29.2 Datendateien	597
29.3 Mehrsprachigkeit/Lokalisierung	598
30 Desired State Configuration (DSC)	601
30.1 Grundprinzipien	602
30.2 DSC für PowerShell 7	602
30.3 Ressourcen	603
30.4 Verfügbare DSC-Ressourcen	604
30.5 Eigenschaften einer Ressource	607
30.6 Aufbau eines DSC-Dokuments	607
30.7 Commandlets für die Arbeit mit DSC	608
30.8 Ein erstes DSC-Beispiel	608
30.9 Kompilieren und Anwendung eines DSC-Dokuments	609
30.10 Variablen in DSC-Dateien	611
30.11 Parameter für DSC-Dateien	612
30.12 Konfigurationsdaten	613
30.13 Entfernen einer DSC-Konfiguration	616
30.14 DSC Pull Server	619
30.15 DSC-Praxislösung 1: IIS installieren	626
30.16 DSC-Praxislösung 2: Software installieren	628
30.17 DSC-Praxislösung 3: Software deinstallieren	630
30.18 Realisierung einer DSC-Ressource	631
30.19 Weitere Möglichkeiten	631
31 PowerShell-Snap-Ins	632
31.1 Einbinden von Snap-Ins	632
31.2 Liste der Commandlets	636

32 PowerShell-Module	637
32.1 Überblick über die Commandlets	637
32.2 Modulararchitektur	638
32.3 Aufbau eines Moduls	639
32.4 Module aus dem Netz herunterladen und installieren mit PowerShellGet	640
32.5 Module manuell installieren	646
32.6 Doppeldeutige Namen	646
32.7 Auflisten der verfügbaren Module	648
32.8 Importieren von Modulen	649
32.9 Entfernen von Modulen	652
33 Ausgewählte PowerShell-Erweiterungen	653
33.1 PowerShell-Module in Windows 8.0 und Windows Server 2012	654
33.2 PowerShell-Module in Windows 8.1 und Windows Server 2012 R2	656
33.3 PowerShell-Module in Windows 10 und Windows Server 2019	658
33.4 PowerShell Community Extensions (PSCX)	662
33.5 PowerShellPack	666
33.6 www.IT-Visions.de: PowerShell Extensions	668
33.7 Quest Management Shell for Active Directory	668
33.8 Microsoft Exchange Server	670
33.9 System Center Virtual Machine Manager	671
33.10 PowerShell Management Library for Hyper-V (pshyperv)	671
33.11 PowerShell Configurator (PSConfig)	672
34 Delegierte Administration/Just Enough Administration (JEA) ..	674
34.1 JEA-Konzept	674
34.2 PowerShell-Sitzungskonfiguration erstellen	674
34.3 Sitzungskonfiguration nutzen	678
34.4 Delegierte Administration per Webseite	679
35 Tipps und Tricks zur PowerShell	680
35.1 Alle Anzeigen löschen	680
35.2 Befehlsgeschichte	680
35.3 System- und Hostinformationen	681
35.4 Anpassen der Eingabeaufforderung (Prompt)	682
35.5 PowerShell-Befehle aus anderen Anwendungen heraus starten	683
35.6 ISE erweitern	684
35.7 PowerShell für Gruppenrichtlinieneskripte	685
35.8 Einblicke in die Interna der Pipeline-Verarbeitung	688

Teil C: PowerShell im Praxiseinsatz	689
36 Dateisystem	691
36.1 Laufwerke	692
36.2 Ordnerinhalte	697
36.3 Dateieigenschaften verändern	704
36.4 Eigenschaften ausführbarer Dateien	705
36.5 Kurznamen	707
36.6 Lange Pfade	707
36.7 Dateisystemoperationen	708
36.8 Praxislösung: Dateien umorganisieren	708
36.9 Praxislösung: Zufällige Dateisystemstruktur erzeugen	710
36.10 Praxislösung: Leere Ordner löschen	711
36.11 Praxislösung: Geschwindigkeitsmessung des Dateisystems (beim Kopieren von Dateien)	713
36.12 Einsatz von Robocopy in der PowerShell	714
36.13 NTFS-Komprimierung	717
36.14 Dateisystemkataloge	718
36.15 Papierkorb leeren	718
36.16 Dateieigenschaften lesen	719
36.17 Praxislösung: Fotos nach Aufnahmedatum sortieren	719
36.18 Datei-Hash	720
36.19 Finden von Duplikaten	721
36.20 Verknüpfungen im Dateisystem	723
36.21 Komprimierung	728
36.22 Dateisystemfreigaben	732
36.23 Überwachung des Dateisystems	743
36.24 Dateiversionsverlauf	744
36.25 Windows Explorer öffnen	745
36.26 Windows Server Backup	745
37 Festplattenverschlüsselung mit BitLocker	747
37.1 Übersicht über das BitLocker-Modul	748
37.2 Verschlüsseln eines Laufwerks	749
38 Dokumente	750
38.1 Textdateien	750
38.2 CSV-Dateien	752
38.3 Analysieren von Textdateien	755
38.4 INI-Dateien	759
38.5 XML-Dateien	759

38.6	HTML- und Markdown-Dateien	771
38.7	JSON-Dateien	774
38.8	Binärdateien	785
38.9	Praxislösung: Grafikdateien verändern	786
38.10	Praxislösung: Drucken vieler Dateien	787
39	Microsoft Office	788
39.1	Allgemeine Informationen zur Office-Automatisierung per PowerShell	788
39.2	Praxislösung: Terminserien aus Textdateien anlegen in Outlook	789
39.3	Praxislösung: Outlook-Termine anhand von Suchkriterien löschen	791
39.4	Praxislösung: Grafiken aus einem Word-Dokument (DOCX) extrahieren	792
40	Datenbanken	795
40.1	ADO.NET-Grundlagen	795
40.2	Beispieldatenbank	801
40.3	Datenzugriff mit den Bordmitteln der PowerShell	802
40.4	Hilfsfunktions für den Datenbankzugriff (DBUtil.ps1)	815
40.5	Datenzugriff mit den PowerShell-Erweiterungen	818
40.6	Datenbankzugriff mit SQLPS	822
40.7	Datenbankzugriff mit SQLPSX	822
41	Microsoft-SQL-Server-Administration	823
41.1	PowerShell-Integration im SQL Server Management Studio	824
41.2	SQL-Server-Laufwerk „SQLSERVER:“	825
41.3	Die SQLPS-Commandlets	828
41.4	Die SQL Server Management Objects (SMO)	830
41.5	SQLPSX	833
41.6	Microsoft-SQL-Server-Administration mit der PowerShell in der Praxis	840
42	ODBC-Datenquellen	846
42.1	ODBC-Treiber und -Datenquellen auflisten	847
42.2	Anlegen einer ODBC-Datenquelle	848
42.3	Zugriff auf eine ODBC-Datenquelle	849
43	Registrierungsdatenbank (Registry)	851
43.1	Schlüssel auslesen	851
43.2	Schlüssel anlegen und löschen	852
43.3	Laufwerke definieren	852
43.4	Werte anlegen und löschen	853
43.5	Werte auslesen	854
43.6	Praxislösung: Windows-Explorer-Einstellungen	855
43.7	Praxislösung: Massenanlegen von Registry-Schlüsseln	855

44 Computer- und Betriebssystemverwaltung	857
44.1 Computerinformationen	857
44.2 Versionsnummer des Betriebssystems	859
44.3 Zeitdauer seit dem letzten Start des Betriebssystems	859
44.4 BIOS- und Startinformationen	860
44.5 Windows-Produktaktivierung	861
44.6 Umgebungsvariablen	861
44.7 Schriftarten	865
44.8 Computername und Domäne	865
44.9 Herunterfahren und Neustarten	866
44.10 Windows Updates installieren	867
44.11 Wiederherstellungspunkte verwalten	871
45 Windows Defender	872
46 Hardwareverwaltung	873
46.1 Hardwarebausteine	873
46.2 Plug-and-Play-Geräte	875
46.3 Druckerverwaltung (ältere Betriebssysteme)	875
46.4 Druckerverwaltung (seit Windows 8 und Windows Server 2012)	877
47 Softwareverwaltung	879
47.1 Softwareinventarisierung	879
47.2 Installation von Anwendungen	882
47.3 Deinstallation von Anwendungen	883
47.4 Praxislösung: Installationstest	884
47.5 Praxislösung: Installierte .NET SDKs aufräumen	885
47.6 Windows 10 Apps verwalten	889
47.7 Installationen mit PowerShell Package Management („OneGet“)	892
47.8 Versionsnummer ermitteln	895
47.9 Servermanager	896
47.10 Windows-Features installieren auf Windows-Clientbetriebssystemen	907
47.11 Praxislösung: IIS-Installation	909
47.12 Softwareeinschränkungen mit dem PowerShell-Modul „AppLocker“	911
48 Prozessverwaltung	917
48.1 Prozesse auflisten	917
48.2 Prozesse starten	918
48.3 Prozesse mit vollen Administratorrechten starten	919
48.4 Prozesse unter einem anderen Benutzerkonto starten	920
48.5 Prozesse beenden	921
48.6 Warten auf das Beenden einer Anwendung	922

49 Windows-Systemdienste	923
49.1 Dienste auflisten	923
49.2 Dienstzustand ändern	926
49.3 Diensteigenschaften ändern	926
49.4 Dienste hinzufügen	927
49.5 Dienste entfernen	928
50 Netzwerk	929
50.1 Netzwerkkonfiguration	929
50.2 DNS-Client-Konfiguration	934
50.3 DNS-Namensauflösung	938
50.4 Erreichbarkeit prüfen (Ping)	939
50.5 Windows Firewall	940
50.6 Remote Desktop (RDP) einrichten	947
50.7 E-Mails senden (SMTP)	948
50.8 Auseinandernehmen von E-Mail-Adressen	949
50.9 Abruf von Daten von einem HTTP-Server	949
50.10 Praxislösung: Linkprüfer für eine Website	956
50.11 Aufrufe von SOAP-Webservices	959
50.12 Aufruf von REST-Diensten	962
50.13 File Transfer Protocol (FTP)	964
50.14 Hintergrunddatentransfer mit BITS	965
51 Ereignisprotokolle (Event Log)	969
51.1 Protokolleinträge auslesen	969
51.2 Ereignisprotokolle erzeugen	971
51.3 Protokolleinträge erzeugen	971
51.4 Protokollgröße festlegen	971
51.5 Protokolleinträge löschen	971
52 Leistungsdaten (Performance Counter)	972
52.1 Zugriff auf Leistungsindikatoren über WMI	972
52.2 Get-Counter	973
53 Sicherheitseinstellungen	975
53.1 Aktueller Benutzer	975
53.2 Grundlagen	976
53.3 Zugriffsrechitelisten auslesen	981
53.4 Einzelne Rechteeinträge auslesen	982
53.5 Besitzer auslesen	984
53.6 Benutzer und SID	984

53.7 Hinzufügen eines Rechteeintrags zu einer Zugriffsrechiteliste	988
53.8 Entfernen eines Rechteeintrags aus einer Zugriffsrechiteliste	990
53.9 Zugriffsrechiteliste übertragen	992
53.10 Zugriffsrechiteliste über SDDL setzen	993
53.11 Zertifikate verwalten	994
54 Optimierungen und Problemlösungen	997
54.1 PowerShell-Modul „TroubleshootingPack“	997
54.2 PowerShell-Modul „Best Practices“	1001
55 Active Directory	1003
55.1 Benutzer- und Gruppenverwaltung mit WMI	1005
55.2 Einführung in System.DirectoryServices	1005
55.3 Basiseigenschaften	1017
55.4 Benutzer- und Gruppenverwaltung im Active Directory	1019
55.5 Verwaltung der Organisationseinheiten	1027
55.6 Suche im Active Directory	1028
55.7 Navigation im Active Directory mit den PowerShell Extensions	1035
55.8 Verwendung der Active-Directory-Erweiterungen von <i>www.IT-Visions.de</i>	1036
55.9 PowerShell-Modul „Active Directory“ (ADPowerShell)	1038
55.10 PowerShell-Modul „ADDSDeployment“	1067
55.11 Informationen über die Active Directory-Struktur	1070
56 Gruppenrichtlinien	1073
56.1 Verwaltung der Gruppenrichtlinien	1073
56.2 Verknüpfung der Gruppenrichtlinien	1075
56.3 Gruppenrichtlinienberichte	1077
56.4 Gruppenrichtlinienvererbung	1079
56.5 Weitere Möglichkeiten	1080
57 Lokale Benutzer und Gruppen	1081
57.1 Modul „Microsoft.PowerShell.LocalAccounts“	1081
57.2 Lokale Benutzerverwaltung in älteren PowerShell-Versionen	1082
58 Microsoft Exchange Server	1085
58.1 Daten abrufen	1085
58.2 Postfächer verwalten	1086
58.3 Öffentliche Ordner verwalten	1087
59 Internet Information Services (IIS)	1088
59.1 Überblick	1088
59.2 Navigationsprovider	1090

59.3	Anlegen von Websites	1092
59.4	Praxislösung: Massenanlegen von Websites	1093
59.5	Ändern von Website-Eigenschaften	1095
59.6	Anwendungspool anlegen	1096
59.7	Virtuelle Verzeichnisse und IIS-Anwendungen	1097
59.8	Website-Zustand ändern	1097
59.9	Anwendungspools starten und stoppen	1098
59.10	Löschen von Websites	1098
60	Virtuelle Systeme mit Hyper-V	1099
60.1	Das Hyper-V-Modul von Microsoft	1100
60.2	Die ersten Schritte mit dem Hyper-V-Modul	1102
60.3	Virtuelle Maschinen anlegen	1106
60.4	Umgang mit virtuellen Festplatten	1112
60.5	Konfiguration virtueller Maschinen	1115
60.6	Praxislösungen: Ressourcennutzung überwachen	1119
60.7	Dateien kopieren in virtuelle Systeme	1121
60.8	PowerShell Management Library for Hyper-V (für ältere Betriebssysteme)	1122
61	Windows Nano Server	1125
61.1	Das Konzept von Nano Server	1125
61.2	Einschränkungen von Nano Server	1127
61.3	Varianten des Nano Servers	1129
61.4	Installation eines Nano Servers	1129
61.5	Docker-Image	1130
61.6	Fernverwaltung mit PowerShell	1131
61.7	Windows Update auf einem Nano Server	1133
61.8	Nachträgliche Paketinstallation	1133
61.9	Abgespeckter IIS unter Nano Server	1135
61.10	Nano-Serververwaltung aus der Cloud heraus	1136
62	Docker-Container	1137
62.1	Container-Varianten für Windows	1137
62.2	Docker-Installation auf aktuellem Windows 10 und Windows 11	1141
62.3	Docker-Installation auf älteren Windows 10-Clients	1149
62.4	Docker-Installation auf Windows Server	1151
62.5	Docker PowerShell installieren	1153
62.6	Docker-Basiswissen	1154
62.7	Container mit modernem .NET	1157
62.8	Container mit IIS-Webserver und klassischem ASP.NET	1166
62.9	Container mit Linux und PowerShell 7	1175

62.10 Container mit Linux und Microsoft SQL Server	1177
62.11 Docker-Container mit Visual Studio	1179
62.12 Weitere Container-Befehle	1184
63 Microsoft Azure	1190
63.1 Azure-Konzepte	1190
63.2 Kommandozeilenwerkzeuge für die Azure-Verwaltung	1192
63.3 Benutzeranmeldung und Informationsabfrage	1195
63.4 Azure Ressourcen-Gruppen	1196
63.5 Azure Web-Apps	1196
63.6 Azure SQL Server	1198
63.7 Azure Kubernetes Services (AKS)	1199
63.8 Azure DevOps (ADO)	1223
64 Grafische Benutzeroberflächen (GUI)	1244
64.1 Einfache Nachfragedialoge	1244
64.2 Einfache Eingabe mit Inputbox	1245
64.3 Komplexere Eingabemasken	1246
64.4 Universelle Objektdarstellung	1248
64.5 WPF PowerShell Kit (WPK)	1249
64.6 Direkte Verwendung von WPF	1257
Teil D: Profiwissen – Erweitern der PowerShell	1259
65 Unit Tests mit Pester	1261
65.1 Einführung in das Konzept des Unit Testing	1261
65.2 Pester installieren	1262
65.3 Befehle in Pester	1262
65.4 Testen einer PowerShell-Funktion	1263
65.5 Testgenerierung	1264
65.6 Tests starten	1264
65.7 Prüf-Operationen	1266
65.8 Mock-Objekte	1266
65.9 Test von Dateisystemoperationen	1267
66 Entwicklung von Commandlets in der PowerShell-Skriptsprache	1269
66.1 Aufbau eines skriptbasierten Commandlets	1269
66.2 Verwendung per Dot Sourcing	1271
66.3 Parameterfestlegung	1272
66.4 Fortgeschrittene Funktion (Advanced Function)	1278

66.5	Mehrere Parameter und Parametersätze	1281
66.6	Unterstützung für Sicherheitsabfragen (-whatif und -confirm)	1283
66.7	Kaufmännisches Beispiel: Test-CustomerID	1285
66.8	Erweitern bestehender Commandlets durch Proxy-Commandlets	1288
66.9	Dokumentation	1294
67	Entwicklung eigener Commandlets mit C#	1298
67.1	Technische Voraussetzungen	1299
67.2	Grundkonzept der .NET-basierten Commandlets	1301
67.3	Schrittweise Erstellung eines minimalen Commandlets	1303
67.4	Erstellung eines Commandlets mit einem Rückgabeobjekt	1311
67.5	Erstellung eines Commandlets mit mehreren Rückgabeobjekten	1313
67.6	Erstellen eines Commandlets mit Parametern	1317
67.7	Verarbeiten von Pipeline-Eingaben	1319
67.8	Verkettung von Commandlets	1322
67.9	Fehlersuche in Commandlets	1326
67.10	Statusinformationen	1329
67.11	Unterstützung für Sicherheitsabfragen (-whatif und -confirm)	1334
67.12	Festlegung der Hilfeinformationen	1336
67.13	Erstellung von Commandlets für den Zugriff auf eine Geschäftsanwendung ...	1341
67.14	Konventionen für Commandlets	1342
67.15	Weitere Möglichkeiten	1344
68	PowerShell-Module erstellen	1345
68.1	Erstellen eines Skriptmoduls	1345
68.2	Praxislösung: Umwandlung einer Skriptdatei in ein Modul	1347
68.3	Erstellen eines Moduls mit Binärdateien	1347
68.4	Erstellen eines Moduls mit Manifest	1348
68.5	Erstellung eines Manifest-Moduls mit Visual Studio	1355
69	Hosting der PowerShell	1357
69.1	Voraussetzungen für das Hosting	1358
69.2	Hosting mit PSHost	1359
69.3	Vereinfachtes Hosting seit PowerShell 2.0	1362
Anhang A: Crashkurs Objektorientierung		1365
Anhang B: Crashkurs .NET		1373
B.1	Was ist das .NET Framework?	1376
B.2	Was ist .NET Core/.NET?	1377
B.3	Eigenschaften von .NET	1378

B.4	.NET-Klassen	1379
B.5	Namensgebung von .NET-Klassen (Namensräume)	1379
B.6	Namensräume und Softwarekomponenten	1381
B.7	Bestandteile einer .NET-Klasse	1382
B.8	Vererbung	1383
B.9	Schnittstellen	1383
Anhang C: Weitere Informationen im Internet		1384
Anhang D: Abkürzungsverzeichnis		1385
Stichwortverzeichnis		1409

Vorwort

Liebe Leserin, lieber Leser,

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

Was ist das Thema dieses Buchs?

Das vor Ihnen liegende Fachbuch behandelt die Windows PowerShell in der Version 5.1 sowie die plattformneutrale PowerShell 7.4 von Microsoft wie auch ergänzende Werkzeuge von Microsoft und Drittanbietern (z. B. PowerShell Community Extensions).

Das Buch ist aber auch für Sie geeignet, wenn Sie noch eine ältere Version der PowerShell einsetzen. Welche Funktionen neu hinzugekommen sind, wird jeweils in diesem Buch erwähnt.

Wer bin ich?

Mein Name ist Holger Schwichtenberg, ich bin derzeit 51 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 Software im Auftrag von Kunden in zahlreichen Branchen.

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

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

Wer sind Sie?

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

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

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

Was ist neu in diesem Buch?

Die vorliegende Auflage wurde auf PowerShell Version 7.4 aktualisiert und bestehende Inhalte des Buchs an vielen Stellen optimiert. Zudem wurde das Feedback einiger Leser eingearbeitet, um Beispiele und Texte weiter zu verbessern.

Sind in diesem Buch alle Features der PowerShell beschrieben?

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

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

Wie aktuell ist dieses Buch?

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

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



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

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

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

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

Welche PowerShell-Versionen werden besprochen?

Das Buch bespricht sowohl die Windows PowerShell 5.1 als auch die PowerShell 7.4.

- Bei der Windows PowerShell 5.1 wird die RTM-Version besprochen, die Microsoft in der aktuellen Version von Windows 10/11 bzw. Windows Server 2019/2022 mitliefert.
- Bei PowerShell 7.4 wird die RTM-Version vom 16. November 2023 behandelt.

Warum behandelt das Buch auch noch Version 5.1 und nicht nur Version 7.4?

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

Die PowerShell 7.4 wird bisher mit keinem einzigen Betriebssystem ausgeliefert, sondern muss getrennt heruntergeladen und installiert werden. Eine Zusatzinstallation ist in vielen Unternehmen mit stark abgeschotteten Systemen gar nicht möglich.

Ein zweites Argument für die Beibehaltung der Version 5.1 in diesem Fachbuch ist, dass die PowerShell 7.4 der Windows PowerShell 5.1 funktional immer noch nicht ganz ebenbürtig ist. Einige Befehle sind weiterhin nur in der Windows PowerShell verfügbar.

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

Welche Betriebssysteme werden besprochen?

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

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

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

Woher bekommt man die Beispiele aus diesem Buch?

Unter <http://www.powershell-doktor.de/leser> biete ich ein **ehrenamtlich betriebenes** Web-portal für Leser meiner Bücher an. Bei der Erstregistrierung müssen Sie das Losungswort **Sektion31** angeben. Nach erfolgter Registrierung erhalten Sie dann ein persönliches Zugangs-kennwort per E-Mail.

In diesem Portal können Sie

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

Über den Autor

- Studienabschluss Diplom-Wirtschaftsinformatik an der Universität Essen
- Promotion an der Universität Essen im Fachgebiet komponentenbasierter Softwareentwicklung
- Seit 1996 in der IT tätig als Softwareentwickler, Softwarearchitekt, Berater, Dozent und Fachjournalist
- Fachlicher Leiter des Expertenteams bei www.ITVisions.de in Essen
- Über 90 Fachbücher bei verschiedenen Verlagen, u. a. Carl Hanser Verlag, O'Reilly, APress, Microsoft Press, Addison Wesley sowie im Selbstverlag
- Mehr als 1500 Beiträge in Fachzeitschriften und Online-Portalen
- Gutachter in den Wettbewerbsverfahren der EU gegen Microsoft (2006 – 2009)
- Ständiger Mitarbeiter der Zeitschriften iX (seit 1999), dotnetpro (seit 2000) und Windows Developer (seit 2010) sowie beim Online-Portal heise.de (seit 2008)
- Regelmäßiger Sprecher auf nationalen und internationalen Fachkonferenzen (z. B. enterJS, BASTA!, Microsoft TechEd, Microsoft Summit, Microsoft IT Forum, OOP, IT Tage, .NET Architecture Camp, Advanced Developers Conference, Developer Week, DOTNET Cologne, MD DevDays, Community in Motion, DOTNET-Konferenz, VS One, NRW.Conf, Net.Object Days, Windows Forum, Container Conf)
- Zertifikate und Auszeichnungen von Microsoft:
 - Microsoft Most Valuable Professional (MVP), kontinuierlich ausgezeichnet seit 2004
 - Microsoft Certified Solution Developer (MCSD)
- Thematische Schwerpunkte:
 - Softwarearchitektur, mehrschichtige Softwareentwicklung, Softwarekomponenten
 - Visual Studio, Continuous Integration (CI) und Continuous Delivery (CD) mit Azure DevOps
 - Microsoft .NET (.NET Framework, .NET Core), C#, Visual Basic
 - .NET-Architektur, Auswahl von .NET-Techniken
 - Einführung von .NET, Migration auf .NET



www.IT-Visions.de
Dr. Holger Schwichtenberg

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

**HINWEIS:**

- Kontakt für Anfragen zu Schulung und Beratung sowie Softwareentwicklungsarbeiten:
kundenteam@IT-Visions.de, Telefon 0201/64 95 90 – 50
- Kontakt für Feedback zu diesem Buch:
www.dotnet-doktor.de/Leserfeedback

5

Objektorientiertes Pipelining

Ihre Mächtigkeit entfaltet die PowerShell erst durch das objektorientierte Pipelining, also durch die Weitergabe von strukturierten Daten von einem Commandlet zum anderen.



HINWEIS: Dieses Kapitel setzt ein Grundverständnis des Konzepts der Objektorientierung voraus. Wenn Sie diese Grundkenntnisse nicht besitzen, lesen Sie bitte zuvor im Anhang den Crashkurs „Objektorientierung“ sowie den Crashkurs „.NET Framework“ oder vertiefende Literatur.

■ 5.1 Befehlsübersicht

Die folgende Tabelle zeigt eine Übersicht der wichtigsten Commandlets, die Basisoperationen auf Pipelines ausführen. Diese Commandlets werden in den folgenden Kapiteln genau besprochen.

Tabelle 5.1 Übersicht über die wichtigsten Pipelining-Commandlets

Commandlet (mit Aliasen)	Bedeutung
Where-Object (where, ?)	Filtern mit Bedingungen
Select-Object (select)	Abschneiden der Ergebnismenge vorne/hinten bzw. Reduktion der Attribute der Objekte. Auch: Eliminieren von Duplikaten
Sort-Object (sort)	Sortieren der Objekte
Group-Object (group)	Gruppieren der Objekte
Foreach-Object { \$... } (%)	Schleife über alle Objekte. Der Befehlsblock { ... } wird für jedes Objekt in der Pipeline einmal ausgeführt.
Get-Member (gm)	Ausgabe der Metadaten (Reflection)
Measure-Object (measure)	Berechnung: -min -max -sum -average
Compare-Object (compare, diff)	Vergleichen von zwei Objektmengen

■ 5.2 Pipeline-Operator

Für eine Pipeline wird – wie auch in Unix-Shells üblich und in der normalen Windows-Konsole möglich – der vertikale Strich „|“ (genannt „Pipe“ oder „Pipeline Operator“) verwendet.

```
Get-Process | Format-List
```

bedeutet, dass das Ergebnis des `Get-Process`-Commandlets an `Format-List` weitergegeben werden soll. Die Standardausgabeform von `Get-Process` ist eine Tabelle. Durch `Format-List` werden die einzelnen Attribute der aufzulistenden Prozesse untereinander statt in Spalten ausgegeben.

Die Pipeline kann beliebig lang sein, d.h., die Anzahl der Commandlets in einer einzigen Pipeline ist nicht begrenzt. Man muss aber jedes Mal den Pipeline-Operator nutzen, um die Commandlets zu trennen.

Ein Beispiel für eine komplexere Pipeline lautet:

```
Get-ChildItem w:\daten -r -filter *.doc  
| Where-Object { $_.Length -gt 40000 }  
| Select-Object Name, Length  
| Sort-Object Length  
| Format-List
```

`Get-ChildItem` ermittelt alle Microsoft-Word-Dateien im Ordner `w:\daten` und in seinen Unterordnern. Durch das zweite Commandlet (`Where-Object`) wird die Ergebnismenge auf diejenigen Objekte beschränkt, bei denen das Attribut `Length` größer ist als 40 000. `$`_ ist dabei der Zugriff auf das aktuelle Objekt in der Pipeline. Der Ausdruck `$_.Length -gt 40000` ruft aus dem aktuellen Objekt die Eigenschaft `Length` ab und vergleicht, ob diese größer (-gt) als 40 000 ist. `Select-Object` beschneidet alle Attribute aus `Name` und `Length`. Durch das vierte Commandlet in der Pipeline wird die Ausgabe nach dem Attribut `Length` sortiert. Das letzte Commandlet schließlich erzwingt eine Listendarstellung.

Nicht alle Aneinanderreihungen von Commandlets ergeben einen Sinn. Einige Aneinanderreihungen sind auch gar nicht erlaubt. Die Reihenfolge der einzelnen Befehle in der Pipeline ist nicht beliebig. Keineswegs kann man im obigen Befehl die Sortierung hinter die Formatierung setzen, weil nach dem Formatieren zwar noch ein Objekt existiert, dieses aber einen Textstrom repräsentiert. `Where-Object` und `Sort-Object` könnte man vertauschen; aus Gründen des Ressourcenverbrauchs sollte man aber erst einschränken und dann die verringerte Liste sortieren. Ein Commandlet kann aus vorgenannten Gründen erwarten, dass es bestimmte Arten von Eingabeobjekten gibt. Am besten sind aber Commandlets, die jede Art von Eingabeobjekt verarbeiten können.

Eine automatische Optimierung der Befehlsfolge wie in der Datenbankabfrage SQL gibt es bei PowerShell nicht.

Seit PowerShell-Version 3.0 hat Microsoft für den Zugriff auf das aktuelle Objekt der Pipeline zusätzlich zum Ausdruck `$`_ den Ausdruck `$PSItem` eingeführt. `$`_ und `$PSItem` sind synonym. Microsoft hat `$PSItem` eingeführt, weil einige Benutzer das Feedback gaben, dass `$`_ zu (Zitat) „magisch“ sei.



ACHTUNG: Die PowerShell erlaubt beliebig lange Pipelines und es gibt auch Menschen, die sich einen Spaß daraus machen, möglichst viel durch eine einzige Befehlsfolge mit sehr vielen Pipes auszudrücken. Solche umfangreichen Befehlsfolgen sind aber meist für andere Menschen extrem schlecht lesbar. Bitte befolgen Sie daher den folgenden Ratschlag: Schreiben Sie nicht alles in eine einzige Befehlsfolge, nur weil es geht. Teilen Sie besser die Befehlsfolgen nach jeweils drei bis vier Pipe-Symbolen durch den Einsatz von Variablen auf (wird in diesem Kapitel auch beschrieben!) und lassen Sie diese geteilten Befehlsfolgen dann besser als PowerShell-Skripte ablaufen (siehe das Kapitel „PowerShell-Skripte“).

■ 5.3 .NET-Objekte in der Pipeline

Objektorientierung ist die herausragende Eigenschaft der PowerShell: Commandlets können durch Pipelines mit anderen Commandlets verbunden werden. Anders als Pipelines in Unix-Shells tauschen die Commandlets der PowerShell keine Zeichenketten, sondern typisierte .NET-Objekte aus. Das objektorientierte Pipelining ist im Gegensatz zum in den Unix-Shells und in der normalen Windows-Shell (*cmd.exe*) verwendeten zeichenkettenbasierten Pipelining nicht abhängig von der Position der Informationen in der Pipeline.

Ein Commandlet kann auf alle Attribute und Methoden der .NET-Objekte, die das vorhergehende Commandlet in die Pipeline gelegt hat, zugreifen. Die Mitglieder der Objekte können entweder durch Parameter der Commandlets (z.B. in `Sort-Object Length`) oder durch den expliziten Verweis auf das aktuelle Pipeline-Objekt (`$_.Length`) in einer Schleife oder Bedingung (z.B. `Where-Object { $_.Length -gt 40000 }`) genutzt werden.

In einer Pipeline wie

```
Get-Process | Where-Object {$_.Name -eq "iexplore"} | Format-Table ProcessName,  
WorkingSet64
```

ist das dritte Commandlet daher nicht auf eine bestimmte Anordnung und Formatierung der Ausgabe von vorherigen Commandlets angewiesen, sondern es greift über den sogenannten Reflection-Mechanismus (den eingebauten Komponentenerforschungsmechanismus des .NET Frameworks) direkt auf die Eigenschaften der Objekte in der Pipeline zu.



HINWEIS: Genau genommen bezeichnet Microsoft das Verfahren als „Extended Reflection“ bzw. „Extended Type System (ETS)“, weil die PowerShell in der Lage ist, Objekte um zusätzliche Eigenschaften anzureichern, die in der Klassendefinition gar nicht existieren.

Im obigen Beispiel legt `Get-Process` ein .NET-Objekt der Klasse `System.Diagnostics.Process` für jeden laufenden Prozess in die Pipeline. `System.Diagnostics.Process` ist eine Klasse aus der .NET-Klassenbibliothek. Commandlets können aber jedes beliebige .NET-Objekt in die Pipeline legen, also auch einfache Zahlen oder Zeichenketten, da es in .NET

keine Unterscheidung zwischen elementaren Datentypen und Klassen gibt. Eine Zeichenkette in die Pipeline zu legen, wird aber in der PowerShell die Ausnahme bleiben, denn der typisierte Zugriff auf Objekte ist wesentlich robuster gegenüber möglichen Änderungen als die Zeichenkettenauswertung mit regulären Ausdrücken.

Deutlicher wird der objektorientierte Ansatz, wenn man als Attribut keine Zeichenkette heranzieht, sondern eine Zahl. `WorkingSet64` ist ein 64 Bit langer Zahlenwert, der den aktuellen Speicherverbrauch eines Prozesses repräsentiert. Der folgende Befehl liefert alle Prozesse, die aktuell mehr als 20 Megabyte verbrauchen:

```
Get-Process | Where-Object {$_.WorkingSet64 -gt 20*1024*1024 }
```

Anstelle von `20*1024*1024` hätte man auch das Kürzel „`20MB`“ einsetzen können. Außerdem kann man `Where-Object` mit einem Fragezeichen abkürzen. Die kurze Variante des Befehls wäre dann also:

```
ps | ? {$_.ws -gt 20MB }
```

Wenn nur ein einziges Commandlet angegeben ist, dann wird das Ergebnis auf dem Bildschirm ausgegeben. Auch wenn mehrere Commandlets in einer Pipeline zusammengeschaltet sind, wird das Ergebnis des letzten Commandlets auf dem Bildschirm ausgegeben. Wenn das letzte Commandlet keine Daten in die Pipeline wirft, erfolgt keine Ausgabe.

■ 5.4 Pipeline Processor

Für die Übergabe der .NET-Objekte zwischen den Commandlets sorgt der *PowerShell Pipeline Processor* (siehe folgende Grafik). Die Commandlets selbst müssen sich weder um die Objektweitergabe noch um die Parameterauswertung kümmern.

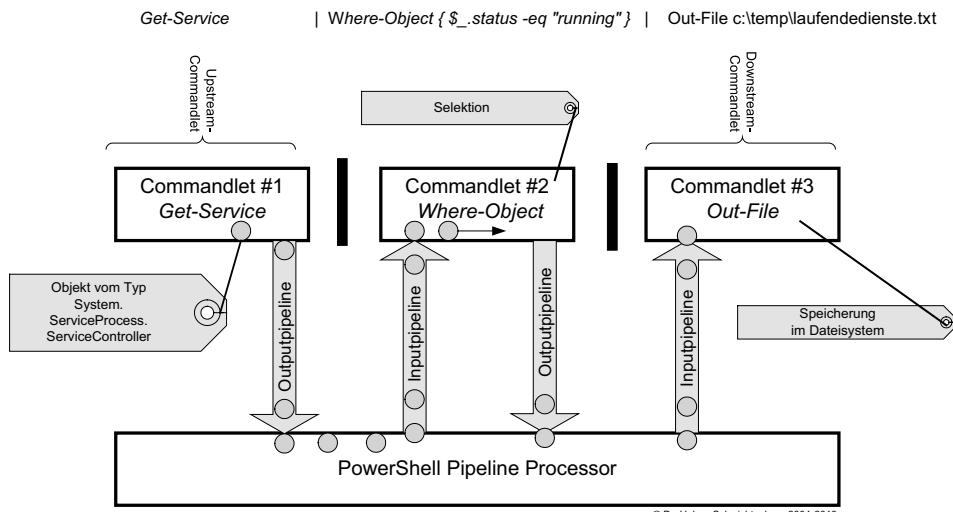


Bild 5.1 Der Pipeline Processor befördert die Objekte vom Upstream-Commandlet zum Downstream-Commandlet. Die Verarbeitung ist in der Regel asynchron.

Wie das obige Bild schon zeigt, beginnt ein nachfolgendes Commandlet mit seiner Arbeit, sobald es ein erstes Objekt aus der Pipeline erhält. Das Objekt durchläuft die komplette Pipeline. Erst dann wird das nächste Objekt vom ersten Commandlet abgeholt. Man nennt dies „Streaming-Verarbeitung“. Streaming-Verarbeitung ist schneller als die klassische sequentielle Verarbeitung, weil die folgenden Commandlets in der Pipeline nicht auf vorhergehende warten müssen.



HINWEIS: Intern arbeitet die einem Thread, d.h. es findet keine parallele Verarbeitung mehrerer Befehle statt. Erst seit PowerShell 7.0 gibt es mit dem Parameter -parallel bei Foreach-Command eine einfache Möglichkeit, jedes Objekt in einem eigenen Thread zu verarbeiten.

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



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

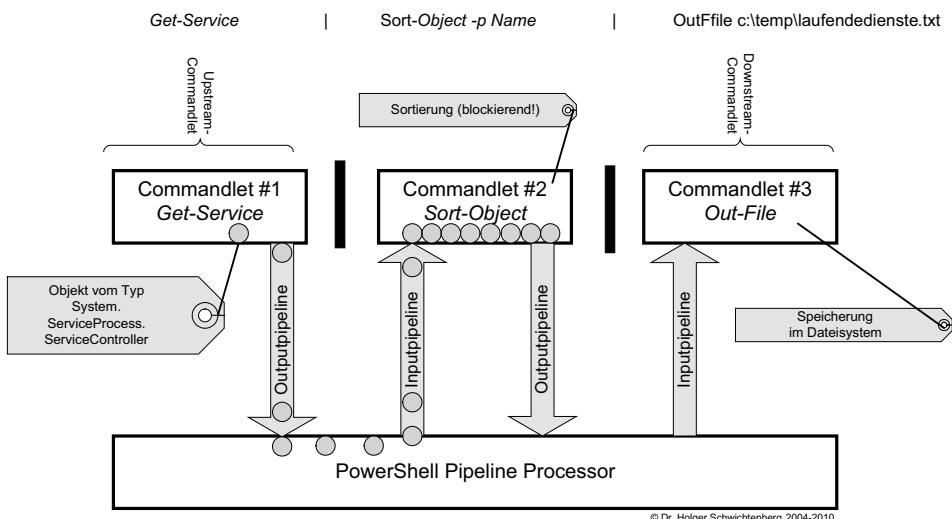


Bild 5.2 Sort-Object blockiert die direkte Weitergabe. Erst wenn alle Objekte angekommen sind, kann das Commandlet sortieren.

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

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

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

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

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

■ 5.5 Pipelining von Parametern

Die Pipeline kann jegliche Art von Information befördern, auch einzelne elementare Daten. Einige Commandlets unterstützen es, dass auch die Parameter aus der Pipeline ausgelesen werden. Der folgende Pipeline-Befehl führt zu einer Auflistung aller Windows-Systemdienste, die mit dem Buchstaben „I“ beginnen.

```
"i*" | Get-Service
```

Die folgende Abbildung zeigt einige Parameter des Commandlets `Get-Service`. Diese Liste erhält man durch den Befehl `Get-Help Get-Service -Parameter *`.

```
-Include <string[]>
    Retrieves only the specified services. The value of this parameter qualifies the Name parameter. Enter a name element or pattern, such as "s*". Wildcards are permitted.

    Required?          false
    Position?         named
    Default value
    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
    Accept pipeline input?   false
    Accept wildcard characters? false
```

Bild 5.3 Hilfe zu den Parametern des Commandlets `Get-Service`

Interessant sind die mit Pfeil markierten Stellen. Nach „Accept pipeline Input“ kann man jeweils nachlesen, ob der Parameter des Commandlets aus den vorhergehenden Objekten in der Pipeline „befüttert“ werden kann.

Bei „-Name“ steht `ByValue` und `ByPropertyName`. Dies bedeutet, dass der Name sowohl das ganze Objekt in der Pipeline sein darf als auch Teil eines Objekts.

Im Fall von

```
"BITS" | Get-Service
```

ist der Pipeline-Inhalt eine Zeichenkette (ein Objekt vom Typ `String`), die als Ganzes auf `Name` abgebildet werden kann.

Es funktioniert aber auch folgender Befehl, der alle Dienste ermittelt, deren Name genauso lautet wie der Name eines laufenden Prozesses:

```
Get-Process | Get-Service -ea silentlycontinue | ft name
```

Dies funktioniert über die zweite Option (`ByPropertyName`), denn `Get-Process` liefert Objekte des Typs `Process`, die ein Attribut namens `Name` haben. Der Parameter `Name` von `Get-Service` wird auf dieses `Name`-Attribut abgebildet.

Beim Parameter `-InputObject` ist hingegen nur „`ByValue`“ angegeben. Hier erwartet `Get-Service` gerne Instanzen der Klasse `ServiceController`. Es gibt aber keine Objekte, die ein Attribut namens `InputObject` haben, in dem dann `ServiceController`-Objekte stecken.

Zahlreiche Commandlets besitzen einen Parameter `-InputObject`, insbesondere die allgemeinen Verarbeitungs-Commandlets wie `Where-Object`, `Select-Object` und `Measure-Object`, die Sie im nächsten Kapitel kennenlernen werden. Der Name `-InputObject` ist eine Konvention.

```
PS P:\> Get-Help Where-Object -Parameter *
-FilterScript <scriptblock>
    Specifies the script block that is used to filter the objects. Enclose the
    script block in braces < {} >.

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

-InputObject <psoobject>
    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`, dass prüft, ob ein Computer per Ping erreichbar ist.

Der normale Aufruf mit Parameter ist:

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

oder ohne benannten Parameter

```
Test-Connection Server123
```

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

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

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

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

```
-ComputerName <string[]>
  Required?          true
  Position?         0
  Accept pipeline input?    true (ByPropertyName) →
  Parameter set name   (All)
  Aliases            CN, IPAddress, __SERVER, Server, Destination
  Dynamic?          false
```

Bild 5.5 Hilfe zum Parameter ComputerName des Commandlets Test-Connection

■ 5.6 Pipelining von klassischen Befehlen

Grundsätzlich dürfen auch klassische Kommandozeilenanwendungen in der PowerShell verwendet werden. Wenn man einen Befehl wie netstat.exe oder ping.exe ausführt, dann legen diese eine Menge von Zeichenketten in die Pipeline: Jede Ausgabezeile ist eine Zeichenkette.

Diese Zeichenketten kann man sehr gut mit dem Commandlet Select-String auswerten. Select-String lässt nur diejenigen Zeilen die Pipeline passieren, die auf den angegebenen regulären Ausdruck zutreffen.



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

In dem folgenden Beispiel werden nur diejenigen Zeilen der Ausgabe von `netstat.exe` gefiltert, die einen Doppelpunkt gefolgt von den Ziffern 59 und zwei weiteren Ziffern enthalten. Die Hervorhebung der Treffer durch Negativschrift gibt es erst seit PowerShell 7.0.

```
PS X:\> netstat | select-string ":59\d\d" -case
TCP    127.0.0.1:5905      E60:49675      ESTABLISHED
TCP    127.0.0.1:5905      E60:49677      ESTABLISHED
TCP    127.0.0.1:5905      E60:49678      ESTABLISHED
TCP    127.0.0.1:5905      E60:57897      ESTABLISHED
TCP    127.0.0.1:5905      E60:57898      ESTABLISHED
TCP    127.0.0.1:5905      E60:57899      ESTABLISHED
TCP    127.0.0.1:5905      E60:57920      ESTABLISHED
TCP    127.0.0.1:5905      E60:57921      ESTABLISHED
TCP    127.0.0.1:49675     E60:5905      ESTABLISHED
TCP    127.0.0.1:49677     E60:5905      ESTABLISHED
TCP    127.0.0.1:49678     E60:5905      ESTABLISHED
TCP    127.0.0.1:57897     E60:5905      ESTABLISHED
TCP    127.0.0.1:57898     E60:5905      ESTABLISHED
TCP    127.0.0.1:57899     E60:5905      ESTABLISHED
TCP    127.0.0.1:57920     E60:5905      ESTABLISHED
TCP    127.0.0.1:57921     E60:5905      ESTABLISHED
PS X:\>
```

Bild 5.6

Einsatz von `Select-String` zur Filterung von Ausgaben klassischer Kommandozeilenwerkzeuge

Ein weiteres Beispiel ist das Filtern der Ausgaben von `ipconfig.exe`. Der nachfolgende Befehl liefert nur die Zeilen zum Thema IPv4:

```
ipconfig.exe /all | select-string IPV4
```

```
PS X:\> ipconfig.exe /all | select-string IPV4
IPv4 Address. . . . . : 192.168.1.60(Preferred)
IPv4 Address. . . . . : 172.17.24.113(Preferred)
PS X:\>
```

Bild 5.7

Abbildung: Ausführung des obigen Befehls

Es gibt aber leider klassische Kommandozeilenbefehle, die inhaltliche Informationen über Farben statt über Texte transportieren. Ein schlechtes Beispiel ist hier:

```
git branch -a
```

Der Befehl `git branch -a` liefert eine Liste aller Git-Branches in einem lokalen Git-Repository als farblich verschieden markierte Textzeilen.

```
T:\CC2 [master *] > git branch -a
* master
  remotes/GITHUB/Feature1
  remotes/GITHUB/master
  remotes/GITHUB/F2
  remotes/GITHUB/Feature1
  remotes/GITHUB/Feature2
  remotes/GITHUB/Feature3
  remotes/GITHUB/HEAD -> GITHUB/master
  remotes/GITHUB/master
```

Eine schwarze Ausgabe (erste beide Zeilen) bedeutet, dass es für den Remote-Branch auch einen lokalen Branch gibt. Eine rote Ausgabe (Zeile 3 bis 8, hier im Buch aufgrund des Schwarz-Weiß-Drucks leider nicht zu sehen) bedeutet dabei, dass ein Remote-Branch noch kein lokales Äquivalent besitzt.

Man kann diesen Befehl zwar in der PowerShell ausführen und sieht dort auch die Farben. Aber eine Weiterverarbeitung per Pipeline mit dem Ziel „Lege einen lokalen Branch an für alle Branches, die lokal noch nicht existieren“, ist nicht möglich.

Man kann lediglich `git branch` für alle ausführen. Hierbei muss man nicht nur filtern, sondern auch mit `Trim()` die Leerzeichen zu Beginn eliminieren:

```
git branch -a | ? { $_ -like "*remotes*" -and $_ -notlike "*HEAD*" } | % { git branch
--track ${remote#origin/} $_.Trim() }
```

oder

```
git branch -a | sls -pattern "remotes" | sls -pattern "HEAD" -NotMatch | % { git
branch --track ${remote#origin/} $_.Line.Trim() }
```

Man bekommt aber immer eine Fehlermeldung für die schon existierenden lokalen Branches.

```
T:\CC2 [master =>] git branch -a | ? { $_ -like "*remotes*" -and $_ -notlike "*HEAD*" } | % { git branch --track ${remote#origin/} $_.Trim() }
fatal: A branch named 'remotes/GITHUB/feature1' already exists.
fatal: A branch named 'remotes/GITHUB/master' already exists.
Branch 'remotes/GITHUB/F2' set up to track local branch 'master'.
fatal: A branch named 'remotes/GITHUB/Feature1' already exists.
Branch 'remotes/GITHUB/Feature2' set up to track local branch 'master'.
Branch 'remotes/GITHUB/Feature3' set up to track local branch 'master'.
fatal: A branch named 'remotes/GITHUB/master' already exists.
```

■ 5.7 Zeilenumbrüche in Pipelines

Wenn sich ein Pipeline-Befehl über mehrere Zeilen erstrecken soll, kann man dies auf mehrere Weisen bewerkstelligen:

- Man beendet die Zeile mit einem Pipe-Symbol [`|`] und drückt EINGABE. PowerShell-Standardkonsole und PowerShell-ISE-Konsole erkennen, dass der Befehl noch nicht abgeschlossen ist, und erwarten weitere Eingaben. Die Standardkonsole zeigt dies auch mit `>>>` an.
- Man kann am Ende einer Zeile mit einem Gravis [```], ASCII-Code 96, bewirken, dass die nächste Zeile mit zum Befehl hinzugerechnet wird (Zeilenumbruch in einem Befehl). Das funktioniert in allen PowerShell-Hosts und auch in PowerShell-Skripten.

```
PS T:\> Get-Process p* | Sort-Object WorkingSet |
>> Format-Table id,Name,WorkingSet
   Id Name      WorkingSet
   -- -- --
10828 powershell    92942336
15340 powershell_ise 220946432
1804 powershell    83664896
4040 powershell    76177408
```

```
PS T:\> -
```

Bild 5.8
Zeilenumbruch nach
Pipeline-Symbol

■ 5.8 Schleifen

Ein wichtiges Commandlet ist

```
Foreach-Object { $_... }
```

Alias:

```
% { $_... }
```

Foreach-Object führt eine Schleife (Iteration) über alle Objekte in der Pipeline aus. Der Befehlsblock { ... } wird für jedes Objekt in der Pipeline einmal ausgeführt. Das jeweils aktuelle Objekt, das an der Reihe ist, erhält man über die eingebaute Variable `$_`. `$_` ist die Abkürzung für `$PSItem`. Beide Schreibweisen haben die gleiche Funktion.

5.8.1 Notwendigkeit für Foreach-Object

Der Einsatz von **Foreach-Object** ist in Pipelines nicht notwendig, wenn das nachfolgende Commandlet die Objekte des vorherigen Commandlets direkt verarbeiten kann.

Beispiele:

```
Get-ChildItem Bu* | Remove-Item
Get-Service BI* | Start-Service
Get-Process chrome | Stop-Process
```

Gleichwohl könnte man in diesen Fällen **Foreach-Object** einsetzen, was den Befehl aber verlängert:

```
Get-ChildItem Bu* | Foreach-Object { Remove-item $_.FullName }
Get-Service BI* | Foreach-Object { Start-Service $_ }
Get-Process chrome | Foreach-Object { Stop-Process $_ }
```

Es liegt an den Eigenarten des jeweiligen Commandlets, ob sie als Standardparameter das gesamte Objekt (`$_`) oder eine bestimmte Eigenschaft (`$_ FullName`) erwarten.

In manchen Situationen ist der Einsatz von **Foreach-Object** aber auch nicht möglich, denn man will mit **Sort-Object** die ganze Menge sortieren und nicht jedes Objekt einzeln:

```
"----- richtig:"
Get-Service x* | Sort-Object name
"----- falsch:"
Get-Service x* | Foreach-Object { Sort-Object $_.Name }
```

Schließlich gibt es Fälle, in denen **Foreach-Object** zwingend eingesetzt werden muss. Dies gilt insbesondere, wenn das nachfolgende Commandlet die Objekte nicht verarbeiten kann. Zudem quittiert die PowerShell diesen Befehl

```
Get-Service BI* | Write-Host $_.DisplayName -ForegroundColor yellow
```

mit dem Laufzeitfehler „The input object cannot be bound to any parameters for the command either because the command does not take pipeline input or the input and its properties do not“.

Richtig ist:

```
Get-Service BI* | foreach-object { Write-Host $_.DisplayName -ForegroundColor Yellow }
```

Ebenso ist `Foreach-Object` notwendig, wenn mehrere Befehle (also ganzer Befehlsblock) ausgeführt werden sollen. Befehlsblöcke werden in den Kapiteln „PowerShell-Skripte“ und „PowerShell-Skriptsache“ erläutert.

```
Get-Service BI* | foreach-object {
    if ($_.Status -eq "Stopped")
    {
        Write-Host "Beendet Dienst " $_.DisplayName -ForegroundColor Yellow
        Start-Service $_
    }
    else
    {
        Write-Host "Starte Dienst " $_.DisplayName -ForegroundColor Yellow
        Stop-Service $_
    }
}
```

5.8.2 Parallelisierung mit Multithreading

In PowerShell 1.0 bis 6.2 erfolgt die Ausführung im Hauptthread der PowerShell, d.h., die einzelnen Durchläufe erfolgen nacheinander. Seit PowerShell 7.0 kann man mit dem Parameter `-parallel` die Ausführung auf verschiedene Threads parallelisieren (via Multithreading), sodass bei längeren Operationen in Summe das Ergebnis schneller vorliegt.



ACHTUNG: Die Multithreading hat immer einigen Overhead. Die Parallelisierung lohnt sich nur bei länger dauernden Operationen. Bei kurzen Operationen ist der Zeitverlust durch die Erzeugung und Vernichtung der Threads höher als der Zeitgewinn durch die Parallelisierung.

Das folgende Beispiel zeigt zwei Varianten der Abfrage, ob die Software „Classic Shell“ auf drei verschiedenen Computern installiert ist. Bei der ersten Variante ohne `-parallel` wird die leider etwas langwierige Abfrage der WMI-Klasse `Win32_Product` auf den drei Computern nacheinander in dem gleichen Thread ausgeführt. Bei der zweiten Variante mit `-parallel` wird die Abfrage parallel in drei verschiedenen Threads gestartet! Die Parallelisierung ist erst möglich seit PowerShell 7.0.



TIPP: Die Nummer des Threads fragt man ab mit der .NET-Klasse `Thread`: `[System.Threading.Thread]::CurrentThread.ManagedThreadId`

Listing 5.1 [\PowerShell\1_Basiswissen\Pipelining\Schleifen.ps1]

```

Write-Host "# ForEach-Object ohne -parallel" -ForegroundColor Yellow
"E27","E29","E44" | ForEach-Object {
    "Abfrage bei Computer $_ in Thread $($([System.Threading.Thread]::CurrentThread.
ManagedThreadId))"
    $e = Get-CimInstance -Class Win32_
Product -Filter "Name='Classic Shell'" -computername $_
    if ($e -eq $null) { "Kein Ergebnis bei $_!"}
    else { $e }
}
Write-host ""
Write-host "# ForEach-Object mit -parallel" -ForegroundColor Yellow
"E27","E29","E44" | ForEach-Object -parallel {
    "Abfrage bei Computer $_ in Thread $($([System.Threading.Thread]::CurrentThread.
ManagedThreadId))"
    $e = Get-CimInstance -Class Win32_
Product -Filter "Name='Classic Shell'" -computername $_
    if ($e -eq $null) { "Kein Ergebnis bei $_!"}
    else { $e }
}
# ohne Read-
Host würde das Skript die später eingehenden Ergebnisse nicht mehr anzeigen!
read-host

```

# ForEach-Object ohne -parallel					
Abfrage bei Computer E27 in Thread 19					
Name	Caption	Vendor	Version	IdentifyingNumber	PSComputerName
Classic Shell	Classic Shell	IvoSoft	4.1.0	{840C85B7-D3D6-4143-9AF9-DAE80FD...}	E27
Abfrage bei Computer E29 in Thread 19					
Classic Shell	Classic Shell	IvoSoft	4.1.0	{840C85B7-D3D6-4143-9AF9-DAE80FD...}	E29
Abfrage bei Computer E44 in Thread 19					
Klein Ergebnis bei E44!					
# ForEach-Object mit -parallel					
Abfrage bei Computer E27 in Thread 80					
Name	Caption	Vendor	Version	IdentifyingNumber	PSComputerName
Classic Shell	Classic Shell	IvoSoft	4.1.0	{840C85B7-D3D6-4143-9AF9-DAE80FD...}	E29
Classic Shell	Classic Shell	IvoSoft	4.1.0	{840C85B7-D3D6-4143-9AF9-DAE80FD...}	E27

Bild 5.9 Parallelität bei Foreach-Object in PowerShell 7

Die Anzahl der Threads, die `ForEach-Object` nutzen soll, kann man mit dem Parameter `-ThrottleLimit` begrenzen:

```

1..20 | ForEach-Object -parallel {
    Write-host "Objekt #$_ in Thread $($([System.Threading.Thread]::CurrentThread.
ManagedThreadId)"
    sleep -Seconds 2 } -ThrottleLimit 5

```

■ 5.9 Zugriff auf einzelne Objekte aus einer Menge

Es ist möglich, gezielt einzelne Objekte über ihre Position (Index) in der Pipeline anzusprechen. Die Positionsangabe ist in eckige Klammern zu setzen und die Zählung beginnt bei 0. Der Pipeline-Ausdruck ist in runde Klammern zu setzen.

Beispiele:

Der erste Prozess:

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

Der dreizehnte Prozess:

```
(Get-Process) [12]
```

Alternativ kann man dies auch mit `Select-Object` unter Verwendung der Parameter `-First` und `-Skip` ausdrücken:

```
(Get-Process i* | Select-Object -first 1).name
(Get-Process i* | Select-Object -skip 12 -first 1).name
```



HINWEIS: Während `(Get-Date) [0]` in PowerShell vor Version 3.0 zu einem Fehler führt („Unable to index into an object of type System.DateTime.“), weil `Get-Date` keine Menge liefert, ist der Befehl seit PowerShell-Version 3.0 in Ordnung und liefert das gleiche Ergebnis wie `Get-Date`, da die PowerShell seit Version 3.0 ja aus Benutzersicht ein einzelnes Objekt und eine Menge von Objekten gleich behandelt. `(Get-Date) [1]` liefert dann natürlich kein Ergebnis, weil es kein zweites Objekt in der Pipeline gibt.

Die Positionsangaben kann man natürlich mit Bedingungen kombinieren. So liefert dieser Befehl den dreizehnten Prozess in der Liste der Prozesse, die mehr als 20 MB Hauptspeicher brauchen:

```
(Get-Process | where-object { $_.WorkingSet64 -gt 20mb } )[12]
```

PS C:\Windows\System32> <get-process>[0]							
Handles	NPM(K)	PM(K)	WS(K)	UM(M)	CPU(s)	Id	ProcessName
20	2	1968	2664	17	0,03	2784	cmd
PS C:\Windows\System32> <get-process>[12]							
Handles	NPM(K)	PM(K)	WS(K)	UM(M)	CPU(s)	Id	ProcessName
69	9	1484	4196	41	0,03	2100	dlpwdnt
PS C:\Windows\System32> <get-process where-object { \$_.WorkingSet64 -gt 20mb } >[12]							
Handles	NPM(K)	PM(K)	WS(K)	UM(M)	CPU(s)	Id	ProcessName
685	29	53924	59544	291	34,39	4984	powershell

Bild 5.10 Zugriff auf einzelne Prozessobjekte

■ 5.10 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 den Format-Commandlets wie `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.

5.10.1 Punkt-Operator

Wenn man wirklich nur den Inhalt einer bestimmten Eigenschaft eines Objekts haben möchte, so verwendet man den in objektorientierten Sprachen üblichen Punkt-Operator, 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:

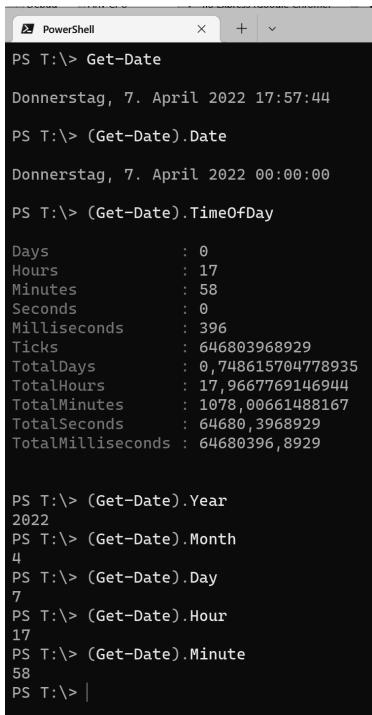
```
(Get-Process)[0].WorkingSet64 / 1MB
```

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

PS C:\Windows\System32> <get-process>[0].Processname
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 5.11 Ausgabe zu den obigen Beispielen

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



```

PS T:\> Get-Date
Donnerstag, 7. April 2022 17:57:44

PS T:\> (Get-Date).Date
Donnerstag, 7. April 2022 00:00:00

PS T:\> (Get-Date).TimeOfDay

Days          : 0
Hours         : 17
Minutes       : 58
Seconds       : 0
Milliseconds  : 396
Ticks         : 646803968929
TotalDays     : 0,748615704778935
TotalHours    : 17,9667769146944
TotalMinutes  : 1078,00661488167
TotalSeconds  : 64680,3968929
TotalMilliseconds : 64680396,8929

PS T:\> (Get-Date).Year
2022
PS T:\> (Get-Date).Month
4
PS T:\> (Get-Date).Day
7
PS T:\> (Get-Date).Hour
17
PS T:\> (Get-Date).Minute
58
PS T:\> |

```

Bild 5.12

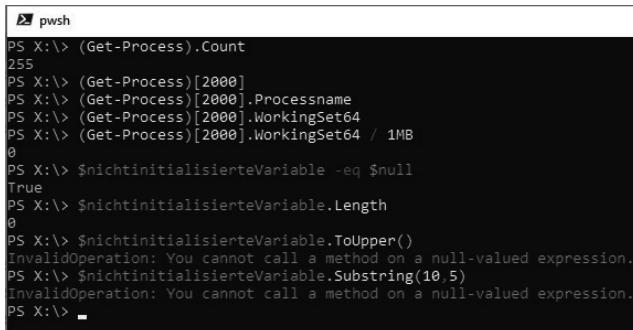
Zugriff auf einzelne Werte aus dem aktuellen Datum/der aktuellen Zeit

5.10.2 Null-Werte

Zu beachten ist, dass PowerShell-Objekte, wie in objektorientierten Sprachen üblich, den Null-Wert (in PowerShell: `$null`) annehmen können mit der Interpretation, dass ein Objekt nicht vorhanden ist. Anders als in den meisten objektorientierten Sprachen führt die Anwendung des Punkt-Operators auf Null-Werte aber nicht zwangsläufig zu einem Laufzeitfehler. Die PowerShell ist sehr tolerant:

- Wenn man einen Null-Wert ausgibt, bekommt man keine Ausgabe.
- Wenn man in der Pipeline auf einen Null-Wert den Punkt-Operator anwendet, wird der Laufzeitfehler unterdrückt und man erhält keine Ausgabe.

Die PowerShell ist aber nicht in allen Fällen gegenüber der Anwendung des Punkt-Operators auf Variablen mit Wert `$null` tolerant (siehe folgende Abbildung).



```

PS X:\> (Get-Process).Count
255
PS X:\> (Get-Process)[2000]
PS X:\> (Get-Process)[2000].Processname
PS X:\> (Get-Process)[2000].WorkingSet64
PS X:\> (Get-Process)[2000].WorkingSet64 / 1MB
0
PS X:\> $nichtinitialisierteVariable -eq $null
True
PS X:\> $nichtinitialisierteVariable.Length
0
PS X:\> $nichtinitialisierteVariable.ToUpper()
InvalidOperationException: You cannot call a method on a null-valued expression.
PS X:\> $nichtinitialisierteVariable.Substring(10,5)
InvalidOperationException: You cannot call a method on a null-valued expression.
PS X:\> -

```

Bild 5.13

Null-Werte in der PowerShell

5.10.3 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 `$_.Name` 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 muss man Foreach-Object anwenden 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!

■ 5.11 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

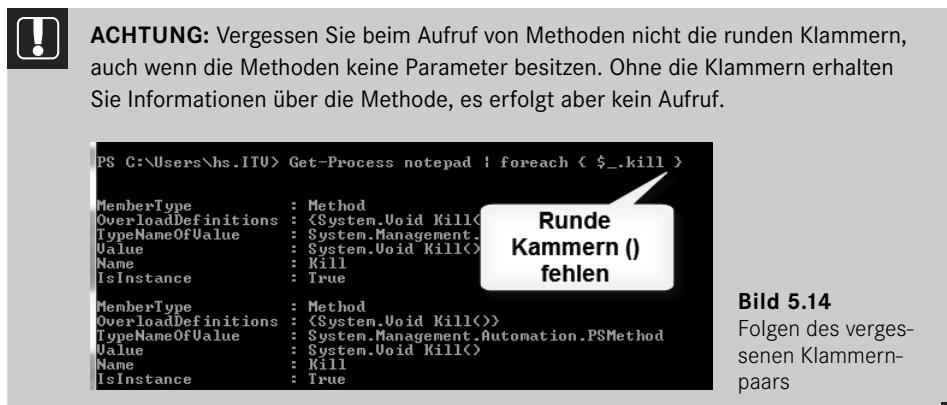
```
(Get-Process iexplore).Kill()
```

oder

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

schreiben.

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



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 "iexplorer" | 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 folgende Abbildung.

```
PS T:\> Get-Date
Donnerstag, 7. April 2022 17:59:50
PS T:\> (Get-Date).ToString()
Donnerstag, 7. April 2022
PS T:\> (Get-Date).ToShortDateString()
07.04.2022
PS T:\> (Get-Date).ToLongTimeString()
18:00:01
PS T:\> (Get-Date).ToShortTimeString()
18:00
PS T:\> |
```

Bild 5.15

Ausgaben der Methoden der Klasse `DateTime`

■ 5.12 Analyse des Pipeline-Inhalts

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

- Wie viele Objekte sind in der Pipeline? (Das wurde schon zuvor in diesem Kapitel erörtert.)
- 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 zwar lesen:

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

Bei anderen Commandlets aber heißt es nur wenig hilfreich:

```
OUTPUTS
Object
```

In keinem Fall sind in der PowerShell-Benutzerdokumentation (siehe <https://docs.microsoft.com/en-us/powershell/> und das Commandlet `Get-Help`) die Attribute und die Methoden der resultierenden Objekte genannt. Diese findet man nur in der .NET API-Dokumentation [<https://docs.microsoft.com/de-de/dotnet/api/>].

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

- Count und Length
- ToString()
- GetType()
- Get-PipelineInfo
- Get-Member

5.12.1 Anzahl der Objekte in der Pipeline mit Count und Length

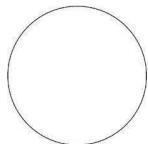
Viele Commandlets legen ganze Mengen von Objekten in die Pipeline (z.B. Get-Process eine Liste der Prozesse und Get-Service eine Liste der Dienste). Bei einer Objektmenge kann man, wie oben bereits gezeigt, mit Where-Object filtern. Das Ergebnis kann ein Objekt, kein Objekt oder eine Menge von Objekten sein.

Es kann aber auch sein, dass ein Commandlet, das normalerweise eine Menge von Objekten liefert, im konkreten Fall (z.B. bei Einsatz eines filternden Parameters) 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.

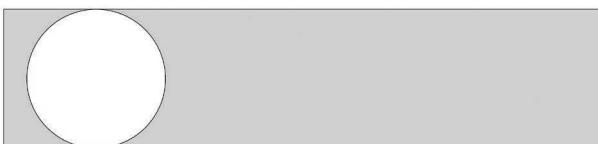
Einige Commandlets legen aber immer nur einzelne Objekte in die Pipeline. Ein Beispiel dafür ist Get-Date, das ein einziges Objekt des Typs System.DateTime in die Pipeline legt. Ruft man z.B. Get-Date ohne Weiteres auf, werden das aktuelle Datum und die aktuelle Zeit ausgegeben.

Zu differenzieren ist, ob die Pipeline ein Objekt direkt enthält oder eine Menge, die aus einem Objekt besteht (siehe Abbildung).

Pipeline mit einem Einzelobjekt



Pipeline mit einer Menge (ein Object[]), die nur ein Objekt enthält



Pipeline mit einer Menge (ein Object[]), die drei Objekte enthält

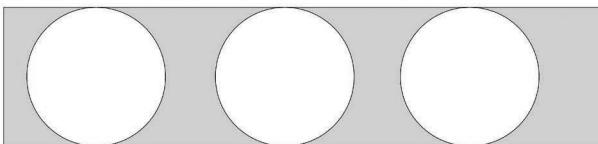


Bild 5.16
Einzelobjekt versus Menge

Bis Version 2.0 der PowerShell 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
```

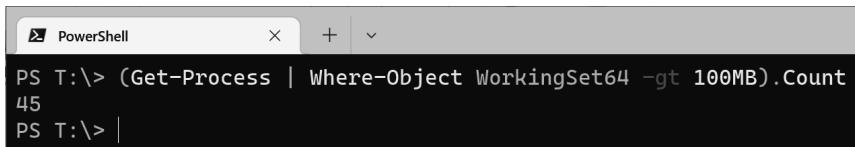
Das führte aber zu keinem Ergebnis:

```
(Get-Process | Where-Object WorkingSet64 -gt 100MB).Count
```

Seit PowerShell-Version 3.0 ist dieser Unterschied (in den meisten Fällen) aufgehoben, man kann auch bei Einzelobjekten 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!

Praxislösung: Wie viele Prozesse gibt es, die mehr als 100 MB Hauptspeicher (RAM) verbrauchen?

```
(Get-Process | Where-Object WorkingSet64 -gt 100MB).Count
```



```
PS T:\> (Get-Process | Where-Object WorkingSet64 -gt 100MB).Count
45
PS T:\> |
```

Bild 5.17 Aufruf von Count für eine Pipeline

Es gibt aber (mindestens) einen Fall, in denen Count auf einem Einzelobjekt nicht funktioniert. Dieser Fall, der nicht dokumentiert, mir aber in der Praxis ausgefallen ist, ist ein einzelnes PSCustomObject in der Pipeline. Es kann sicherlich weitere solcher nicht-dokumentierter Fälle geben. Wenn Sie Fälle kennen, schreiben Sie mir bitte!

Das folgende Beispiel zeigt auch, wie Sie diese Anomalie umgehen: Mit einem vorangestellten Komma macht man aus dem Einzelobjekt (System.Management.Automation.PSCustomObject) eine Menge mit einem Objekt (System.Object[]) mit einem System.Management.Automation.PSCustomObject.

Listing 5.2 [\\PowerShell\\1_Basiswissen\\Pipelining\\Pipelining.ps1]

```
$prozesse = Get-Process | select -First 1
Write-Host "Anzahl Prozesse: " $prozesse.Count # 1

$zahlen = 123
Write-Host "Anzahl Zahlen: " $zahlen.Count # 1

$firma1 = [PSCustomObject]@{
    Firma      = "www.IT-Visions.de"
    Ort        = "Essen"
}

Write-Host "Anzahl Firmen: " $firma1.Count # geht nicht! $null
$firma1.GetType().FullName # System.Management.Automation.PSCustomObject
if ($firma1.Count -eq $null) { Write-Warning "Count ist null!" }
```

```
# Workaround für Anomalie: Das vorangestellte Komma macht aus dem Einzelobjekt eine
Menge mit einem Objekt.
$firmen = , $firmal
$firmen.GetType().FullName # System.Object[]
Write-Host "Anzahl Firmen: " $firmen.Count # 1
```



TIPP: Ob die Pipeline ein Einzelobjekt oder eine Menge enthält, können Sie über den Aufruf von Count oder Length nicht zuverlässig feststellen. Hierzu müssen Sie das der PowerShell zu Grunde liegende .NET fragen, aus welcher Klasse die Pipeline stammt. Dies erfolgt durch den Aufruf .GetType().FullName. Wenn dieser Aufruf System.Object[] liefert, ist der Inhalt ein „Array von Objekten“, also eine Menge. Die geschweiften Klammern bedeuten in .NET ein „Array“ (Menge).

```
# Einzelobjekt
$pipeline = 1
$pipeline.GetType().FullName # System.Int32
# Menge
$pipeline = 1,2
$pipeline.GetType().FullName # System.Object[]
```

Sie lernen dies im Detail noch im Kapitel „Verwendung von .NET-Klassen“.

5.12.2 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 zeigen die folgende Abbildung:

```
PS X:\> (Get-Date).GetType()
IsPublic IsSerial Name                                     BaseType
-----  -----  -----
True      True    DateTime                               System.ValueType

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

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

PS X:\> (Get-Process)[0].GetType().Name
Process
PS X:\> (Get-Process)[0].GetType().FullName
System.Diagnostics.Process
PS X:\> |
```

Bild 5.18 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 jedoch zumeist 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.

5.12.3 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. Manchmal wird der Name des Objekts alleine (z.B. bei den Instanzen der Klasse `System.Diagnostics.Process`, die das Commandlet `Get-Process` liefert), manchmal der Name der Klasse mit dem Objektnamen geliefert (z.B. bei den Instanzen der Klasse `System.ServiceController`, die das Commandlet `Get-Service` liefert).

Listing 5.3 [Basiswissen\Pipelining\ToString.ps1]

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



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 fast keine .NET-Klasse. Bei vielen .NET-Klassen liefert `ToString()` nur den Klassennamen.

Ob `ToString()` eine sinnvolle Ausgabe liefert, hängt von der jeweiligen Klasse ab. Der Autor dieses Buchs und auch Sie als Nutzer haben darauf keinen Einfluss für die Klassen, die Microsoft und andere geschrieben haben. Sie können darauf nur in den Klassen Einfluss nehmen, die Sie selbst schreiben.

```

pwsh
PS X:\> (Get-service a*) | foreach {$_.ToString()}
AarSvc_abb5e
AcronisActiveProtectionService
AcrSch2Svc
AdobeARMservice
afcdpsrv
AIRouter
ALG
AMD External Events Utility
AntiViruskit Client
AppHostSvc
AppIDSvc
AppInfo
AppLgmt
AppReadiness
AppVClient
AppXsvc
aspnet_state
AssignedAccessManagerSvc
AudioEndpointBuilder
Audiosrv
autotimesvc
AVKProxy
AVKWct1
AxInstSV
PS X:\> (Get-Process a*) | foreach {$_.ToString()}
System.Diagnostics.Process (AcroRd32)
System.Diagnostics.Process (AcroRd32)
System.Diagnostics.Process (afcdpsrv)
System.Diagnostics.Process (anti_ransomware_service)
System.Diagnostics.Process (ApplicationFrameHost)
System.Diagnostics.Process (armsvc)
System.Diagnostics.Process (aticlxx)
System.Diagnostics.Process (atiesrxx)
System.Diagnostics.Process (audiogd)
System.Diagnostics.Process (AVKproxy)
System.Diagnostics.Process (AVKWctlx64)
PS X:\> (Get-Host).ToString()
System.Management.Automation.Internal.Host.InternalHost
PS X:\>

```

Bild 5.19

Anwendung von `ToString()` auf Instanzen verschiedener Klassen

5.12.4 Get-PipelineInfo

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

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

```

Windows PowerShell
PS T:\> Get-ChildItem T:\Daten\ | Get-PipelineInfo
Count TypeName          String
-----
1 System.IO.DirectoryInfo Kunden
2 System.IO.DirectoryInfo Webservices
3 System.IO.FileInfo      dienste.csv
4 System.IO.FileInfo      links.txt
5 System.IO.FileInfo      LinksToCheck-Error.txt.lnk
6 System.IO.FileInfo      webserver.txt

PS T:\>

```

Bild 5.20

`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 Abbildung) ist erklärungsbedürftig: Dies ist die Zeichenkettenrepräsentation mit `ToString()`

5.12.5 Get-Member

Das eingebaute Commandlet Get-Member (Alias: gm) ist sehr hilfreich: Es zeigt den .NET-Klassennamen für die Objekte in der Pipeline sowie die Attribute und Methoden dieser Klasse. Für `GetProcess | Get-Member` ist die Ausgabe so lang, dass man dazu zwei Bildschirmabbildungen braucht.



HINWEIS: Wenn sich mehrere verschiedene Objekttypen in der Pipeline befinden, werden die Mitglieder aller Typen ausgegeben, gruppiert durch die Kopfsektion, die mit „`TypeName:`“ beginnt.

```
PS T:\> Get-Process | Get-Member

TypeName: System.Diagnostics.Process

Name           MemberType      Definition
----           --           --
Handles        AliasProperty Handles = HandleCount
Name          AliasProperty Name = ProcessName
NPM           AliasProperty NPM = NonpagedSystemMemorySize64
PM            AliasProperty PM = PagedMemorySize64
SI             AliasProperty SI = SessionId
VM             AliasProperty VM = VirtualMemorySize64
WS             AliasProperty WS = WorkingSet64
Parent         CodeProperty   System.Object Parent{get=GetParentProcess;}
Disposed       Event          System.EventHandler Disposed(System.Object, System.EventArgs)
ErrorDataReceived Event        System.Diagnostics.DataReceivedEventHandler ErrorDataReceived(System.Object,...)
Exited        Event          System.EventHandler Exited(System.Object, System.EventArgs)
OutputDataReceived Event        System.Diagnostics.DataReceivedEventHandler OutputDataReceived(System.Object...
BeginErrorReadLine Method      void BeginErrorReadLine()
BeginOutputReadLine Method     void BeginOutputReadLine()
CancelErrorRead  Method      void CancelErrorRead()
CancelOutputRead Method     void CancelOutputRead()
Close          Method      void Close()
CloseMainWindow Method     void CloseMainWindow()
Dispose        Method      void Dispose(), void IDisposable.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      void Kill(), void Kill(bool entireProcessTree)
Refresh        Method     void Refresh()
Start          Method      bool Start()
ToString       Method      string ToString()
WaitForExit    Method      void WaitForExit(), bool WaitForExit(int milliseconds)
WaitForExitAsync Method     System.Threading.Tasks.Task WaitForExitAsync(System.Threading.CancellationToken...
WaitForInputIdle Method     void WaitForInputIdle(), bool WaitForInputIdle(int milliseconds)
__NounName     NoteProperty string __NounName=Process
BasePriority   Property     int BasePriority {get;}
Container     Property     System.ComponentModel.IContainer Container {get;}
EnableRaisingEvents Property  bool EnableRaisingEvents {get;set;}
ExitCode       Property     int ExitCode {get;}
ExitTime       Property     datetime ExitTime {get;}
Handle         Property     System.IntPtr Handle {get;}
HandleCount    Property     int HandleCount {get;}
HasExited      Property     bool HasExited {get;}
Id             Property     int Id {get;}
MachineName   Property     string MachineName {get;}
MainModule     Property     System.Diagnostics.ProcessModule MainModule {get;}
MainWindowHandle Property  System.IntPtr MainWindowHandle {get;}
MainWindowTitle Property  string MainWindowTitle {get;}
MaxWorkingSet  Property     System.IntPtr MaxWorkingSet {get;set;}
MinWorkingSet  Property     System.IntPtr MinWorkingSet {get;set;}
Modules        Property     System.Diagnostics.ProcessModuleCollection Modules {get;}
NonpagedSystemMemorySize Property  int NonpagedSystemMemorySize {get;}
NonpagedSystemMemorySize64 Property long NonpagedSystemMemorySize64 {get;}
```

Bild 5.21 Teil 1 der Ausgabe von `Get-Process | Get-Member`

```

NonpagedSystemMemorySize64 Property long NonpagedSystemMemorySize64 {get;}
PagedMemorySize Property int PagedMemorySize {get;}
PagedMemorySize64 Property long PagedMemorySize64 {get;}
PagedSystemMemorySize Property int PagedSystemMemorySize {get;}
PagedSystemMemorySize64 Property long PagedSystemMemorySize64 {get;}
PeakPagedMemorySize Property int PeakPagedMemorySize {get;}
PeakPagedMemorySize64 Property long PeakPagedMemorySize64 {get;}
PeakVirtualMemorySize Property int PeakVirtualMemorySize {get;}
PeakVirtualMemorySize64 Property long PeakVirtualMemorySize64 {get;}
PeakWorkingSet Property int PeakWorkingSet {get;}
PeakWorkingSet64 Property long PeakWorkingSet64 {get;}
PriorityBoostEnabled Property bool PriorityBoostEnabled {get;set;}
PriorityClass Property System.Diagnostics.ProcessPriorityClass PriorityClass {get;set;}
PrivateMemorySize Property int PrivateMemorySize {get;}
PrivateMemorySize64 Property long PrivateMemorySize64 {get;}
PrivilegedProcessorTime Property timespan PrivilegedProcessorTime {get;}
ProcessName Property string ProcessName {get;}
ProcessorAffinity Property System.IntPtr ProcessorAffinity {get;set;}
Responding Property bool Responding {get;}
SafeHandle Property Microsoft.Win32.SafeHandles.SafeProcessHandle SafeHandle {get;}
SessionId Property int SessionId {get;}
Site Property System.ComponentModel.ISite Site {get;set;}
StandardError Property System.IO.StreamReader StandardError {get;}
StandardInput Property System.IO.StreamWriter StandardInput {get;}
StandardOutput Property System.IO.StreamReader StandardOutput {get;}
StartInfo Property System.Diagnostics.ProcessStartInfo StartInfo {get;set;}
StartTime Property datetime StartTime {get;}
SynchronizingObject Property System.ComponentModel.ISynchronizeInvoke SynchronizingObject {get;set;}
Threads Property System.Diagnostics.ProcessThreadCollection Threads {get;}
TotalProcessorTime Property timespan TotalProcessorTime {get;}
UserProcessorTime Property timespan UserProcessorTime {get;}
VirtualMemorySize Property int VirtualMemorySize {get;}
VirtualMemorySize64 Property long VirtualMemorySize64 {get;}
WorkingSet Property int WorkingSet {get;}
WorkingSet64 Property long WorkingSet64 {get;}
PSConfiguration PropertySet PSConfiguration {Name, Id, PriorityClass, FileVersion}
PSResources PropertySet PSResources {Name, Id, HandleCount, WorkingSet, NonPagedMemorySize, PagedMem...}
CommandLine ScriptProperty System.Object CommandLine {get=...}
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=$this.MainModule.FileVersionInfo.FileDescript...}
FileVersion ScriptProperty System.Object FileVersion {get=$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.ProductVe...}
PS T:\> |

```

Bild 5.22 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 anzeigen 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.

5.12.6 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.

5.12.7 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 Abbildung). 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.

```

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
PM           AliasProperty PM = PagedMemorySize
UM           AliasProperty UM = VirtualMemorySize
VM           AliasProperty VM = WorkingSet
WS           AliasProperty WS = WorkingSet64
Disposed      Event          System.EventHandler Disposed<System.Object, System.EventArgs>
EventReceived 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<>()
CancelIErrorRead Method    System.Void CancelIErrorRead()
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;}
MaxWindowTitle Property   System.String MaxWindowTitle {get;}
MaxWorkingSet Property   System.IntPtr MaxWorkingSet {get;set;}
MinWorkingSet Property   System.IntPtr MinWorkingSet {get;set;}
Modules      Property     System.Diagnostics.ProcessModuleCollection Modules {get;}

```

Bild 5.23 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"
```

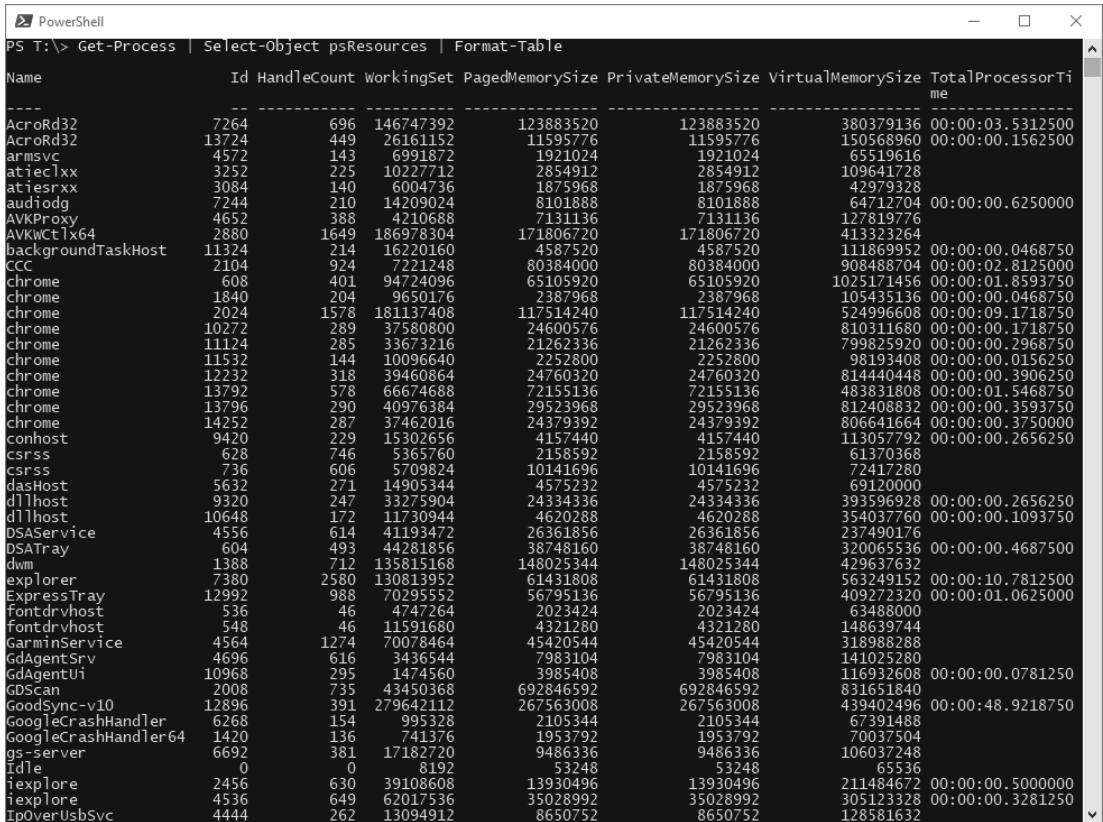
Hier geht nur die o.g. Schreibweise mit Foreach-Object.

5.12.8 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.



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

```
PS T:\> Get-Process | Select-Object psResources | Format-Table
```

The output is a table with columns: Name, Id, HandleCount, WorkingSet, PagedMemorySize, PrivateMemorySize, VirtualMemorySize, and TotalProcessorTime.

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	
atjeclxx	3252	225	10227712	2854912	2854912	109641728	
atiesrxx	3084	140	6004736	1875968	1875968	42979328	
audiogd	7244	210	14209024	8101888	8101888	64712704	00:00:00.6250000
AVKProxy	4652	388	4210688	7131136	7131136	127819776	
AVKwCtIx64	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
cssrs	628	746	5365760	2158592	2158592	61370368	
cssrs	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
DSAService	4556	614	41193472	26361856	26361856	237490176	
DSATray	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	7029552	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	
GdAgentSrv	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	
gs-server	6692	381	17182720	9486336	9486336	106037248	
Idle	0	0	8192	53248	53248	65536	
iexplore	2456	630	39108608	13930496	13930496	211484672	00:00:00.5000000
explorer	4536	649	62017536	35028992	35028992	305123328	00:00:00.3281250
IpOverUsbSvc	4444	262	13094912	8650752	8650752	128581632	

Bild 5.24 Verwendung des Eigenschaftssatzes „psResources“

```
<PropertySet>
  <Name>PSConfiguration</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 5.25

Definition der Eigenschaftssätze für die Klasse
System.Diagnostics.Process in types.ps1xml

5.12.9 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.



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

5.12.10 Skripteigenschaften (ScriptProperty)

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

```
Get-Process | Select-Object name, product | where { $_.product -ne "" -and $_.product -ne $null }
```

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



TIPP: Nicht zu allen Prozessen bekommt man eine Produktinfo. Manchmal liefert die Eigenschaft `$null`, manchmal eine leere Zeichenkette. Die obige Bedingung schließt beides aus.

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

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

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

```
PS T:\> Get-Process | Select-Object name, product | where { $_.product -ne "" -and $_.product -ne $null }

Name                               Product
----                               -----
Acrobat                            Adobe Acrobat DC
Acrobat                            Adobe Acrobat DC
ApplicationFrameHost               Microsoft® Windows® Operating System
Code                               Visual Studio Code
Code                               Visual Studio Code
Code                               Visual Studio Code
Code                               Visual Studio Code
CompPkgSrv                          Microsoft® Windows® Operating System
conhost                            Microsoft® Windows® Operating System
CyberProtectHomeOfficeMonitor      Acronis Cyber Protect Home Office
devenv                             Microsoft® Visual Studio®
dllhost                            Microsoft® Windows® Operating System
dllhost                            Microsoft® Windows® Operating System
eWallet                             eWallet
explorer                           Microsoft® Windows® Operating System
filezilla                           FileZilla
git-credential-manager-core       git-credential-manager-core
HelpPane                           Microsoft® Windows® Operating System
iisexpress                         Internet Information Services
iisexpresstray                     Microsoft® Web Platform Extensions
ITV_CRMTools                       ITV CRM
ITV_CRMTools                       ITV CRM
laclient                            Logitech Analytics Client
LCDClock                           Logitech GamePanel Software
LCDCountdown                      Logitech GamePanel Software
LCDMedia                            Logitech GamePanel Software
LCDMovieViewer                     Logitech GamePanel Software
LCDPictureViewer                  Logitech GamePanel Software
LCDPOP3                            Logitech GamePanel Software
LCDRSS                             Logitech GamePanel Software
LCDWebCam                          Logitech GamePanel Software
LCore                              Logitech Gaming Framework
LockApp                            Microsoft® Windows® Operating System
logitechchg_discord                Logitech G Discord Applet
```

Bild 5.26 Auflistung der berechneten Eigenschaft „Product“

```
615    <ScriptProperty>
616        <Name>Product</Name>
617        <GetScriptBlock>$this.MainModule.FileVersionInfo.ProductName</GetScriptBlock>
618    </ScriptProperty>
```

Bild 5.27 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 das Kapitel „*Dynamische Objekte*“.

5.12.11 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.

5.12.12 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.

5.12.13 Hintergrundwissen: Adapted Type System (ATS) / Extended Type System (ETS)

Als Extended Type System (ETS) bezeichnet Microsoft die Möglichkeit, .NET-Klassen in der PowerShell um Klassenmitglieder zu erweitern, ohne im klassischen Sinne der Objektierung von diesen Klassen zu erben.

Als Adapted Type System (ATS) bezeichnet Microsoft die grundsätzliche Anpassung von .NET-Klassen aus der .NET-Klassenbibliothek auf die Bedürfnisse von PowerShell-Benutzern. 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.

Die Ergänzung von Mitgliedern per ATS 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 objects are "meta" objects (for example: WMI Objects, ADO objects, and XML objects) whose members describe the data they contain. However, in a scripting environment it is the contained data that is most interesting, not the description of the contained data. ETS resolves this issue by introducing the notion of Adapters that adapt the underlying .NET object to have the expected default semantics.
- 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.

Bild 5.28 Quelle: <https://docs.microsoft.com/en-us/powershell/scripting/developer/ets/overview>

Dies heißt im Klartext, dass das PowerShell-Team mit der Arbeit des Entwicklungsteams der .NET-Klassenbibliothek nicht ganz zufrieden ist.

Das ATS verpackt grundsätzlich jedes Objekt, das von einem Commandlet in die Pipeline gelegt wird, in ein PowerShell-Objekt des Typs PSObject. Die Implementierung 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, ManagementClass, DirectoryEntry und DataRow implementiert wurden,
- die Deklarationen in der *types.ps1xml*-Datei,
- in den Commandlets hinzugefügte Mitglieder,
- mit dem Commandlet Add-Member hinzugefügte Mitglieder.

Die folgende Tabelle zeigt die .NET-Klassen, die im Standard per ATS verändert werden:

Tabelle 5.2 .NET-Klassen mit ATS

PowerShell-Wrapper	.NET Framework-Klasse
WMI Class	System.Management.ManagementClass
WMI Object	System.Management.ManagementObject
ADSI Object	System.DirectoryServices.DirectoryEntry
ADO.NET DataRowView	System.Data.DataRowView
ADO.NET DataRow	System.Data.DataRow
XML	System.Xml.XmlNode
PSObject	System.Management.Automation.PSObject
PSMemberSet	System.Management.Automation.PSMemberSet
COM Object	System.__ComObject
.NET Object	System.Object

■ 5.13 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
```

Mit `Select-Object` kann man auch eine Teilmenge aus der Mitte auswählen, indem man am Beginn einige Elemente mit `-Skip` überspringt:

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

5.13.1 Operatoren

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

Dazu zwei Beispiele:

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

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

Windows PowerShell		
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	DusmSvc	Data Usage
Stopped	embeddedmode	Embedded Mode

Bild 5.29 Ausgabe zu obigem Beispiel

- 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)	VM(M)	CPU(s)	Id ProcessName
0	0	0	24	0	0	Idle
PS H:\> ..						

Bild 5.30
Ausgabe zu obigem Beispiel

Tabelle 5.3 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
-is		Typvergleich, z. B. (Get-Date) -is [DateTime]
-in -contains		Ist enthalten in Menge
-notin -notcontains		Ist nicht enthalten in Menge

Tabelle 5.4 Logische Operatoren in der PowerShell-Sprache

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

5.13.2 Vereinfachte Schreibweise von Bedingungen seit PowerShell 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 befreindlich.

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.

5.13.3 Where()-Methode seit PowerShell 4.0

In PowerShell hat Microsoft eine Optionen für das Filtern von Pipelines eingebaut, die sich vor allem an fortgeschrittene PowerShell-Nutzer richtet bzw. an Softwareentwickler, die die PowerShell nutzen. Alternativ zum Commandlet `Where-Object` kann man nun auch mit einer `Where()`-Methode filtern. Anstelle von

```
Get-Service a* | where status -eq "stopped"
```

oder

```
Get-Service a* | Where-Object { $_.status -eq "stopped" }
```

Ist nun auch diese Syntax möglich:

```
(Get-Service a*).Where({ $_.status -eq "stopped"})
```

Dabei ist die Eingabemenge, die auch eine Pipeline mit mehreren Commandlets sein kann, zu klammern.

Man kann auch mehrere Bedingungen verketten:

```
(Get-Service).Where({ ($_.name.startswith("a") -or $_.name.startswith("A")) -and $_.status -eq "stopped" })
```

Soweit bietet die Methode Where() nichts, was das Commandlet Where-Object nicht auch könnte - nur in anderer Syntax.

Interessant sind die weiteren Optionen. Man kann bei der Where()-Methode einen weiteren Parameter angeben: Default, First, Last, SkipUntil, Until, Split. Dieser Parameter muss als Zeichenkette übergeben werden.

Beispiele:

```
# Alle, bis Bedingung erfüllt
(1..10).Where({ $_ -eq 5},'Until')
# Nur das erste Objekt, das Bedingung erfüllt, also 6
(1..10).Where({ $_ -gt 5},'First')
# Nur das letzte Objekt, das Bedingung erfüllt, also 10
(1..10).Where({ $_ -gt 5},'Last')
```

Sehr spannend ist die Möglichkeit, eine Menge mit Where() im Modus 'Split' in zwei Teilmengen zu teilen und als Ergebnis des Befehls direkt zwei Ausgabevariablen zu erhalten:

```
# Teile eine Menge von Zahlen in zwei Teile
$kleiner,$groesser = (Get-Random -max 49 -Count 7).Where({ $_ -lt 30},'Split')
"# Zahlen < 5"
$kleiner
"# Zahlen >= 5"
$groesser
```



HINWEIS: Dieses Beispiel setzt PowerShell 7.0 oder höher voraus, da der Parameter -count bei Get-Random erst in PowerShell 7 eingeführt wurde.

```
# Zahlen < 5
27
11
17
29
15
# Zahlen >= 5
40
36
```

Bild 5.31

Gespaltene Ausgabe der Zufallszahlen

Auch komplexe Objekte kann man so mit Where() im Modus 'Split' in Teilmengen aufteilen:

```
# Teile die Dienste in zwei Teilmengen
$Running,$Stopped = (Get-Service a*).Where({$.Status -eq 'Running'},'Split')
$Running
$Stopped
```

■ 5.14 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`).

Count	TypeName	
1	System.Diagnostics.Process	System.Diagnostics.Process (Idle)
2	System.Diagnostics.Process	System.Diagnostics.Process (iisexpress)
3	System.Diagnostics.Process	System.Diagnostics.Process (iisexpressstray)
4	System.ServiceProcess.ServiceController	icssvc
5	System.ServiceProcess.ServiceController	IKEEXT
6	System.ServiceProcess.ServiceController	InstallService
7	System.ServiceProcess.ServiceController	iphlpsvc
8	System.ServiceProcess.ServiceController	IpxlatCfgSvc

Bild 5.32 Anwendung von `Get-PipelineInfo` auf eine heterogene Pipeline

■ 5.15 „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 Abbildung). 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.

```

PowerShell
PS T:\> Get-Process | Select-Object ProcessName, get_WorkingSet64, 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_WorkingSet64 NoteProperty object get_WorkingSet64=null
ProcessName NoteProperty string ProcessName=aakore
WS         NoteProperty long WS=37052416

```

Bild 5.33 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.

■ 5.16 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.

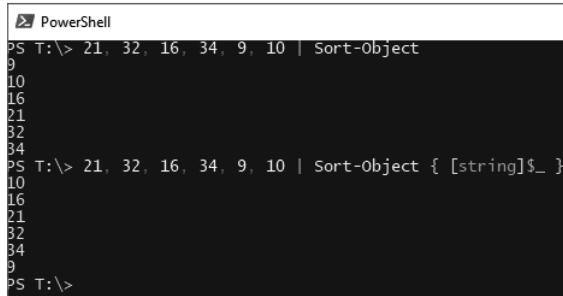
```
Get-Service | Sort-Object Status, DisplayName
```

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]$_ }
```



```
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 5.34

Numerische versus alphabetische Sortierung von sechs Zahlen

■ 5.17 Duplikate entfernen

Sowohl `Select-Object -Unique` als auch `Get-Unique` entfernen Duplikate aus einer Liste. Beispiel

```
1,5,7,8,5,7 | Select-Object -Unique
```

liefert als Ergebnis eine Pipeline mit vier Zahlen: 1,5,7 und 8.



ACHTUNG: Bei Get-Unique muss die Liste 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
```

Beide Commandlets arbeiten nicht nur auf elementaren Datentypen wie Zahlen und Zeichenketten, sondern auch auf komplexen Objekten, z. B.

```
(Get-process | Select-Object -Unique).Count
(Get-process | sort-object | get-unique).Count
```

```

PS X:\> 1,5,7,8,5,7 | Select-Object -Unique
1
5
7
8
PS X:\> (Get-process ).Count
267
PS X:\> (Get-process | sort-object | get-unique).Count
101
PS X:\> 1,5,7,8,5,7 | Get-Unique
1
5
7
8
5
7
PS X:\> 1,5,7,8,5,7 | Sort-Object | Get-Unique
1
5
7
8
PS X:\> (Get-process | get-unique).Count
101
PS X:\> (Get-process | sort-object | get-unique).Count
101
PS X:\> .

```

Bild 5.35

Einsatz von Get-Unique und Select-Object -unique

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

■ 5.18 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 Abbildung): 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
```

```
PS T:> Get-Service | Group-Object status
Count Name                               Group
---- --                                --
124 Running                            {AdobeARMservice, AMD External Events Utility, AntiVirusKit Client, AppHostSv...
143 Stopped                            {AIRouter, 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 .msc                             {0409, 1029, 1033, 1036...}
120 .NLS                            {C_037.NLS, C_10000.NLS, C_10001.NLS, C_10002.NLS...}
42 .msc                             {adsiedit.msc, azman.msc, certlm.msc, certmgr.msc...}
30 .dat                             {amde3la.dat, amdicdxx.dat, atijicdxz.dat, ativce02.dat...}
18 .cpl                             {appwiz.cpl, bthprops.cpl, desk.cpl, Firewall.cpl...}
17 .png                             {@AudioToasterIcon.png, @BackgroundAccessToasterIcon.png, @bitlockertoastimage.png, @edp...
15 .t16                             {activets.t16, amcompat.t16, mqoa.t16, mqoa10.t16...}
15 .ax                             {bdaplginn.ax, g711codec.ax, kspproxy.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                             {cer0.rs, cob-au.rs, csrr.rs, djctd.rs...}
8 .uce                             {bopomofo.uce, gb2312.uce, ideograff.uce, kanji_1.uce...}
7 .bin                             {AverageRoom.bin, DefaultHrtfs.bin, edgehtmpluginpolicy.bin, LargeRoom.bin...}
6 .scr                             {Bubbles.scr, Mystify.scr, PhotoScreensaver.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                          {AppStreamingUX.exe.config, ClusterUpdateUI.exe.config, DfsMgmt.dll.config, dsac.ex...
4 .tsp                             {hidphone.tsp, kmddsp.tsp, remotesp.tsp, unimdm.tsp}
```

Bild 5.36 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                          <\oscd_tmp.ini, bsvc.exe, bootstat.dat, DtcInstall.log...
   2 True                           <explorer.exe, WindowsUpdate.log>
```

Bild 5.37 Ergebnis des obigen Befehls (Zahlen können in Abhängigkeit vom Betriebssystem abweichen)

Praxislösung 1

Es sollen in einer Menge von Zeichenketten (hier: Feldnamen für Work Items in Azure DevOps) Duplikate ermittelt werden. Der eingebettete Here-String wird zunächst mit dem Split-Operator zeilenweise in eine Menge von Zeichenketten aufgespalten. Danach wird diese Menge mit Group-Objekt gruppiert. Im Ergebnis findet man die doppelten Zeichenketten, indem man prüft, bei welchen Elementen die Eigenschaft count größer als eins ist.

Listing 5.4 [Finde doppelte Zeichenketten.ps1]

```
# Finde doppelte Zeichenketten
# Eingabemenge: Zeichenketten (eingebettet als "Here-String" oder aus einer Datei)
# Ausgabe: Liste der doppelt vorkommenden Zeichenketten

$eingabe = @"
Microsoft.VSTS.Build.FoundIn
Microsoft.VSTS.Build.IntegrationBuild
Microsoft.VSTS.CMMI.ActualAttendee1
Microsoft.VSTS.CMMI.ActualAttendee2
Microsoft.VSTS.CMMI.ActualAttendee3
Microsoft.VSTS.CMMI.ActualAttendee4
Microsoft.VSTS.CMMI.ActualAttendee5
Microsoft.VSTS.CMMI.ActualAttendee6
Microsoft.VSTS.CMMI.ActualAttendee7
Microsoft.VSTS.CMMI.ActualAttendee8
Microsoft.VSTS.CMMI.Analysis
Microsoft.VSTS.CMMI.Blocked
Microsoft.VSTS.CMMI.CalledBy
Microsoft.VSTS.CMMI.CalledDate
Microsoft.VSTS.CMMI.Comments
Microsoft.VSTS.CMMI.Committed
Microsoft.VSTS.CMMI.ContingencyPlan
Microsoft.VSTS.CMMI.CorrectiveActionActualResolution
Microsoft.VSTS.CMMI.CorrectiveActionPlan
Microsoft.VSTS.CMMI.Escalate
Microsoft.VSTS.CMMI.FoundInEnvironment
Microsoft.VSTS.CMMI.HowFound
Microsoft.VSTS.CMMI.ImpactAssessmentHtml
```

```
Microsoft.VSTS.CMMI.ImpactOnArchitecture
Microsoft.VSTS.CMMI.ImpactOnDevelopment
Microsoft.VSTS.CMMI.ImpactOnTechnicalPublications
Microsoft.VSTS.CMMI.ImpactOnTest
Microsoft.VSTS.CMMI.ImpactOnUserExperience
Microsoft.VSTS.CMMI.Justification
Microsoft.VSTS.CMMI.MeetingType
Microsoft.VSTS.CMMI.Minutes
Microsoft.VSTS.CMMI.MitigationPlan
Microsoft.VSTS.CMMI.MitigationTriggers
Microsoft.VSTS.CMMI.OptionalAttendee1
Microsoft.VSTS.CMMI.OptionalAttendee2
Microsoft.VSTS.CMMI.OptionalAttendee3
Microsoft.VSTS.CMMI.OptionalAttendee4
Microsoft.VSTS.CMMI.OptionalAttendee5
Microsoft.VSTS.CMMI.OptionalAttendee6
Microsoft.VSTS.CMMI.OptionalAttendee7
Microsoft.VSTS.CMMI.OptionalAttendee8
Microsoft.VSTS.CMMI.Probability
Microsoft.VSTS.CMMI.ProposedFix
Microsoft.VSTS.CMMI.Purpose
Microsoft.VSTS.CMMI.RequiredAttendee1
Microsoft.VSTS.CMMI.RequiredAttendee2
Microsoft.VSTS.CMMI.RequiredAttendee3
Microsoft.VSTS.CMMI.RequiredAttendee4
Microsoft.VSTS.CMMI.RequiredAttendee5
Microsoft.VSTS.CMMI.RequiredAttendee6
Microsoft.VSTS.CMMI.RequiredAttendee7
Microsoft.VSTS.CMMI.RequiredAttendee8
Microsoft.VSTS.CMMI.RequirementType
Microsoft.VSTS.CMMI.RequiresReview
Microsoft.VSTS.CMMI.RequiresTest
Microsoft.VSTS.CMMI.RootCause
Microsoft.VSTS.CMMI.SubjectMatterExpert1
Microsoft.VSTS.CMMI.SubjectMatterExpert2
Microsoft.VSTS.CMMI.SubjectMatterExpert3
Microsoft.VSTS.CMMI.Symptom
Microsoft.VSTS.CMMI.TargetResolveDate
Microsoft.VSTS.CMMI.TaskType
Microsoft.VSTS.CMMI.UserAcceptanceTest
Microsoft.VSTS.CodeReview.AcceptedBy
Microsoft.VSTS.CodeReview.AcceptedDate
Microsoft.VSTS.CodeReview.ClosedStatus
Microsoft.VSTS.CodeReview.ClosedStatusCode
Microsoft.VSTS.CodeReview.ClosedStatusCode
Microsoft.VSTS.CodeReview.ClosingComment
Microsoft.VSTS.CodeReview.Context
Microsoft.VSTS.CodeReview.ContextCode
Microsoft.VSTS.CodeReview.ContextOwner
Microsoft.VSTS.CodeReview.ContextType
Microsoft.VSTS.Common.AcceptanceCriteria
Microsoft.VSTS.Common.ActivatedBy
Microsoft.VSTS.Common.ActivatedDate
Microsoft.VSTS.Common.Activity
Microsoft.VSTS.Common.BusinessValue
Microsoft.VSTS.Common.ClosedBy
Microsoft.VSTS.Common.ClosedDate
Microsoft.VSTS.Common.Discipline
Microsoft.VSTS.Common.Issue
```

```
Microsoft.VSTS.Common.Priority
Microsoft.VSTS.Common.Rating
Microsoft.VSTS.Common.Resolution
Microsoft.VSTS.Common.ResolvedBy
Microsoft.VSTS.Common.ResolvedDate
Microsoft.VSTS.Common.ResolvedReason
Microsoft.VSTS.Common.ReviewedBy
Microsoft.VSTS.Common.Risk
Microsoft.VSTS.Common.Severity
Microsoft.VSTS.Common.StackRank
Microsoft.VSTS.Common.StateChangeDate
Microsoft.VSTS.Common.StateCode
Microsoft.VSTS.Common.TimeCriticality
Microsoft.VSTS.Common.Triage
Microsoft.VSTS.Common.ValueArea
Microsoft.VSTS.Feedback.ApplicationLaunchInstructions
Microsoft.VSTS.Feedback.ApplicationStartInformation
Microsoft.VSTS.Feedback.ApplicationType
Microsoft.VSTS.Scheduling.CompletedWork
Microsoft.VSTS.Scheduling.DueDate
Microsoft.VSTS.Scheduling.Effort
Microsoft.VSTS.Scheduling.FinishDate
Microsoft.VSTS.Scheduling.OriginalEstimate
Microsoft.VSTS.Scheduling.RemainingWork
Microsoft.VSTS.Scheduling.Size
Microsoft.VSTS.Scheduling.StartDate
Microsoft.VSTS.Scheduling.StoryPoints
Microsoft.VSTS.Scheduling.TargetDate
Microsoft.VSTS.TCM.AutomatedTestId
Microsoft.VSTS.TCM.AutomatedTestName
Microsoft.VSTS.TCM.AutomatedTestStorage
Microsoft.VSTS.TCM.AutomatedTestType
Microsoft.VSTS.TCM.AutomationStatus
Microsoft.VSTS.TCM.LocalDataSource
Microsoft.VSTS.TCM.Parameters
Microsoft.VSTS.TCM.QueryText
Microsoft.VSTS.TCM.ReproSteps
Microsoft.VSTS.TCM.Steps
Microsoft.VSTS.TCM.SystemInfo
Microsoft.VSTS.TCM.TestSuiteAudit
Microsoft.VSTS.TCM.TestSuiteType
Microsoft.VSTS.TCM.TestSuiteTypeId
System.AreaId
System.AreaPath
System.AssignedTo
System.AttachedFileCount
System.AuthorizedAs
System.AuthorizedDate
System.BoardColumn
System.BoardColumnDone
System.BoardLane
System.ChangedBy
System.ChangedDate
System.CommentCount
System.CreatedBy
System.CreatedDate
System.Description
System.ExternalLinkCount
System.History
```

```

System.HyperLinkCount
System.Id
System.IterationId
System.IterationPath
System.NodeName
System.Reason
System.RelatedLinkCount
System.RemoteLinkCount
System.Rev
System.RevisedDate
System.State
System.Tags
System.Tags
System.TeamProject
System.Title
System.Watermark
System.WorkItemType
"@"

# Alternativ: Einlesen einer Datei
# $eingabe = get-content "eingabedatei.txt"

# Der eingebettete Here-String wird zunächst mit dem Split-Operator zeilenweise in
# eine Menge von Zeichenketten aufgespalten.
$gespaltet = $eingabe -split "`n" |Sort-Object
# Danach wird diese Menge mit Group-Objekt gruppiert.
$gruppiert = $gespaltet | Group-Object

$anz = ($gespaltet).Count
$anzGruppiert = ($gruppiert).Count
$Duplikate = $gruppiert | where count -gt 1

if ($Duplikate.Count -eq 0)
{
    Write-Host "$Anz Elemente. Keine Duplikate!" -ForegroundColor Green
}
else
{
    Write-Host "$($Duplikate.Count) Zeichenketten kommen mehrfach vor /"
    $anzGruppiert verschiedenen Zeichenketten in $anz Zeilen:" -ForegroundColor red
    $Duplikate | Ft Name, Count
}

$Duplikate | Ft Name, Count
}

```

Praxislösung 2

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

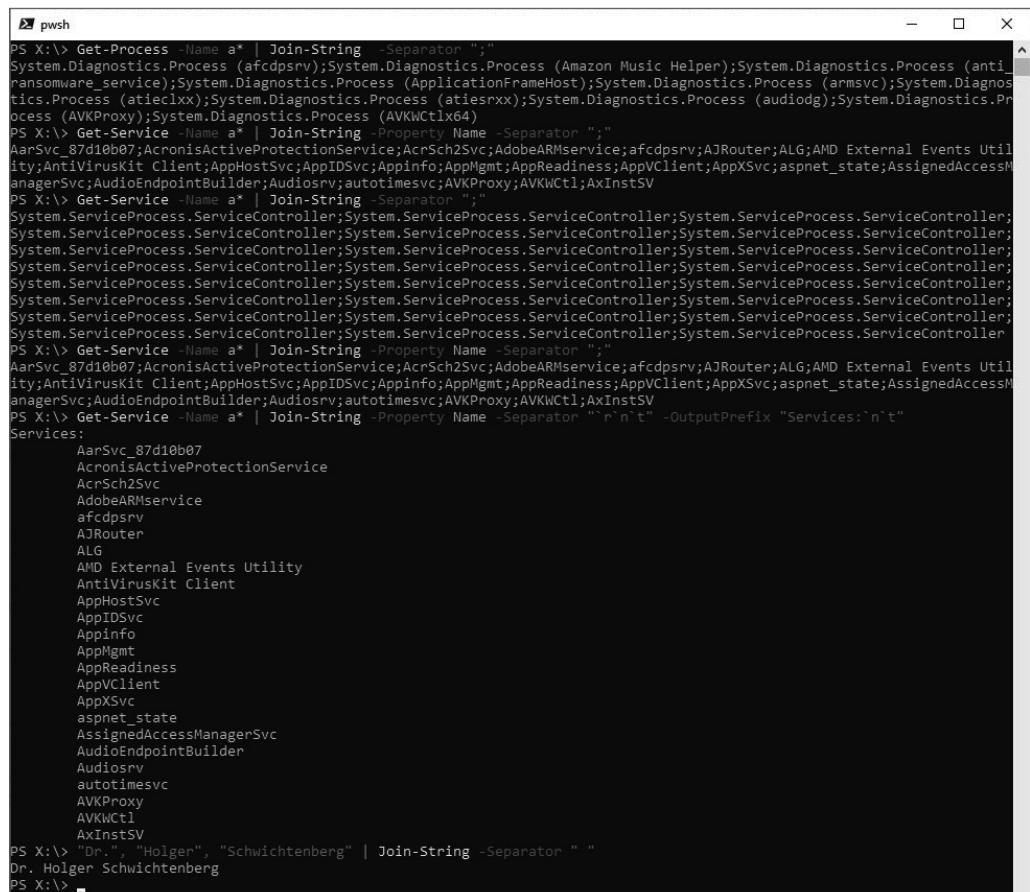
Beispiel: Ermittle aus dem Verzeichnis `System32` alle Dateien, die mit dem Buchstaben „b“ beginnen. Beschränke die Menge auf diejenigen Dateien, die größer als 40 000 Byte sind, und gruppiere die Ergebnismenge nach Dateierweiterungen. Sortiere die Gruppen nach der Anzahl der Einträge absteigend und 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
```

■ 5.19 Objekte verbinden mit Join-String

Das in PowerShell 6 neu eingeführte Commandlet **Join-String** verbindet eine zu bennende Eigenschaft der Objekte in der Pipeline zu einer einzigen Zeichenkette mit einem beliebigen Trennzeichen.

Wenn der Parameter **-Property** nicht angegeben wird, ruft **Join-String** auf den Objekten in der Pipeline die Methode **ToString()** auf. Dies ergibt manchmal einen sinnvollen Inhalt, bei vielen Objekten wird aber nur der Klassenname geliefert. Leer lassen muss man den Parameter **-Property**, wenn der Inhalt der Pipeline primitive Datentypen (Zahlen, Zeichenketten, Datumsangaben etc.) sind.



```
PS X:\> Get-Process -Name a* | Join-String -Separator ":"  
System.Diagnostics.Process (afcdpsrv);System.Diagnostics.Process (Amazon Music Helper);System.Diagnostics.Process (anti_ransomware_service);System.Diagnostics.Process (ApplicationFrameHost);System.Diagnostics.Process (armsvc);System.Diagnostics.Process (aticlxx);System.Diagnostics.Process (atiesrx);System.Diagnostics.Process (audiodg);System.Diagnostics.Process (AVKProxy);System.Diagnostics.Process (AVKWCtrlx64)  
PS X:\> Get-Service -Name a* | Join-String -Property Name -Separator ";"  
AarSvc_87d10b07;AcronisActiveProtectionService;AcrSch2Svc;AdobeARMservice;afcdpsrv;AJRouter;ALG;AMD External Events Utility;AntiViruskit Client;AppHostSvc;AppIDSvc;AppInfo;AppMgmt;AppReadiness;AppVClient;AppXSvc;aspnet_state;AssignedAccessManagerSvc;AudioEndpointBuilder;Audiosrv;autotimesvc;AVKProxy;AVKWCtrl;AxInstSV  
PS X:\> Get-Service -Name a* | Join-String -Separator ";"  
System.ServiceProcess.ServiceController;System.ServiceProcess.ServiceController;System.ServiceProcess.ServiceController;  
System.ServiceProcess.ServiceController;System.ServiceProcess.ServiceController;System.ServiceProcess.ServiceController;  
System.ServiceProcess.ServiceController;System.ServiceProcess.ServiceController;System.ServiceProcess.ServiceController;  
System.ServiceProcess.ServiceController;System.ServiceProcess.ServiceController;System.ServiceProcess.ServiceController;  
System.ServiceProcess.ServiceController;System.ServiceProcess.ServiceController;System.ServiceProcess.ServiceController;  
System.ServiceProcess.ServiceController;System.ServiceProcess.ServiceController;System.ServiceProcess.ServiceController;  
System.ServiceProcess.ServiceController;System.ServiceProcess.ServiceController;System.ServiceProcess.ServiceController;  
System.ServiceProcess.ServiceController;System.ServiceProcess.ServiceController;System.ServiceProcess.ServiceController;  
PS X:\> Get-Service -Name a* | Join-String -Property Name -Separator ";"  
AarSvc_87d10b07;AcronisActiveProtectionService;AcrSch2Svc;AdobeARMservice;afcdpsrv;AJRouter;ALG;AMD External Events Utility;AntiViruskit Client;AppHostSvc;AppIDSvc;AppInfo;AppMgmt;AppReadiness;AppVClient;AppXSvc;aspnet_state;AssignedAccessManagerSvc;AudioEndpointBuilder;Audiosrv;autotimesvc;AVKProxy;AVKWCtrl;AxInstSV  
PS X:\> Get-Service -Name a* | Join-String -Property Name -Separator "`r`n`t" -OutputPrefix "Services:`n`t"  
Services:  
    AarSvc_87d10b07  
    AcronisActiveProtectionService  
    AcrSch2Svc  
    AdobeARMservice  
    afcdpsrv  
    AJRouter  
    ALG  
    AMD External Events Utility  
    AntiViruskit Client  
    AppHostSvc  
    AppIDSvc  
    AppInfo  
    AppMgmt  
    AppReadiness  
    AppVClient  
    AppXSvc  
    aspnet_state  
    AssignedAccessManagerSvc  
    AudioEndpointBuilder  
    Audiosrv  
    autotimesvc  
    AVKProxy  
    AVKWCtrl  
    AxInstSV  
PS X:\> "Dr.", "Holger", "Schwichtenberg" | Join-String -Separator " "  
Dr. Holger Schwichtenberg  
PS X:\>
```

Bild 5.38 Einsatzbeispiele für Join-String

■ 5.20 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      : 2599
Average    : 465545.188030888
Sum        : 1205684337
Maximum    : 26575296
Minimum    : 35
Property   : length

PS C:\Windows\System32\WindowsPowerShell\v1.0>
```

Bild 5.39 Beispiel für den Einsatz von Measure-Object

■ 5.21 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 zwischenzuspeichern. 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" angegeben.
In Zeile:1 Zeichen:6
+ $x | get-member
+
+ CategoryInfo          : CloseError: (:) [Get-Member], InvalidOperationException
+ FullyQualifiedErrorId : NoObjectInGetMember,Microsoft.PowerShell.Commands.GetMemberCommand
PS T:\> -
```

Bild 5.40 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.

■ 5.22 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 x:\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 x:\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 weitergereicht 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.$_" }
```

■ 5.23 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
```

```
pwsh
PS X:\> $vorher = Get-Process
PS X:\> notepad
PS X:\> notepad
PS X:\> mmc
PS X:\> $nachher = Get-Process
PS X:\> Compare-Object $vorher $nachher

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

PS X:\> -
```

Bild 5.41
Vergleich von zwei Pipelines

■ 5.24 Weitere Praxislösungen

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.

```
Get-Process | Measure-Object workingset-sum
```

- Gruppiere die Einträge im System-Ereignisprotokoll nach Benutzernamen.

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

- Zeige die neuesten zehn Einträge im System-Ereignisprotokoll.

```
Get-EventLog -logname system | Sort-Object timegenerated -desc | Select-Object
-first 10
```

- Oder kürzer und schneller (dieses Commandlet besitzt eine eingebaute Filterfunktion):

```
Get-EventLog System -newest 10
```

- 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 gruppiere 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 gruppiere 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 {$_ .Length -gt 40000}
| Group-Object Extension | Select-Object -first 1 | Select-Object group | foreach
{$_ .group} | Select-Object name,length | Format-Table -autosize
```

Stichwortverzeichnis

Symbol

?? 176
?. 176, 178
& 80
&& 231
% 108
> 268
>> 268
|| 231
\$_ 99f., 108, 114, 181, 502
\$? 231, 233
\$ConfirmPreference 1285
\$ErrorView 229
\$null 113, 128, 156, 218,
 807
\$PSItem 108
\$PSSStyle 318, 394
\$psUnsupportedConsole-
 Applications 337
\$psversiontable 395
\$PSVersionTable 27
-and 134
-as 178
-band 213, 442
-Bit 21
-bnot 213, 442
-bor 213, 442
-bxor 213
.cat 718
-cmatch 197
-cnotmatch 197
.dll 61, 166, 433, 705,
 1349f.
.exe 166, 350
-expression 688
-force 66
-imatch 197

-inotmatch 197
-ItemsSource 1257
-Join 196
-match 197
-notmatch 197
-or 134
-Parameter 87
-Split 195f.
-Verbose 66, 610
.NET 3, 17, 46, 100, 181,
 204, 424, 511, 1244, 1298,
 1326, 1341, 1373
- Bibliothek 417
- Klasse 417, 1379
- Runtime Host 15
.NET 6.0 45, 47, 1373
.NET 7.0 45, 1373
.NET API Portability Analyzer
 1128
.NET CLI 51
.NET Core 17, 378, 1154,
 1182, 1373, 1377
.NET Core 3.1 381
.NET Core SDK 51
.NET Data Provider 795f.
.NET Framework 4, 15, 281,
 552, 632, 896, 1137, 1373,
 1376, 1380
- 4.0 20
.NET Standard 420, 1378
.nupkg 437
.pfx 534
.pkg 54
.ps1 28, 76, 152, 1350
.psd1 598, 639, 1272, 1350,
 1355
.psm1 1272, 1345ff., 1350,
 1355
.psproj 348
-wait 610
.yml 1183
32-Bit 21, 332, 801, 858
64-Bit 801, 858
[Type] 429

A

Ablaufverfolgung 3, 517f.
About 168
Absent 616
AbsoluteTimerInstruction
 466
abstract 1370
Accelerator 172
Accent Grave
 – Gravis 68
AccessControl 975, 980
Access Control Entry 976
Access Control List 976,
 988
Access Control Type 976
Access Mask 976
AccessMask 732
AccountDisabled 1023
AceFlags 976
ACL 979, 993
ACR 1154
Active Directory 3, 273, 452,
 606, 653, 668, 1012, 1019,
 1023, 1038, 1068, 1070,
 1273
- PowerShell 1038
- Struktur 1070
- Suche 1028
Active Directory Application
 Mode 1070

- Active Directory Domain Services 1067
- Active Directory Service Interface *siehe* ADSI 1082
- ActiveScriptEventConsumer 467
- Active-Scripting 158
- ActiveX Data Objects 795, 806, 1009
- ADAccount 1044
- Adapted Type System *siehe* ATS 485
- ADComputer 1044
- Add() 432
- Add-ADGroupMember 1048, 1066
- AddCommand() 1363
- Add-Computer 865
- Add-Content 751, 785
- Add-DirectoryEntry 668
- Add-DistributionGroup-Member 1086
- Add-Feature 747
- Add-JobTrigger 547
- Add-LDAPObject 1036, 1274f.
- Add-LocalGroupMember 1081
- Add-Member 131, 499, 502, 1283
- Add-Migration 353
- Add-ODBCDSN 846
- Add-PSSnapin 378, 1302, 1307
- AddScript() 1360, 1363
- ADDSDeployment 1067, 1069
- Add-Type 434f., 503, 510, 719, 831, 964
- Add-VirtualHardDisk 671
- Add-VMDisk 1124
- Add-VMDrive 1124
- Add-VMHardDiskDrive 1101, 1113
- Add-VMNIC 1124
- Add-VMSwitch 1101
- Add-WBSystemState 746
- Add-WindowsCapability 907
- Add-WindowsFeature 745, 896, 900ff., 1067f.
- Administration
 - delegiert 674
 - webbasiert 357, 679
- Administrator 160, 305, 975
- Administratorrechte 282, 302, 305, 321ff., 491, 643, 739, 919, 975, 1151, 1157
- ADODB.Connection 1010
- ADO.NET 795, 803, 1009, 1029
- ADPowerShell 1038, 1044
- ADSI 1006, 1009ff., 1013, 1016, 1082
 - Bindung 1011f.
 - COM 1009, 1016
 - Container 1017
 - .NET 1003, 1005, 1009
 - Pfad 1011
- AdsPath 1028
- ADUser 1044
- Advanced Function 1269, 1278
- ADWS 1040
- AgentPC 1005
- AKS 1199
- Akte X 1004
- Aktivierung 673
- Aktivität 559, 563
- Alias 57, 70, 271, 691
- Aliaseigenschaft 123, 130
- AliasInfo 70
- AllNodes 613
- AllowClobber 36, 643
- AllowEmptyCollection 1278
- AllowEmptyString 1278
- AllowNull 1278
- AllowPrerelease 39
- AllSigned 159, 531
- Alpine 13
- Alvin Kersh 1004
- Amazon Web Service *siehe* AWS
- Änderungshistorie 800
- Animation 1257
- Ankerelement 199
- ANSI-Terminal 318
- Anwendungspool 1096, 1098
- Anzeigesprache 598
- Apache 1154
- AppDomain 435, 831
- AppendChild(). 764
- Apple Software Package 54
- AppLockerPolicy 912f.
- appSettings 359
- AppX 890
- Args 156, 181, 291
- ArrayList 208
- Artifact 1224
- AsJob 537, 556
- ASP.NET 452, 1137, 1180
- ASP.NET Core 373
- Assembly 419, 422, 433, 632, 637, 705, 1307, 1337
 - verbreiten 1381
- AssocClass 925
- ASSOCIATORS OF 468
- Assoziation 460
 - WMI 457, 460
- Asynchronous 966
- ATS 130, 485, 1010
- Attribut 1365, 1370, 1382
 - indiziert 1382
- Audio 428
- Aufgabe
 - geplant 542
- Aufzählung 441
- Aufzählungstyp 311
- Ausdruck 78
 - Regulär 197
- Ausdruckauflösung 189
- Ausdrucksmodus 78
- Ausführungsrichtlinie 158
- Ausgabe
 - mehrspaltig 252
 - unterdrücken 267
- Ausgabeobjekt 1311
- Auslagerungsdatei 673
- Authentifizierung 511, 1023, 1067
- AuthorizationRuleCollection 981f.
- AutoUpdate 867
- AWS 388
- Az 1192
- az aks 1205

- az.cmd 1194, 1204
 az extension 1205
 Az.Tools.Predictor 316
 Azure 79, 373, 388
 - Kontext 1195
 - Kubernetes Services 1199
 - Resource Group 1190,
 1196
 - SQL Server 1198
 - Subscription 1190, 1195
 - Web-App 1196
 Azure CLI 1194, 1204
 Azure Cloud Shell 362
 Azure Container Registry
 1154
 Azure Container Registry
 siehe ACR 1154
 Azure DevOps 167, 1223f.,
 1238
 Azure DevOps CLI 1224
 Azure Kubernetes Services
 siehe AKS
 AzureRM 1192
 Azure SQL 1198
 Azure Subscription 1190
 Azure-Webportal 1190
- B**
- BackgroundColor 182, 338,
 361
 Background Intelligent
 Transfer Service 965
 Background Intelligent
 Transfer Service *siehe* BITS
 965
 Backspace 191
 Backup 745f., 830
 Backup-GPO 1075
 Backup-SqlDatabase 828,
 830
 Base 1028, 1050
 bash 314, 362, 373, 400,
 1143
 BasicHtmlWebResponse-
 Object 952
 Basisauthentifizierung 279
 Basisimage 1182f.
 Basisklasse 799
- Batterie 875
 Bedingung 219
 Beep 191
 Beep() 431
 Befehl
 - Extern 57, 79
 Befehls-Add-On 89
 Befehlseingabefenster 41
 Befehlsgeschichte 680
 Befehlsmodus 78
 Befehlsobjekt 804
 begin 1269
 Begin 721
 BeginProcessing() 1301,
 1305
 Benutzer 452, 1033, 1366,
 1369
 - Active Directory 1019
 - anlegen 1022
 - lokal 1081
 - löschen 1024
 - umbenennen 1024
 - verschieben 1025
 Benutzerabmeldung 685
 Benutzeranmeldung 685
 Benutzerdaten lesen 1053
 Benutzer-DSN 848
 Benutzereingabe 508
 Benutzergruppe 1066
 Benutzerkennwort 1023
 Benutzerkontensteuerung
 siehe UAC
 Benutzerkonto 1053
 Benutzername 511
 Benutzerschnittstelle 450
 Berechnung 146
 Best Practice 1001
 Beta 40
 Betriebssystembasis-Image
 1125
 Bezeichner 1380
 Beziehung 1369
 Bibliothek 1377
 Big Endian 1028
 Bild 667
 Bildschirmschoner 505,
 1005
 Binärdatei 785
 Binäre Operation 213
- Binärmodul 1345
 Bindung
 - ADSI 1011
 - serverlos 1012
 - WMI 475
 Bing 1252
 BIOS 451
 Bitflag 213, 441
 BitLocker 747
 - Überblick 747
 Bitmap 719f.
 BITS 59, 965, 968
 Bitweise Operation 213
 Bitweises NOT 213
 Bitweises ODER 213
 Bitweises UND 213
 Blatt 1009
 Blockierung 162
 BMC 450
 Board 1224
 Boolean 231
 Boot-Konfiguration 451
 Bootstrap 438
 break 214, 216, 226, 234
 Breaking Change 603
 Build 1237
 bxor 704, 1083
 Bypass 159
 ByPropertyName 104f.
 Byte 184
 ByVal 104f.
 BZIP2 730
- C**
- C# 4, 168, 503, 505, 1224,
 1269, 1298, 1305, 1373
 C++ 1378f.
 C++/CLI 1298
 CAB 93
 Canvas 1254
 Carriage Return 191
 cat 400, 402
 CategoryView 229
 CATID 469
 CD 433
 Certificate
 - Zertifikat 994
 ChangeAccess 739

Checkpoint-Computer 871
 Checkpoint-VM 1100, 1115
 Children 1009
 ChildSession 296
 Chkdsk() 493
 CHKDSK 452
 chmod 401
 Chocolatey 894
 Chocolatey.org 893
 chown 405
 Chrome 605, 893f.
 CIL 15, 1376
 CIM 8, 450, 453
 - Repository 461
 CimClass 474f., 477, 486
 CimClassProperties 486
 CIM Explorer 369
 CimInstance 474f., 477,
 486
 CimInstanceProperties 486
 CimProperty 486
 CIM Query Language *siehe*
 CQL
 Cisco 450
 City 1045f.
 class 243
 ClassCreationEvent 470, 585
 ClassDeletionEvent 470, 585
 ClassModificationEvent 470,
 585
 Clear-BitLockerAutoUnlock-
 Funktion 748
 ClearCase 167
 Clear-Content 751
 Clear-DnsClientCache 938
 Clear-EventLog 280, 971
 Clear-History 681
 Clear-Host 228, 680
 Clear-Item 691
 Clear-RecycleBin 718
 Clear-Variable 180
 Click 1255
 Clipboard *siehe* Zwischen-
 ablage 513
 CliXml 768
 Close() 1254
 Cloud 1190
 CLR 15, 27, 1349, 1376
 cmd 325
 cmd.exe 100
 Cmdlet 1334
 CmdletBinding 1278, 1283
 Cmdlet Help Editor 1337
 Cmdlets 3
 cn 1019
 Codeausschnitt 333, 355
 Codeeigenschaft 123, 130
 Color 877
 COM 17, 443f., 718, 1379
 - Kategorie 469
 - Klasse 446
 - Komponente 452
 - Moniker 1011
 - Sicherheit 465
 Commandlet 3, 57, 69, 79,
 82, 100, 277, 375
 - binär 1298
 - erstellen 1269, 1298
 - Klasse 1301
 - Konvention 1324, 1342
 - Provider 272
 - Proxy 1288
 - Verkettung 1322
 Command Line Event
 Consumer 468
 Command Mode 78
 Comma-Separated Values
 755
 CommitChanges() 427,
 1007, 1016, 1023
 Common Information
 Model *siehe* CIM
 Common Intermediate
 Language *siehe* CIL 15
 Common Language Runtime
 siehe CLR 27
 Common Language Specifi-
 cation 1376
 Common Management
 Information Protocol 450
 Common Parameter 63
 Common Type System 1376
 compare 98
 Compare-Object 98, 149,
 703
 Compare-VM 1100, 1117f.
 CompatiblePSEditions 385
 Complete-BITSTransfer 966
 Complete-Transaction 378,
 522f., 525
 Component Object Model 4
 Component Object
 Model *siehe* COM
 Compress-Archive 728
 Computer 459, 1033, 1369
 Computergruppe 359
 ComputerInfo 857
 Computername 156, 391,
 865, 877
 Computerrichtlinie 519
 Computerverwaltung 857,
 919
 ConciseView 229
 configuration 607
 ConfigurationData 614
 ConfigurationID 624
 ConfigurationNamingContext
 1042
 confirm 65f.
 Confirm 64, 66, 878, 933,
 1050, 1053, 1283, 1285,
 1334
 ConfirmPreference 66, 1285
 conhost 325
 Connect-AzAccount 1195
 Connection 802
 Connect-VMNetworkAdapter
 1101
 ConsolePaneBackground-
 Color 338
 Console.WriteLine() 1329,
 1343
 Container 1017, 1075, 1129,
 1137
 Container-Klasse 1009
 continue 65, 226
 Continue 214, 216, 234, 238
 ConvertFrom-JSON 774
 ConvertFrom-Markdown
 771f.
 ConvertFrom-String 756
 ConvertFrom-StringData 596
 Convert-Html 771
 Convert-String 755
 ConvertTo-ContainerImage
 1187
 ConvertTo-CSV 755

- ConvertTo-DataTemplate 1257
 ConvertTo-JSON 774, 784
 ConvertTo-SecureString 513, 1067
 ConvertTo-WebApplication 1097
 ConvertTo-XML 769
 Convert-VHD 1101, 1112, 1115
 Convert-Xml 770
 copy 708
 Copy-ContainerFile 1171f.
 Copy-GPO 1074
 Copy-Item 234, 296, 691, 708, 714f., 852, 995
 Copy-NetFirewallRule 941
 Copy/Paste 311
 Copy-ToZip 730
 Copy-VMFile 1121
 CORBA 1379
 Core 395
 Count 117f., 207, 226
 Country 1046
 CPU 150
 CQL 468, 483
 Create() 494, 1289
 CreateCommand() 804
 CreateElement() 764
 CreateInstance() 876
 CreateObject() 447
 CreationTime 719, 1370
 Credential 1067
 Credentials 1048
 CSV 76, 508, 752ff., 855, 1062, 1093
 CSV-Datei 150
 CultureInfo 1311
 CurrentThread 109
 Cursor 799
 CustomerID 1286
 CVS 167
- D**
- Dana Scully 1004
 DataReader 798f., 804, 806, 808f.
 DataRow 130f.
- DataSet 798f., 806, 809 ff.
 Data Source Name *siehe*
 DSN 846
 DataTable 809
 Date 112
 Datei 247, 452, 1366
 – Eigenschaft 704, 719
 – kopieren 708
 – löschen 59
 – Rechte 452
 – verschieben 708
 Dateiname 57
 Dateinamenerweiterung 150, 152
 Dateisystem 3, 711, 979, 1369, 1381
 Dateisystemfreigabe 605, 732
 Dateisystemkatalog 718
 Dateisystemoperation 698, 1267
 Dateisystemstruktur 710
 Dateiversionsverlauf 744
 Datenabfrage 469
 Datenbank 461, 668, 795
 Datenbankmanagementsystem 802
 Datenbankverbindung 802
 Datenbankzeile 130
 Datenbankzugriff 795
 Datenbereich 595
 Datenbindung 1257
 Datendatei 595
 Datenmenge 271
 Datenquelle 846f.
 Datenquellensteuerelement 798
 Datentyp 171, 180f., 209, 1013, 1322, 1382
 – .NET 101
 – PowerShell 171
 – WMI 456
 Datenzugriff 803
 DateTime 112, 116, 120, 429
 Datum 203
 Day 112
 DB2 796, 846
 dBase 846
- DbCommand 804
 DbDataReader 806
 DBG 520
 DBNull 807
 DbProviderFactories 797
 DCOM 447, 461, 477, 868, 1317, 1330
 – Konfiguration 452
 dcromo 1067
 DDL 468
 debug 1329
 Debug 65f.
 Debugger 44
 Debugging 3, 43, 355, 520, 566
 Debug-Modus 520
 DebugPreference 66, 1329
 Decimal 184
 Deep Throat 1004
 Default Domain Policy 1079
 DefaultNamingContext 1042
 Deinstallation 883
 Delete() 1370
 Deleting 877
 Delimiter 752
 Deployment 1381
 Deployment Image Servicing and Management *siehe* DISM
 Description 1020, 1046, 1294
 DESCRIPTION 1287
 Deserialisierung 288
 Desired State Configuration *siehe* DSC
 Desktop 395, 452
 Desktop-Anwendungen 1377
 Desktop Management Task Force *siehe* DMTF 279
 Destruktor 1366, 1382
 Deutsche Telekom 531
 Developer PowerShell 350ff.
 Dezimalzahl 183
 DHCP 929f., 932
 Diagnose 517
 Dialogfenster 511, 1244
 diff 98
 Digest 279
 dir 404

- Dir 394
 Directory 979
 DirectoryEntry 130f., 173, 424f., 1007ff., 1013ff., 1022, 1025
 DirectoryInfo 121, 136, 499, 704, 979, 1370f.
 Directory Management Objects 1070
 DirectorySearcher 173, 1029
 DirectorySecurity 981
 DirectoryString 1020
 Disable-ComputerRestore 871
 Disable-ExperimentalFeature 393
 Disable-JobTrigger 547
 Disable-Mailbox 1086
 Disable-NetFirewallRule 941
 Disable-PnpDevice 875
 Disable-PSRemoting 285
 Disable-PSSessionConfiguration 295, 297
 Disable-VMIntegrationService 1104
 Disable-WindowsOptional- Feature 907, 909
 Disk Quotas 452
 DISM 907
 Dismount-VHD 1112
 DisplayName 1046
 Distinguished Name 1012, 1017, 1020, 1042, 1046
 Distributed COM 279
 Distributed Component Object Model *siehe* DCOM 447
 Distributed COM *siehe* DCOM 447
 Distributed File System 452
 Distributed Managements Objects *siehe* DMO
 DML 468
 DMO 830
 DMTF 279, 450
 DNS 306, 938, 1067, 1128f.
 DnsClient 931, 938
 DNS-Client 934
 DNSClient 933f.
- DNS-Konfigurations- einstellungen
 - Per WMI abfragen 935
 DNS-Server 452, 933, 937
 do 214
 Docker 51, 79, 374, 1125, 1128, 1130, 1137, 1141, 1149, 1151
 Docker CLI 1137, 1153
 docker cp 1171f.
 Docker EE 1152
 Docker Enterprise 1152
 docker events 1184
 docker exe 1166
 Dockerfile 1156, 1183
 Docker Hub 1154
 Docker-Image 1157
 docker info 1184
 docker inspect 1169, 1175
 docker logs 1184
 docker pull 1160
 docker run 1160
 docker system 1184
 DockPanel 1254
 DOCX 792
 Dokument 750
 Dokumentation
 - Active Directory 1022
 - .NET 96
 Dollarzeichen 146, 189
 Domain 941
 Domain Controller 1068
 Domain Specific Language *siehe* DSL
 Domäne 865, 1012, 1070, 1366, 1369
 - Beitritt 306, 865
 - hinzufügen 865
 DOS 4
 dotnet.exe 51, 233
 DotNetTypes.Format.ps1xml 247, 255f.
 Dot Sourcing 76, 157f., 530, 632, 1271f., 1275
 Double 184
 DownloadString() 426, 949
 DriveInfo 429
 Driver 849
 DriveType 439
- Druckauftrag 876
 - löschen 876
 Drucker 247, 267, 451, 877
 - verwalten 875, 877
 Druckerport 876
 Druckerverwaltung 875, 877, 929
 DSC 601
 DSC Pull Server 619, 624
 DSL 1262
 DSN 846, 848ff.
 DuplexingMode 877
 Duration 165
 DVD 433, 1118, 1123
- ## E
- echo 78f.
 Echo 78
 Edit-NanoServerImage 1129
 Eigenschaft 123f.
 Eigenschaftenzwischen- speicher 1007
 Eigenschaftssatz 123, 126
 Eingabe 508
 Eingabeaufforderung 682f.
 Eingabedialog 510
 Eingabemaske 1246
 Eingabeobjekt 1319
 Eingabesteuerelement 1257
 Eingabeunterstützung 332, 355
 Einzelobjekt 118f.
 Einzelschrittmodus 515
 elevated 919
 Elevated 160, 164, 919, 975, 1157
 Elevation 160
 Else 220
 Emacs 153, 314f., 356
 E-Mail 949, 960
 - Adresse 949
 - EmailEvent 471
 - senden 948
 EmailAddress 1046
 EmailEvent 471
 Enable-BitLocker 749
 Enable-ComputerRestore 871

- Enable-ExperimentalFeature 393
 Enable-JobTrigger 547
 Enable-NetFirewallRule 941, 947
 Enable-ODBCPerfCounter 846
 Enable-PnpDevice 875
 Enable-PSRemoting 283, 332, 877, 947
 Enable-PSSessionConfiguration 295, 297
 Enable-PSSessionConfiguration 297
 Enable-VMIIntegrationService 1104
 Enable-WindowsOptional- Feature 907, 909
 Encoding 752
 end 1269
 End 721
 EndProcessing() 1301, 1305
 endregion 341
 Enter-PSSession 285, 295, 300, 408, 520, 1131
 Enum 440
 EnumerateCollection 1313
 Enumeration 311
 Enumerationsklasse 439
 env 400, 863
 Environment 603
 Ereignis 1365, 1382
 - PowerShell 584, 593
 - WMI 466, 584
 Ereignisabfrage 470
 Ereigniskonsument 466ff.
 - permanent 466
 - temporär 466
 Ereignisprotokoll 150, 378, 451f., 462, 468, 519, 605, 969
 - Überwachung 471, 586
 Ereignisprovider 466
 Ereignissystem 584
 Error 181, 231, 242, 877
 ErrorAction 65f., 231, 238, 240f., 712
 ErrorActionPreference 66, 181, 231, 240, 1055
 ErrorBackgroundColor 317
 ErrorRecord 234, 237, 241 f.
 ErrorVariable 65, 241
 ErrorView 230
 Ethernet 931
 ETS 100, 123, 130, 1344
 Event 454
 EventConsumer 454
 Event *siehe* Ereignis
 EventViewerConsumer 467
 Example 1294
 EXAMPLE 1287
 Exception 234, 237, 243, 1325, 1331
 Exchange Management Shell 670, 1085
 Exchange Server 96, 452, 462, 1085
 ExecuteNonQuery() 804
 ExecuteReader() 804, 806
 ExecuteRow() 804
 ExecuteScalar() 804
 ExecutionPolicy 29 ff.
 EXIF 719
 Exists() 1015
 exit 214
 Exit-PSSession 287, 295
 Expand-Archive 728
 explorer.exe 745
 Export-Alias 76
 Export-CliXml 508, 768
 Export-Console 634, 1308
 Export-Counter 974
 Export-CSV 136, 391, 508, 755
 Export-ModuleMember 637
 Export-PfxCertificate 996
 Export-VM 1100, 1117
 Export-VMSnapshot 1116
 Express 824
 Expression 250
 Expression Mode 78
 Extended Reflection 100
 Extensible Application Markup Language *siehe* XAML
 Extrinsic Event 466
 facsimileTelephoneNumber 1046
 false 60, 66, 172, 181
 Farbe 317f., 394
 Fax 1046
 FBI 1004, 1025
 Feature 896
 FeatureOperationResult 902
 Fehler 65
 Fehlerausgabe 229
 Fehlerbehandlung 231, 1378
 Fehlerklasse 214, 237
 Fehlermeldung 60
 Fehlerstatus 231
 Fehlersuche 514, 1326
 Fehlertext 214
 Fernaufruf 280
 Fernausführung 156, 279
 - Hintergrundauftrag 540
 Fernverwaltung 279
 Fernzugriff 279
 Festplatte
 - virtuell 1112
 Festplattenverschlüsselung 747
 Fibre-Channel 1100
 Field 124
 File 979, 1368
 File History 744
 FileInfo 121, 136, 499, 704, 979, 1370f.
 FileInformation 915
 FileSecurity 979, 981
 FileSystem 666, 691
 FileSystemAccessRule 982
 FileSystemInfo 1370
 FileSystemObject 1368
 FileSystemRights 441
 FileSystemWatcher 592
 FileInfo 895
 filter 700
 Find() 1008
 Find-Module 644f.
 Find-Package 353, 893f.
 Firefox 605
 Firewall 403, 673
 Firewall-Regel 944

- First 111
 fish 373
 For 215, 217
 Force 63, 702, 1050
 Foreach 114, 145, 150f.,
 214, 217, 564, 1008, 1315
 Foreach-Object 98, 108,
 110, 114f., 126, 144, 146,
 207, 218, 225, 563, 1093,
 1313
 Foregroundcolor 88
 Forest 1070
 Format 250
 Formatkennzeichner 260
 Format-List 99, 248, 981
 Format-Table 112, 126, 150,
 248, 250, 257f., 981
 Format-Wide 247f., 252f.
 Format-Xml 761
 Form Feed 191
 Fortschrittsanzeige 269
 Fox Mulder 1004
 FoxPro 846
 Framework Class Library
 417
 Freigabe 741
 FTP 391
 FullAccess 739
 FullName 119, 1370
 function 57, 214, 223, 228,
 271
 Funktion 57, 222f., 271,
 375
 - eingebaut 228
 - fortgeschritten 1278
- G**
- GAC 433f.
 Ganzzahl 183
 Gast 1099
 Gateway 932
 GeneralizedTime 1020
 Geplante Aufgabe 542
 Gesamtstruktur 1070f.
 Geschäftsanwendung 1341
 GetAccessRules() 981 ff.
 Get-Acl 975, 979, 981, 992
 Get-ADComputer 1048
 Get-ADDomain 1071
 Get-ADDomainController
 1071
 Get-ADForest 1071
 Get-ADGroup 1048, 1066
 Get-ADGroupMember 1048,
 1066
 Get-ADObject 61, 1003,
 1035, 1048ff.
 Get-ADOptionalFeature 1071
 Get-ADOrganizationalUnit
 1048, 1052
 Get-ADPrincipalGroup-
 Membership 1066
 Get-ADRootDSE 1071
 Get-ADUser 1048, 1053f.
 Get-Alias 70f.
 Get-AppLockerFileInformation
 912
 Get-AppLockerPolicy 912f.
 GetAssemblies() 435
 Get-AuthenticodeSignature
 535
 Get-AzAks 1204, 1207
 Get-AzAppServicePlan 1196
 Get-AzLocation 1195
 Get-AzResource 1196
 Get-AzResourceGroup 1195
 Get-AzSqlServer 1196
 Get-AzSubscription 1195
 Get-AzWebApp 1196, 1198
 Get-BitLockerVolume 748f.
 Get-BITSTransfer 967
 Get-BITSTransfer 966
 Get-BPAModel 1001
 Get-BPAResult 1001
 Get-CDRomDrive 668, 873
 Get-ChildItem 58ff., 62, 99,
 136, 140, 145, 150f., 271,
 394, 697, 700, 728, 851,
 881
 - BitLocker 749
 Get-CimAssociatedInstance
 476
 Get-CimClass 449, 476, 483
 Get-CimInstance 378, 449,
 472, 476, 478ff., 494, 874,
 877
 Get-Clipboard 513, 647, 663
 Get-Command 69, 82f., 375,
 650
 Get-Computerinfo 865
 Get-ComputerInfo 857, 1311
 Get-Computername 1304
 Get-ComputerRestorePoint
 871
 Get-Container 1186f.
 Get-ContainerImage 1162,
 1186, 1189
 Get-Content 691, 712, 750,
 785, 855, 880, 1093, 1258
 Get-Counter 280, 972f.
 Get-Credential 89, 303, 511,
 675, 920, 949
 Get-Culture 194
 Get-DataRow 668, 819
 Get-DataTable 668, 818
 Get-Date 112, 116, 203f.
 Get-DHCPServer 34, 930
 Get-DirectoryChildren 668
 Get-DirectoryEntry 189, 668,
 1324
 Get-DirSize 1269
 Get-Disk 692, 694, 873,
 1313, 1317, 1319f., 1322,
 1330
 Get-DisplaySetting 875
 Get-DnsClient 935
 Get-DnsClientCache 938
 Get-DnsClient-Funktion
 - Beispiel 935
 Get-DnsClientServerAddress
 935
 Get-DomainController 34,
 40, 1012
 GetDrives() 428
 Get-DSCConfiguration 618
 Get-DVDDrive 671
 Get-Error 231, 242, 319
 Get-Event 588
 Get-EventLog 26, 150, 280,
 378, 969f.
 Get-ExCommand 1085
 Get-ExecutionPolicy 29
 Get-ExperimentalFeature 393
 Get-ExportedType 705
 GetFactoryClasses() 797
 Get-Filecatalog 96

- Get-FileHash 720f.
- Get-FileVersionInfo 705, 895
- Get-FloppyDrive 671
- Get-Flug 1341
- Get-Flugziele 1341
- Get-Font 865
- Get-FreeDiskSpace 694, 696
- Get-GPIInheritance 1079
- Get-GPO 1073ff.
- Get-GPOReport 1077
- Get-GPPermissions 1080
- Get-GPPrefRegistryValue 1080
- Get-GPRegistryValue 1080
- Get-GPResultantSetOfPolicy 1077
- Get-GPStarterGPO 1074
- Get-Help 86f., 90, 92, 95, 116, 168, 391, 395, 1269, 1287, 1296, 1336
- Get-History 395, 680
- Get-Host 395, 681
- Get-HotFix 280
- Get-Item 719, 851, 1092, 1095
- Get-ItemProperty 719, 851
- Get-Job 536, 538
- Get-JobTrigger 547
- Get-Keyboard 874
- Get-LDAPChildren 1036, 1274
- Get-LDAPObject 1036, 1274
- Get-LocalGroupMember 1081
- Get-LocalUser 1081
- Get-Location 70, 691
- Get-LogicalDiskInventory 692
- Get-LongDateString() 116
- Get-LongTimeString() 116
- Get-Mailbox 1085f.
- Get-Mailboxdatabase 1085
- Get-MarkdownOption 772
- Get-Member 98, 117, 122, 124, 136f., 429, 438, 444, 486, 1010, 1313
- Get-Members 1025
- Get-MemoryDevice 668, 873
- Get-Methode 124
- Get-Module 36, 251, 637, 648, 1345
- Get-MountPoint 34
- Get-MPComputerStatus 872
- Get-MultiTouchMaximum 875
- GetNames() 440
- Get-NanoServerPackage 1129
- Get-NetAdapter 931
- Get-NetAdapterBinding 931
- Get-NetFirewallAddressFilter 941
- Get-NetFirewallAddressFilter-Funktion 943
- Get-NetFirewallApplication-Filter 941
- Get-NetFirewallInterfaceFilter 941
- Get-NetFirewallInterface-TypeFilter 941
- Get-NetFirewallPortFilter 941
- Get-NetFirewallProfile 941f.
- Get-NetFirewallRule 941, 943, 945
- Get-NetFirewallRule-Funktion 943
- Get-NetIPInterface 932
- Get-NetworkAdapter 668, 873
- GetObject() 447
- Get-ODBCDriver 846
- Get-ODBCDSN 846
- Get-OSVersion 859
- Get-Package 894
- Get-PackageProvider 892f.
- Get-PackageSource 893
- Get-Passagier 1341
- Get-PipelineInfo 117, 121, 136, 1323
- Get-PnpDevice 875
- Get-PnpDeviceProperty 875
- Get-PointingDevice 874
- Get-PowerShellDataSource 1257
- Get-Printer 877f.
- Get-PrintJob 877f.
- Get-Process 42, 58f., 61f., 69f., 76, 99, 104, 114f., 120, 122, 124, 136, 146, 150, 231, 257f., 267, 280, 391, 400, 917, 1299, 1322, 1324
- Get-Processor 668, 873f.
- Get-PSBreakpoint 521
- Get-PscxUptime 860
- Get-PSDrive 692
- Get-PSProvider 274
- Get-PSReadLineKeyHandler 316
- Get-PSRepository 645
- Get-PSSession 295
- Get-PSSessionConfiguration 295f., 677
- Get-PSSnapIn 635
- Get-PswaAuthorizationRule 359
- Get-Random 135, 185
- GetRelated() 925
- Get-ReparsePoint 727
- Get-Service 88, 104, 116f., 120, 136, 156, 267, 280, 391f., 411, 923f., 1288f.
- Get-SHA1 721
- GetShortDateString() 116
- Get-ShortPath 707
- Get-ShortTimeString() 116
- Get-SmbShare 733, 739
- Get-SmbShareAccess 740
- Get-SoundDevice 873
- Get-SqlData 835
- Get-Storagegroup 1085
- Get-Tapedrive 873
- Get-TargetResource 631
- GetTempName() 446
- Getter 124f., 1382
- Get-TerminalSession 34
- Get-Timestamp() 412
- Get-TraceSource 517
- Get-Transaction 378, 522, 525
- GetType() 119, 181, 265, 385
- Get-Unique 85, 138
- Get-Uptime 412, 859f.

- Get-USB 874
 Get-USBController 668, 874
 Get-Variable 170, 174, 182
 Get-VHD 1112
 Get-VideoController 668,
 873
 Get-VirtualHardDisk 671
 Get-VM 1100, 1102, 1111,
 1119
 Get-VMBIOS-VM 1103
 Get-VMBuildScript 1122,
 1124
 Get-VMHost 671, 1101
 Get-VMMemory 1103
 Get-VMProcessor 1103,
 1120
 Get-VMSnapshot 1116
 Get-VMSummary 1124
 Get-VMThumbnail 1123f.
 Get-WBBackupSet 746
 Get-WBPolicy 746
 Get-WBSummary 746
 Get-WebApplication 1092
 Get-WebitemState 1097f.
 Get-Website 1092, 1095
 Get-WebvirtualDirectory
 1092
 Get-WindowsCapability 908
 Get-WindowsEdition 859
 Get-WindowsFeature 896,
 898, 900
 Get-WindowsOptionalFeature
 907f.
 Get-WinEvent 280
 Get-WmiObject 16, 280,
 378, 433, 449, 476, 478ff.,
 482, 696, 873, 879, 924,
 926, 929, 972, 1005
 Get-xDscOperation 626
 Gigabyte 184
 Git 79, 106, 167
 git.exe 233
 Github 36, 412
 GitHub 47, 167
 Gitternetz 1254
 GivenName 1045f.
 Gleichheitszeichen 211
 global 179, 1254
 Global 1254
- Global Assembly Cache *siehe*
 GAC
 GlobalSign 531
 Global Unique Identifier
 1017, 1379
 gm 98
 Go 1224
 Google 1252
 GPMC 1073
 Grafikkarte 452, 479
 Grant-SmbShareAccess 740
 Gravis 68, 107, 169, 191
 grep 373
 Grid 1254f.
 GridView 253
 Group 139, 603, 1010
 GROUP BY 468, 585
 Group-Object 98, 139, 150,
 970
 Gruppe 1033, 1048, 1062f.
 - Active Directory 1019
 - anlegen 1026
 - auflisten 1025
 - lokal 1081
 - Mitglied aufnehmen 1026
 Gruppenmitglieder 1048
 Gruppenmitgliedschaft 1027
 Gruppenrichtlinie 519, 685,
 1073, 1075, 1079
 - Vererbung 1079
 Gruppierung 139
 GUID 696
 Gültigkeitsbereich 171, 179
 GZIP 730
- H**
- Haltepunkt 43, 335, 520
 Hardlink 725
 Hardware 452, 668
 Hardwareverwaltung 873
 Hash-Tabelle 209, 425, 494
 Hashtable 209f., 425, 613f.,
 631
 Hashwert 718
 HAVING 468, 585
 Heimatordner 181
 Help-Info 93
 HelpMessage 1278
- Herausgeber 532, 535
 Here-String 187
 Herunterfahren 866
 Hexadezimalzahl 183
 hidden 245
 Hilfe 82
 Hilfetext 92
 Hintergrundauftrag 536
 Hintergrunddatentransfer
 965
 Hintergrundübertragungs-
 dienst 59
 History 680
 Hit Refresh 373
 HKCU 271
 HKEY_CURRENT_USER 853
 HKEY_LOCAL_MACHINE 853
 HKLM 271
 Home 181
 HomeDirectory 1046
 HomeDrive 1046
 Host 182, 338, 361, 681,
 1099
 Hosting 1357
 hostname.exe 865
 Hotfix 452
 Hour 112
 HTML 771f., 1094, 1246
 HtmlWebResponseObject
 951
 HTTP 279, 391, 619
 HTTPS 279, 391
 Hyper-V 33, 305, 653, 671,
 1099, 1121, 1125, 1128f.,
 1152, 1167
 - Überblick 1099
 Hyper-V-Container 1139
 Hyper-V-Integrationsdienste
 1104
 Hypervisor 1099
 Hyper-V-Modul
 - Überblick 1100
- I**
- i 212
 IADs 1007, 1009
 IADsComputer 1009
 IADsContainer 1007, 1009

- IADsGroup 1010
- IADsUser 1009, 1023
- idempotent 601f.
- Identität 1096
- Identity 976
- IdentityReference 984
- IDL 463
- IEnumerable 1008
- if 214, 220
- IIS 33, 273, 357, 1012, 1128f., 1156
 - Internet Information Services 1088
 - Nano 1135
- IISAdmin 925
- IISAdministration 1088, 1135
- IIS-Anwendung 1097
- IIS Management Service 1135
- ILSpy 438
- Impersonifizierung 511, 1014
 - WMI 465
- Implizites Remoting 299
- Import-Alias 76
- Import-AzAksCredential 1204, 1207
- Import-CliXml 508, 681, 768
- Import-Counter 974
- Import-CSV 150, 508, 703, 754, 1063, 1093
- Import-GPO 1075
- Import-INIFile 759
- Import-Module 566, 638, 650, 1351f.
- Import-PSSession 299
- Import-VM 1101, 1117f.
- IncludeUserName 918
- Index 205
- Indexer 1382
- Informix 796, 846
- Ingres 796
- Inheritance Flags 976
- INI-Datei 759
- inlinescript 560f.
- Innertext 765
- Input 181
- Inputbox 1245
- InputBox 435, 510, 1246
- InputBox() 510, 1245
- InputObject 104, 438, 1324, 1326
- inquire 65, 239
- Install-ADDSDomain-Controller 1069
- Install-ADDSForest 1069
- Installation 882
- Installationsordner 21, 128, 181
- Installationstechnologie 882
- Install-Module 36, 363, 642, 645, 1262
- Install-Package 353, 437
- Install-PswaWebApplication 358
- installutil.exe 1302, 1307
- Install-WindowsFeature 358, 620, 896
- InstanceCreationEvent 470, 585f., 588
- InstanceDeletionEvent 454, 466, 470, 585
- InstanceModificationEvent 466, 470, 585
- Instanz 425, 1366
- Instanziierung 1367
- Instanzmitglied 1383
- int 172
- Int32 172
- Int64 184
- INTEGER 1020
- Integrated Scripting Environment 41
- Integrated Scripting Environment *siehe* ISE 688
- IntelliJ 356
- IntelliSense 42, 58, 316, 348
- Interface 1368
- Interface Definition Language 463
- InternalHost 681
- InternalHostUserInterface 509
- International .NET Association XXXIII
- Internet Control Message Protocol 939
- Internet Information Server 452
- Internet Information Services 453, 619, 626, 909, 925, 1012, 1088
- Interpretermodus 319
- IntervalTimerInstruction 466
- Intrinsic Event 466
- InvalidOperationException 1008
- Invoke-BPAModel 1001
- Invoke-CimMethod 378, 476, 494
- Invoke-Command 285, 287, 289, 291 ff., 295, 302, 305, 520, 537, 1131
- Invoke-ContainerImage 1162
- Invoke-DbCommand 818
- Invoke-DBMaint 833
- Invoke-DBScalarCommand 818
- Invoke-Expression 213
- Invoke-History 680
- InvokeMethod() 475
- Invoke-Query 835
- Invoke-SqlBackup 842f.
- Invoke-SqlCmd 822f., 828ff., 840, 843
- Invoke-SqlCommand 668
- Invoke-WebRequest 952, 965
- Invoke-WebRequest 391, 951, 956, 962
- Invoke-WmiMethod 378, 476, 493
- IP
 - Adresse 149, 173, 306, 469, 605, 929, 932, 1198
 - Konfiguration 469
- IPAddress 173, 932
- ipconfig 79, 106
- IPHostEntry 938
- IP Routing 452
- IRQ 452
- Is64BitOperatingSystem 858
- Is64BitProcess 858
- ISA 468
- IsCoreCLR 391

ISE 153, 269, 336, 396,
 1131, 1357
 – Debugging 335, 520
 – Integrated Scripting
 Environment 331
 IsePack 666
 ISE Steroids 362
 IsInRole() 976
 IsLinux 391
 IsmacOS 391
 ISO 1099, 1106, 1109,
 1118
 Isolation 1167
 IsWindows 391
 Item() 1008
 Iteration 108
 IT-Visions XXXII

J

J 185
 Java 1224, 1378f.
 JavaScript 438
 JEA 674
 Jeffrey Snover 154
 Job
 – zeitgesteuert 546
 Job-Trigger 546
 – Zeitgesteuerte Jobs 546
 John Doggett 1004
 Join 193
 Join() 196
 Join-String 145, 196
 JPEG 719
 JScript .NET 503
 JSON 774, 962, 1237
 Junction 726
 Junction Point 725f.
 Just-In-Time-Compiler 1376

K

Kennwort 511f., 1006, 1023
 Kerberos 279, 465
 Kill() 114f.
 Kilobyte 184
 Klammer 61
 – rund 199
 Klammeraffe 78

Klasse 171, 428, 455, 459,
 499, 1366, 1368f., 1382
 – CIM 453
 – COM 422, 444
 – Commandlet 1301, 1321,
 1343
 – .NET 243, 417, 1301,
 1379, 1381
 – PowerShell 243f.
 – statisch 430
 – WMI 451, 453
 Klassendiagramm 1370f.
 Klassenhierarchie 1370
 Klassenmitglied 428, 1383
 Klassenname 424
 Known Host 409
 Kommandozeilenbefehl 79
 Kommentar 169
 Komponentenorientierung
 1376f.

Komponententest *siehe* Unit
 Test
 Komposition 1257
 Komprimierung 717, 728f.,
 731
 Konstante 788
 Konstruktor 1366, 1382
 Konstruktorfunktion 423ff.,
 444
 Kontakt 1033
 Konvention 1342
 Kopieren/Einfügen 311
 Kosinus 429
 Kreuzzuweisung 212
 ksh 373
 Kubernetes Command Line
 Client 1204

L

Label 250
 Language Server Protocol
siehe LSP
 LastAccessTime 1370
 LastExitCode 181, 231, 233
 Laufwerk 271f., 278, 692,
 852
 – virtuell 1112

LCOW 1137f.
 LDAP 1003, 1010, 1012,
 1028
 – Suchanfrage 1007, 1028
 – Suche 1034
 LDAP-Query 1029
 Leaf 1017
 Least Privilege 674
 Leistung 972
 Leistungsindikator 972f.
 Length 118
 Length 117, 499
 Limit-EventLog 280, 969,
 971

LinearGradientBrush 1255
 Linie 1370
 LINK 1287
 Linux 6f., 13, 25, 45, 53, 56,
 274, 373, 381, 400, 408,
 443, 641, 1138, 1147,
 1379
 – Container 1182f.
 – Dateisystem 404
 Linux-Container 1137
 Linux Foundation 373
 LinuxKit 1138
 Literal 260
 Little Endian 1028
 Lizenzierung 673
 Load-ContainerImage 1188
 LoadFrom() 434
 LoadWithPartialName() 434
 Logarithmus 429
 Log File Event Consumer 468
 Logoff 685
 Logon 685
 Lokalisierung 458, 598
 Loopback 282
 ls 400
 LSP 355

M

Machine.config 797
 MachineName 281, 1305
 macOS 7, 13, 25, 45, 55,
 274, 373, 381, 400, 443,
 641, 1379
 – Dateisystem 404

MailAddress 949
 MailMessage 948
 makecert.exe 531
 MAML 92, 1336
 man 400
 Manage-Bde 747
 Managed Code 1006
 Managed Object 450
 Managed Object Format 463
 Managed Provider 795, 1029
 ManagedThreadId 109
 ManagementBaseObject 474
 ManagementClass 130f.,
 173, 474f., 477, 480, 485,
 1313
 ManagementEventWatcher
 587
 Management Infrastructure
 API 474f.
 ManagementObject 131,
 173, 474f., 477, 480, 484,
 486f., 493, 925, 1313
 ManagementObjectCollection
 484, 1313
 ManagementObjectSearcher
 173, 480, 1313
 ManagementScope 586
 Mandatory 1278, 1317
 Manifest 639
 Manifestmodul 1345
 Markdown 771
 MarkdownInfo 772
 Maschinencode 1376
 Match 132
 MaximumDriveCount 278
 MaximumErrorCount 181
 maxSessionsAllowedPerUser
 359
 MCR 1154
 md 712, 720, 855
 measure 146
 Measure-Command 516f.,
 697
 Measure-Object 98, 146,
 150
 Measure-VM 1101
 Megabyte 184
 Mehrsprachigkeit 598
 Menge 118f.

Mercurial 167
 Merge-VHD 1112
 Message 234
 MessageBox 511, 1244
 MessageBoxButtons 1244f.
 MessageBoxDefaultButton
 1245
 MessageBoxIcon 1245
 Metamodell 460
 Metaobjekt 1017
 Methode 123f., 1365, 1370,
 1382
 - Getter 1382
 - Setter 1382
 Method *siehe* Methode
 Microsoft Access 811
 - Treiber 801
 Microsoft-Access 21
 Microsoft.ACE.OLEDB 801
 Microsoft Azure 1190
 Microsoft Certified Solution
 Developer XXXII
 Microsoft Command Shell 4
 Microsoft Container Registry
 siehe MCR 1154
 Microsoft Developer Network
 siehe MSDN 211
 Microsoft Excel 1063
 Microsoft Exchange 1068
 Microsoft Exchange Server
 605, 670, 1085
 Microsoft.GroupPolicy 1074
 Microsoft.Jet.OLEDB 801
 Microsoft.Management.
 Infrastructure.CimClass
 486
 Microsoft.Management.
 Infrastructure.CimClass-
 Properties 486
 Microsoft.Management.
 Infrastructure.CimInstance
 486
 Microsoft.Management.
 Infrastructure.CimInstance-
 Properties 486
 Microsoft.Management.
 Infrastructure.CimProperty
 486
 Microsoft Office 453, 788

Microsoft Outlook 922
 Microsoft.PowerShell 50
 Microsoft Print Ticket XML
 877
 Microsoft Shell 4
 Microsoft SQL Server 823,
 1198
 Microsoft System Center 472
 Microsoft.Update 867
 Microsoft.Vhd.PowerShell
 1112
 Microsoft.VisualBasic 510,
 1245f.
 Microsoft.VisualBasic.Inter-
 action 447, 510, 1246
 Microsoft.Win32 881
 Microsoft.Win32.RegistryKey
 851
 Microsoft Word 447
 Minute 112
 Mirantis 1152
 Mitglied 1365
 - .NET 1382
 - statisch 1383
 - WMI 490
 MMC 465
 Mock-Objekt 1266
 Modul 637, 646
 Module Browser 643
 Modulo 210
 MOF 463, 612, 614
 Monad 4
 Monica Reyes 1004
 Moniker 1011
 Mono 1379
 Month 112
 more 258
 Most Valuable Professional
 XXXII
 Mount-SpecialFolder 693
 Mount-VHD 1112
 move 708
 Move-ADObject 1048f.
 Move-Item 691, 708, 720
 Move-Mailbox 1086
 Move-VM 1101
 MSCL 15
 mscorlib.dll 1381
 MSDN 211, 1251

- MSDN Library 421
 MSFT_Printer 877
 MSFT_PrintJob 877
 MSFT_SmbShare 739
 MSFT_SmbShareAccess-
 ControlEntry 740
 MSFT_WUOperationsSession
 867, 870, 1133
 MSFT_WUSettings 867
 MSFT_WUUpdate 867
 MSH *siehe* Microsoft Shell
 MSI 628, 879, 881 ff., 890,
 1302
 MTA 1249
 Multithreading 109, 1378
 MySQL 796, 808, 822, 833,
 1154
 MySqlConnection 808
 MySqlLib 822
- N**
- Nachkommastelle 183, 429
 Name 104, 1046
 Namensauflösung 938
 Namensraum 455, 457,
 459 f., 979, 1070, 1379
 - ADSI 1012
 - .NET 1379, 1381
 - WMI 457, 460
 Namensraumhierarchie 1380
 NamespaceCreationEvent
 470, 585
 NamespaceDeletionEvent
 470, 585
 Namespace-ID 1011
 NamespaceModificationEvent
 470, 585
 Nano Server 12, 371, 1125,
 1128, 1164
 - IIS 1135
 - Installation 1129
 - Paketinstallation 1129,
 1133
 NanoWbem 449
 NativeObject 1008, 1010
 Navigation 271
 Navigation Provider 272
 Navigationsbefehl 274
- Navigationsmodell 691
 Navigationsparadigma 271
 Navigationsprovider 1003
 Negation 213, 442
 NetAdapter 931 f.
 NetSecurity 941
 NetSecurity-Modul
 - Überblick 940
 Netsh 934, 937, 944 f.
 Netsh
 - Per PowerShell aufrufen
 936
 netstat 79
 NetTCPIP 931 f.
 NetworkInterface 1311
 Network Load Balancing 452
 Netlaufwerk 451
 Netlaufwerksverbindung
 452
 Netzwerkadapter 1099
 Netzwerkcenter 284
 Netzwerkkarte 452, 469,
 929, 1366
 - Geschwindigkeit 934
 Netzwerkkartenprofil 947
 Netzwerkkonfiguration 673,
 929
 Netzwerkmanagement 449
 Netzwerkprofil
 - Per PowerShell setzen 947
 Netzwerkverbindung 452,
 605, 931 f.
 Neustart 306, 902
 Neustarten 866
 new() 423, 425, 444
 New-ADGroup 1048, 1066
 New-ADObject 1049
 New-ADOrganizationalUnit
 1048, 1052
 New-ADUser 1048, 1053,
 1056
 New-AppLockerPolicy 912,
 915
 New-AzAks 1204, 1206
 New-AzSqlDatabase 1198
 New-AzSqlServer 1198
 New-AzSqlServerFirewallRule
 1198
 New-AzWebApp 1197
- New-Buchung 1341
 New-Button 87, 1253, 1255
 New-CheckBox 1257
 New-CimInstance 476, 494
 New-CimSession 479
 New-CimSessionOption 479
 New-ComboBox 1257
 New-Container 1185
 New-DSCChecksum 625
 New-Elipse 1257
 New-Event 593
 New-EventLog 280, 969,
 971
 New-FileCatalog 718
 New-Fixture 1264
 New-GPLink 1075
 New-GPO 1074
 New-GPStarterGPO 1074
 New-Grid 1255
 New-Guid 423
 New-Hardlink 726
 New-HardwareProfile 671
 New-ISSite 1088, 1135
 New-Image 1257
 New-Int64Animation 1257
 New-Item 271, 691, 723,
 751, 851 f., 855
 New-Itemproperty 853, 855
 New-JobTrigger 547, 549
 New-Junction 727 f.
 New-Label 1250, 1255
 New Line 191
 New-Line 1257
 New-ListBox 1257
 New-LocalUser 28
 New-Mailbox 1086
 New-Mailboxdatabase 1086
 New-MediaElement 1257
 New-Menu 1257
 New-Module 637
 New-NanoServerImage
 1129 f.
 New-NetFirewallRule 941,
 944
 New-NetIPAddress 932 f.
 New-Object 422 ff., 431,
 433, 444, 788, 964, 1283,
 1383
 New-PasswordBox 1255

- New-ProgressBar 1257
 New-PSDrive 278, 852, 881, 1043
 New-PSSession 283, 285, 295, 298f., 302, 305, 678
 New-PSSessionConfiguration-File 674
 New-RadioButton 1257
 New-Rectangle 1257
 New-RichTextBox 1257
 New-ScheduledJobOption 550
 New-ScrollBar 1257
 New-SelfSignedCertificate 532
 New-Service 923, 927
 New-Shortcut 724
 New-Slider 1257
 New-SmbShare 65, 738f.
 New-StatusBar 1257
 New-Storagegroup 1086
 New-Storyboard 1257
 New-TextBlock 1257
 New-TextBox 1255
 New-TimeSpan 204
 New-TreeView 1257
 New-UrlShortcut 725
 New-VHD 1108, 1112f.
 New-ViewBox 1257
 New-VirtualDVDDrive 671
 New-VirtualNetworkAdapter 671
 New-VM 671, 1101, 1106, 1124
 New-VMSwitch 1101
 New-WebApplication 1097
 New-WebAppPool 1096
 New-WebServiceProxy 959 ff.
 New-Website 1088, 1092, 1135
 New-WebVirtualDirectory 1097
 New-Window 1255
 New-Zip 730
 NoAccess 739
 Node 764
 node.js 1154
 nodeJS 1224
- NoElement 140
 Non-Terminating Error 234, 1330, 1333
 NoProfile 530
 NormalView 229
 Northwind 824, 844
 Notation 1370
 – umgekehrt polnische 1028
 Notepad 153
 NoteProperty 499, 759
 Notes 1294
 NOTES 1287
 Notizeigenschaft 123, 128, 137
 Novell 1012
 Now 429
 Nslookup 934, 938
 NtAccount 984
 NTAccount 980, 983f.
 NT Event Log Event Consumer 468
 NTFS 462, 717
 NTLM 279
 NTSecurityDescriptor 1019
 Nuget 420, 436
 NuGet 36, 51, 353, 1224
 NuGet Package Manager 353
 null 147, 156, 176, 219, 223, 267
 Null Coalescing Assignment Operator 176
 Null Coalescing Operator 176, 392
 Null Conditional Operator 175f.
 Null-Wert 113
- O**
- Object 1371
 Object[] 118, 226
 ObjectCategory 1019, 1032, 1051
 ObjectClass 1019, 1032, 1045f., 1051
 ObjectGUID 1020, 1046
 ObjectiveC 1224
- ObjectSecurity 979
 ObjectSecurityDescriptor 1019
 ObjectSid 1019
 ObjectVersion 1021
 Objekt 204, 756, 1343, 1365, 1368f.
 – Dynamisch 499, 1269
 – .NET 1248
 – WMI 453
 Objektadapter 131, 484f., 798
 Objektassoziation
 – WMI 460
 Objektbaum 1369
 Objektidentifikation
 – ADSI 1011f.
 Objektmenge 702
 Objektorientierte Programmierung *siehe* OOP 1365
 Objektorientierung 100
 Objektorientierung *siehe* OO 1376
 Objekt-Pipeline 703
 Objekttyp 1366
 OCI 388
 OData 619
 ODBC 795f., 846, 849
 Office 1046
 OFS 181
 ogv 247
 Oh-my-Posh 329
 OK 1244
 OKCancel 1244
 OLEDB 795f., 1009
 – Provider 811, 1009, 1029
 OleDbCommand 804, 811
 OleDbConnection 802, 806, 811
 OleDbDataAdapter 811
 OMI 449, 478
 OMI *siehe* Open Management Infrastructure
 On_Click 1255
 OneGet 892
 OneLevel 1050
 ONELEVEL 1028
 OO 1365, 1376f.
 OOP 1365

- OOP *siehe* COP 98
 Open() 802
 Open Database Connectivity
 - Einstellung 452
 Open Management Infrastructure 406
 - OMI 478
 Open Source 5
 OpenSSH 406
 OpenView 472
 Operator 133, 196, 206, 210
 Optimize-VHD 1112
 Oracle 388, 796, 808
 Oracle Cloud Infrastructure
 siehe OCI 388
 OracleCommand 804
 OracleConnection 802
 Ordner 59, 81, 275, 697, 707f., 711
 - Dateisystem 452
 Ordnerstatistik 701
 Organisationseinheit 1033, 1062f.
 - anlegen 1027
 OSS
 - Open Source 5
 OutBuffer 65, 102
 Out-Default 247, 255, 1360, 1362
 Out-File 247, 268
 Out-GridView 89, 247f., 253f.
 Out-Host 247, 258
 Outlook 139, 789
 Outlook.Application 788
 Out-Null 247f., 267, 434
 Out-Printer 247, 267, 876
 OUTPUTS 1287
 Out-Speech 247, 269
 Out-SqlScript 833
 OutVariable 65, 148
 ov *siehe* OutVariable
- P**
- Packaged Script 350
 PackageManagement 640
 Page File 673
- Paketinstallation
 - Nano Server 1133
 Paketmanager 36
 Panel 1254
 PaperSize 877
 Papierkorb 718
 parallel 563
 Parallelisierung 109
 Parallelität 110
 Parameter 58f., 105, 227, 1278, 1294, 1317, 1326
 - Abkürzung 62f.
 - Skript 156
 PARAMETER 1287
 Parameterliste 156
 ParentSession 296
 ParsedHtml 952
 parsen 755
 Partition 1048
 PascalCasing 1380
 PASH 4
 PassThru 148, 1048
 passwd 402
 Pause 877
 PE 705
 PercentComplete 269
 Perforce 167
 Performance Counter
 Provider 972
 Performance Monitor 451, 462
 PERL 168
 Persistenz 564
 Pester 1261f., 1355
 Petabyte 184
 Pfad 459
 - ADSI 1011
 - Verzeichnisdienst 1011
 - WMI 455, 458f.
 Pfadangabe 275
 Pfeilspitze 1370
 Pflichtparameter 26
 PHP 168, 1224
 PhysicalDeliveryOfficeName 1021, 1046
 PIN 748
 Ping 79, 452, 939f., 1330
 Ping-Host 34, 40f., 939
 Ping-VM 1124
- Pipe 100
 Pipeline 3, 15, 99f., 120, 146, 267, 374, 489, 1224, 1237f.
 - Ausgabe 1311
 - Eingabe 1319
 Pipeline Processor 101, 1302
 PipelineVariable 65, 149, 257
 Pipelining 98, 210, 271
 Plattform Invoke 505
 Plattformunabhängigkeit 1376
 Platzhalter 61, 260
 Plug-and-Play 875
 Polymorphismus 1372
 Port 403
 Portable-Executable-Format
 siehe PE 705
 PoshConsole 366
 posh-git 329
 Position 1278, 1317
 Postfach 1086
 Postfix-Notation 1028
 PowerGUI 153, 331
 Power Management 462
 PowerShell 3, 70, 98
 - Extension 668
 - Hosting 3
 - Konsole 309
 - Laufwerk 271, 278, 852
 - Skriptsprache 168
 - Version 1.0 4
 - Version 2.0 4
 - Version 3.0 4
 - Version 4.0 4
 - Version 5.0 4
 - Version 5.1 5
 - Web Admin 679
 PowerShell Analyzer 347
 PowerShell Community
 Extensions 34, 662, 1003
 PowerShell Community
 Extensions *siehe* PSCX 662
 PowerShell Core 7, 17, 45
 - Funktionsumfang 375
 - installieren 45

- Konsole 394
- Version 5.1 12, 371, 1128
- Version 6.x 6, 56, 373
- WMI 449
- PowerShell Direct 304, 306, 1131
- PowerShell Editor Services 355f.
- powershell.exe 69, 651, 683, 1249
- powerShellExePath 398f.
- PowerShell Gallery 33, 36, 38, 604, 640
- PowerShellGet 38f., 640
- PowerShell Management Library for Hyper-V 1100, 1122
- PowerShellPlus 153, 331, 363, 1324
- PowerShell Remoting
 - Port 304
- PowerShell-Remoting 357
- PowerShell Remoting Protocol 279, 281, 406
- PowerShell Remoting *siehe* Remoting 279
- PowerShell Script Analyzer 342, 355
- PowerShell Web Access *siehe* PSWA
- PowerStudio 349
- Predictive IntelliSense 316
- PrimalScript 367
- Principal 980
- Printer 247
- Printing 877
- PrintManagement 877
- Print Ticket XML 877
- Private 941
- Privileg 465
- process 1269
- Process 104, 120, 136, 149, 603, 721, 1324
- ProcessRecord() 1301, 1305, 1317
- Professional Developer Conference 4
- profile 526
- ProfilePath 1046
- profile.ps1 434, 1271, 1326
- Profilskript 322, 526, 530
- Programmcodeanalyse 342
- Programmgruppe 452
- Programmiersprache 219
- Programmiersprachen-unabhängigkeit 1376
- Prompt 682
- PromptForChoice() 509
- Propagation Flags 977
- Property 124, 257, 1365, 1382
- Property Cache 1016
- PropertyCollection 1007
- PropertyDataCollection 475, 485f.
- PropertyGrid 1248
- PropertyName 1008
- PropertyValueCollection 1007, 1015
- ProtectedFromAccidental-Deletion 1046, 1050
- Protokolldatei 468
- Protokollierung 519
- Provider 274
 - ADO.NET 795
 - Dateisystem 691
 - PowerShell 272
 - Verzeichnisdienst 1003
 - WMI 453
- Proxy 960
- ProxyCommand 1289
- Proxy-Commandlet 299, 1288
- Prozedur 223
- Prozess 58f., 150, 452, 461
 - auflisten 267, 917
 - beenden 921
- ps 400
- PSAnsiRenderingFileInfo 394
- PSAzureProfile 1195
- PSBase 1010, 1016, 1018
- PSCmdlet 1301
- PSCodeGen 667
- PSComputerName 293
- PSConfig 653
- PSCredential 511, 920, 1067
- PSCustomObject 118, 137, 501, 753, 759, 1280, 1283
- PSCX 39f., 247, 513, 642, 653, 662, 723, 730f.
- PSCX *siehe* PowerShell Community Extensions
- psd1 597
- PSDriveInfo 692
- PSDSCRunAsCredential 607
- psedit 333, 340
- PSEdition 371, 395f.
- PSHome 181
- PSHost 181, 1359
- PSHostRawUserInterface 1359, 1361
- PSHostUserInterface 1359,
- PSImageTools 667
- psISE 338
- PSItem 99, 502
- PSModuleAutoLoading-Preference 181, 649
- PSModulePath 390, 400, 638, 1346
- PSObject 131, 1362f.
- PSReadline 313, 315, 328f., 394f.
- PSRemotingJob 537
- PSRP *siehe* PowerShell Remoting Protocol
- PSRSS 667
- PSScheduledJob 542
- PSScriptRoot 158
- PSSession 294
- PSSnapIn 1303
- PSStyle 318
- PSSystemTools 667, 692, 874f.
- psUnsupportedConsole-Applications 337
- PSUserTools 667
- PSVariable 170
- PSVersion 396
- psversiontable 52, 371
- PSWA 357f.
- Public 941, 1317
- Public Network 284
- Pull-ContainerImage 1157
- Pull Request 412

Punktnotation 112, 426,
490
Push-ContainerImage 1189
Put() 490f.
pv *siehe* PipelineVariable
148
pwsh 47, 394
pwsh.exe 321
Python 168, 1224

Q

Quantifizierer 199
Quantor 199
QueryDialect 483
Quest 653, 668

R

Range 193
Raspberry Pi OS 13
Raspbian 13
RawUI 338
RDP 306, 947
ReadAccess 739
Read-Host 508, 513, 557,
559
Receive-Job 536, 538f., 556
Rechenleistung 150
Recovery Console 1125f.
recurse 60, 700, 1048
recursive 1048, 1053
Red Hat Enterprise Linux 13
Redirection *siehe* Umleitung
Redo 316
Redstone 5
REFERENCES OF 468
Referenzkopie 211f.
Refresh() 1370
RefreshCache() 1016
RefreshFrequencyMins 624
Regel 911, 944
region 341
Register-ArgumentCompleter
312
Register-CimIndicationEvent
476, 592
Register-DnsClient 935
Register-Event 590

Register-ObjectEvent 743
Register-PackageSource 644,
893
Register-PSSessionConfigura-
tion 295, 298, 674
Register-ScheduledJob 548f.
Register-WmiEvent 587, 592
Registrierungsdatenbank 3,
271, 277f., 452, 616, 851,
881
- Schlüssel 851
Registry 460, 462, 603, 851
RegistryKey 881, 979
RegistrySecurity 981
RegistryValueChangeEvent
466, 470, 585
Regulärer Ausdruck 173,
197, 717
Relative Distinguished Name
1017f.
Remote Desktop Protocol
siehe RDP
Remote Procedure Call *siehe*
RPC
Remote Server Administration
Tools 1040
Remoting 156, 279, 306,
408, 877, 1131
- Implizit 299, 1288
Remove-ADGroup 1066
Remove-ADGroupMember
1048, 1066
Remove-ADObject 1048ff.
Remove-ADOrganizationalUnit
1052
Remove-ADUser 66, 1053
Remove-Alias 76
Remove-AzAks 1204, 1223
Remove-AzSqlDatabase
1199
Remove-AzSqlServer 1199
Remove-AzWebApp 1198
Remove-Buchung 1341
Remove-CimInstance 476,
495
Remove-Computer 865
Remove-Container 1186
Remove-ContainerImage
1186, 1188f.

Remove_DirectoryEntry
1334
Remove-DirectoryEntry 668,
1324
Remove-Event 590
Remove-EventLog 280, 969
Remove-GPLink 1075
Remove-GPO 1074
Remove-GPPrefRegistryValue
1080
Remove-GPRegistryValue
1080
Remove-Item 59, 63, 66, 76,
691, 708, 852, 855
Remove-ItemProperty 854
Remove-Job 536, 539
Remove-JobTrigger 547
Remove-LDAPObject 1036,
1274
Remove-LocalUser 1081
Remove-Module 638, 652
Remove-NetFirewallRule 941,
945
Remove-NetFirewallRule-
Funktion 945
Remove-NetIPAddress 932
Remove-NetRoute 932
Remove-ODBCDsn 846
Remove-PrintJob 877f.
Remove-PSBreakpoint 521
Remove-PSSession 295f.
Remove-PSSnapin 378
Remove-PswaAuthorization-
Rule 359
Remove-Service 928
Remove-SmbShare 63, 66,
739
Remove-Variable 180
Remove-VM 1101, 1111
Remove-VMSnapshot
1116
Remove-WebApplication
1098
Remove-WebAppPool 1098
Remove-Website 1098
Remove-WebVirtualDirectory
1098
Remove-WindowsCapability
907

Remove-WindowsFeature 896, 902	Revoke-SmbShareAccess 740	Schlüssel 271
Remove-WmiObject 476, 495	RHEL 13	Schlüsselattribut
Rename-ADObject 1048f.	Richtlinienergebnisbericht 1077	- WMI 455
Rename-Computer 865	Robocopy 714ff.	Schnittstelle 243, 799, 1368, 1372
Rename-Drive 696	Rolle 896, 1128	- .NET 1383
Rename-GPO 1074	Rollendienst 896	Schriftart 865
Rename-Item 708	rootcimv2 459	Schtasks.exe 542
Rename-NetAdapter 934	RPC 280	Schwichtenberg, Holger XXXII
Rename-NetFirewallRule 941	RSAT 33, 1100	Scope <i>siehe</i> Gültigkeitsbereich 171
Rename-VM 1101	RSS 667, 950	script 179
Rename-VMSnapshot 1116	Ruby on Rails 1154	Script 618
Repair-VM 1101	Rückgabeobjekt 1311	Script Analyzer 342
Replace 195	RuleCollection 913	Scripting.FileSystemObject 446
Replikation 451	Run-ContainerImage 1162, 1181, 1185	ScriptMethod 499
Repo 1224	RunNow 548	Script-Migration 353
Repository 461	Runspace 347, 674	ScriptPaneBackgroundColor 339
requires 164	RuntimeException 243	SDL 677, 733, 993
Resize-VHD 1101, 1112		sealed 1371
Resolve-Assembly 435		Searcher 867
Resolve-DnsName 938		SearchScope 1050
Resolve-DNSName-Funktion - Beispiel 938		Secure Socket Layer 358f., 952, 961
Resolve-DsnName 935		Secure String 747f.
Resolve-Host 938		Security Descriptor 976
Resolve-Path 275		Security Descriptor Definition Language 297
ResponseHeaders 426		Security Identifier 976, 980f., 983f.
Ressource 603		Security Service Provider 465
Ressource Group 1190		Select
REST 962		- PowerShell 136
Restart-Computer 280, 866, 902		SELECT 468, 584
Restart-PrintJob 877		- WQL 468ff.
Restart-Service 66, 301, 304, 923, 926		SelectNodes() 762f.
Restart-VM 1101		Select-Object 26, 98f., 111, 126, 131f., 136, 138, 140, 150, 258, 488, 763, 1326
Restore-ADObject 1049		SelectSingleNode() 762f.
Restore-Computer 871		Select-String 80, 105, 716, 751
Restore-DscConfiguration 616		Select-Xml 762f.
Restore-GPO 1075		Self-Contained App <i>siehe</i>
Restore-VMSnapshot 1116		SCA 46
Restricted 158		
Restricted Runspace 674		
Resume-PrintJob 877		
Resume-Service 923, 926		
Resume-VM 1101		
return 214, 277, 1302		

Semantic Versioning 173, 603
 Semaphore 979
 Semikolon 150, 919
 Send-MailMessage 948f.
 Send-SmtpMail 948
 sequence 560
 Serialisierung 120, 288
 Seriennummer 861
 Server 1048
 ServerCertificateValidation-Callback 961
 Server Management Objects *siehe* SMO
 Server Manager 1067
 ServerRemoteHost 361
 ServerURL 624
 Service 603
 ServiceController 120, 288, 923
 Serviceorientierung 1377
 Serviceorientierung *siehe* SOA 1376
 ServicePointManager 952
 Session 447, 867
 sessionState 359
 Set-Acl 975, 989, 992
 Set-ADAccountPassword 1048
 Set-ADGroup 1066
 Set-ADObject 1049f.
 Set-ADOrganizationalUnit 1052
 Set-ADUser 1053, 1055
 Set-Alias 75
 Set-AppLockerPolicy 912
 Set-AuthenticodeSignature 533
 Set-AzAks 1204
 Set-AZContext 1195
 Set-AzWebApp 1197
 Set-BPAResult 1001
 Set-CimInstance 476, 492
 Set-Clipboard 513
 Set-Content 691, 751, 785, 949
 Set-DataRow 819
 Set-DataTable 668, 819
 Set-Date 204
 Set-DistributionGroup 1086
 Set-DnsClientServerAddress 933ff.
 Set-ExecutionPolicy 29 ff., 152, 158, 531, 535
 Set-FileTime 704f.
 Set-GPIInheritance 1079
 Set-GPLink 1075
 Set-GPPermissions 1080
 Set-GPPrefRegistryValue 1080
 Set-GPRegistryValue 1080
 Set-Info() 1007, 1016
 Set-Item 301, 691
 Set-ItemProperty 704, 854, 947
 Set-JobTrigger 547
 Set-Location 70, 271, 691, 851
 Set-Mailbox 1086
 Set-MarkdownOption 772
 Set-Methode 124
 Set-NetFirewallPortFilter 941
 Set-NetFirewallProfile 941f.
 Set-NetFirewallRule 941, 945
 Set-NetIPInterface 932
 Set-ODBCDriver 846
 Set-ODBCDsn 846
 Set-PrintConfiguration 877
 Set-PSBreakpoint 520f.
 Set-PSDebug 174, 514f.
 Set-PSReadLineKeyHandler 316
 Set-PSReadlineOption 315, 395
 Set-PSReadLineOption 316f.
 Set-PSSessionConfiguration 295
 Set-ScheduledJob 548
 Set-Service 392, 923, 926f.
 Set-StrictMode 174
 Set-TargetResource 631
 Setter 124f., 1382
 Set-TraceSource 518
 Set-Variable 170, 182, 1253f.
 Set-VHD 1112
 Set-VM 1101, 1103
 Set-VMMemory 1124
 Set-VMProcessor 1140, 1144
 Set-VolumeLabel 696
 Set-VolumneLabel 34
 Set-WmiInstance 476, 491
 Set-WSManQuickConfig 284
 SHA256 718
 Shell 3, 99
 Shell.Application 718, 731
 Shielded VM 1129
 ShouldProcess() 1334f.
 Show() 1244
 Show-Command 89, 334
 ShowDialog() 1258
 Show-EventLog 280, 970
 Show-HyperVMenu 1123
 Show-Markdown 772
 Show-NetFirewallRule 941
 Show-Service 280
 Show-VMMenu 1123
 ShowWindow 90
 Shutdown 685
 Sicherheit
 - COM 465
 - Dateisystem 452, 462
 - PowerShell 158
 - WMI 465
 Sicherheitsabfrage 1283, 1334
 Sicherheitsbeschreibung 976
 Sicherheitseinstellung 975
 Sicherheitsmodell 3
 Sicherheitsrichtlinie 159
 SID 976
 Side-by-Side Executing 1378
 Signatur
 - digital 531
 Signieren 531
 SilentlyContinue 65, 239, 712, 1055
 Simple Network Management 451, 462
 Simple Object Access Protocol *siehe* SOAP 279, 959
 Sitzung 294ff.
 Sitzungskonfiguration 297, 674, 678
 Skip 111
 SkipEditionCheck 385

SkipNetworkProfileCheck	Speicherbereinigung 1378	Start-Service 289, 923, 926
285, 947	Speicherverbrauch 800	Start-Sleep 165
Skript 152, 154	Speicherverwaltung 1378	Start-Transaction 522ff.
- PowerShell 152	Spitzname 1368	Start-Transcript 557
Skriptausführungsrechte 29	Spoolerdienst 877	Startup 685
Skriptausführungsrichtlinie	Spooling 877	Start-VM 1101, 1109, 1124
30	Sprachausgabe 247, 269,	Start-WBBackup 746
Skriptblock 179, 287, 1253	445	Start-WBFileRecovery 746
Skriptdatei 152	Sprache 598	Start-WBHyperVRecovery
Skripteigenschaft 123, 128	Sprachkürzel 598	746
Skriptfenster 41	SQL 468, 850	Start-WBSystemState-
Skriptmodul 1345	SQLASCOMMANDLETS 823	Recovery 746
Skriptsprache 1269	Sqlcmd.exe 829	Start-WBVolumeRecovery
SMB 1128	SqlCommand 804, 828	746
SMO 824, 830f., 840, 842,	SqlConnection 425, 802,	Start-Webitem 1098
844	806, 808	Start-Website 1097
Smoking Man 1004	SqlDataSourceEnumerator	static 243
SMTP 948f.	798	Status 269, 877
SmtpClient 948	SQLPS 273, 822, 824, 828	Stop 65, 238
Snap-in 1302ff., 1326, 1337	SQLPSX 822, 824, 833f.,	Stop-Computer 280, 866
Snap-In 378, 632, 637, 651	840	Stop-Container 1186
Snapshot	SQL Server 373, 796, 822f.	Stop-Job 536, 539
- Hyper-V 1115	- Agent 840	Stop-Process 114, 557, 917,
SNA Server 462	- Laufwerk 825	921, 1324
Snippet 333	SqlServerCe 796	StopProcessing() 1301
SOA 1376	SqlServerCmdletSnapin100	Stop-Service 59, 923, 926
SOAP 279, 461, 959	823	Stop-VM 1101
Software 451f., 672	SQL Server Management	Stopwatch 412, 517
- deinstallieren 883	Studio 824, 840	Stop-WBJob 746
- installieren 628, 630, 882	SSH 407f.	Stop-Webitem 1098
- inventarisieren 879	sshs 407	Stop-Website 1097
- verwalten 879	SSL <i>siehe</i> Secure Socket	Stored Procedure 810
Softwareentwickler 421	Layer	Streaming 102
Softwareentwicklungs-	STA 1249	StreetAddress 1046
plattform 1377	StackPanel 1254	String 187, 195
Softwarekomponente 419,	StackTrace 181	Subnetzmaske 932
433, 436	Stammzertifizierungsstelle	Substring() 193
Softwarepaket 893	532, 535	SubTree 1050
Softwarequelle 893	Standarddrucker 267	SUBTREE 1028
Software Restriction Policy	Standardkonsole 309	Subversion 167
911	Start-BITSTransfer 966	Suche
Sortieren 137	Start-Container 1185	- Active Directory 1028
Sort-Object 98ff., 102, 131,	Start-ContainerProcess 1166	- Assembly 705
137, 140, 145, 150, 226,	Start-DscConfiguration 609	- LDAP 1009
1302	Start-Job 536ff.	- Verzeichniseintrag 1018
Speak() 445	Startmenü 452	- XML 762
Speech 247	Start-Process 542, 787,	SupportsShouldProcess
SpeechSynthesizer 269	917f., 920	1334
Speicher 118	Start-PSSession 540	Surname 1046

- Suse 13
 Suspend 64
 Suspend-PrintJob 877
 Suspend-Service 923, 926
 Suspend-VM 1101
 Switch 60, 214, 220, 1099
 SwitchParameter 1344
 Sybase 796
 Symbolic Link 725, 727
 SymLink 727f.
 Synopsis 1294
 SYNOPSIS 1287
 Syntaxfarbhervorhebung 355
 System 1380f.
 System32 144, 150f.
 System ACL 980
 System.ApplicationException 243
 Systemattribut - WMI 455
 System.Boolean 275
 System Center Virtual Machine Manager 671
 System.Collections.Hashtable 209
 System.Console 431, 680
 System.Data 423
 System.Data.ODBC 796, 849
 System.Data.OleDb 796, 803
 System.Data.OLEDB 796
 System.Data.OracleClient 796, 803
 System.Data.SqlClient 796, 803, 828
 System.Data.SqlClient. SqlConnection 425
 System.Data.SqlServerCe 796
 System.DateTime 203, 425f., 429
 System DBNull 807
 System.Diagnostics 517
 System.Diagnostics.EventLog 969
 System.Diagnostics.Process 100, 114, 255, 917, 1322
 Systemdienst 103, 452, 923 - auflisten 469 - überwachen 471
 System.DirectoryServices 423, 1003, 1005ff., 1009f., 1013, 1019, 1022, 1029, 1070, 1082
 System.DirectoryServices.ActiveDirectory 1070
 System.DirectoryServices.DirectoryEntry 425, 427
 System.dll 1381
 System-DSN 848
 Systemende 685
 System.Enum 440
 System.Environment 286, 858f., 861, 863, 975, 1305f., 1311
 SystemEvent 470, 585
 System.Globalization.CultureInfo 682
 System.Int32 172, 184
 System.IO.Compression 729, 731
 System.IO.Directory 979
 System.IO.DirectoryInfo 499, 697, 701
 System.IO.DriveInfo 428, 431, 433, 439, 692, 694
 System.IO.DriveType 439
 System.IO.File 979
 System.IO.FileInfo 499, 697, 701, 719
 Systemklassen - WMI 454
 Systemmanagement 449
 System.Management 423, 1313
 System.Management.Automation 96, 1301, 1303, 1313, 1362
 System.Management.Automation.Cmdlet 1301
 System.Management.Automation.PathInfo 275f.
 System.Management.Automation.PSCustomObject 753
 System.Management.Automation.PSDriveInfo 692
 System.Management.ManagementObject 204
 System Management Server 462
 System.Math 429
 System.Media.SoundPlayer 428
 Systemmodul 38
 System.Net.Mail 948f.
 System.Net.WebClient 426f., 949f., 952, 962
 System.Object 120, 501, 1092, 1322, 1326
 SystemParametersInfo 505
 System.Random 186, 425
 System.Reflection 433
 System.Security 980
 System.Security.AccessControl 979
 System.ServiceProcess.ServiceController 923, 925, 1322
 Systemstart 685
 System.String 187, 192, 402, 1305
 System.TimeSpan 165, 204
 System.Type 119, 181, 265
 System.ValueType 211
 Systemwiederherstellung 871
 System.Windows 1249
 System.Windows.FontStyle 1252
 System.Windows.Forms 719, 1244
 System.Xml.Node 764
 Sysvol 685

T

- Tab Completion 311
 Tabellenformatierung 250
 TabPanel 1254
 Tabulator 191
 Tabulatorvervollständigung 311
 Tag 1154
 Tag-ContainerImage 1188
 TAR 730

- TaskScheduler 666
 TCP/IP 932
 tcsh 373
 Team Foundation Server *siehe*
 TFS
 Tee-Object 147f.
 Teilmenge 135
 Telnet 285
 Terminal Services 452
 Terminating Error 234, 1330
 Ternary Operator 221
 Terrabyte 184
 Test-32Bit 874
 Test-64Bit 874
 Test-AppLockerPolicy 912,
 914
 Test-Assembly 705
 Test-Connection 41, 105,
 939f.
 Test-CustomerID 1285
 Test-DbConnection 818
 Test-DscConfiguration 610
 Test-FileCatalog 95, 718
 Test-IsAdmin 975
 Test-JSON 774, 783
 Test-ModuleManifest 639
 Test-Path 275
 Test Plan 1224
 Test-PswaAuthorizationRule
 359
 Test-ServiceHealth 1085
 Test-SqlScript 833
 Test-TargetResource 631
 Test-UserGroupMembership
 1027
 Test-VHD 1101, 1112
 Test-Xml 761
 Textanzeige 1257
 Textdatei 150, 750
 Texteingabefeld 510, 1245
 TextInfo 194
 Textpad 153
 TFS 167, 1224
 Thawte 531
 this 502, 1254
 Thread 109, 563
 Thread-Modell 1249
 ThrottleLimit 110, 293
 throw 214, 243
 ThrowTerminatingError()
 1330
 Thumbprint 994
 TIFF 719
 TimeOfDay 112
 TimeSpan 516
 Tivoli 472
 TLS *siehe* Transport Layer
 Security
 ToLower() 194
 Ton 431
 ToString() 120f., 385, 432,
 1322, 1371
 TotalProcessorTime 266
 ToTitleCase() 194
 ToUpper() 194
 TPM 747
 Trace-Command 688
 Tracing 517
 Transaktion 378, 522
 Transformation 1257
 Translate() 984
 Transport Layer Security 952,
 961
 trap 214
 Trap 231, 234, 236f.
 Treiber 672
 – ODBC 847
 Trigger 546
 Troubleshooting Pack 997
 true 172, 181, 874
 Trusted Host 302
 Trusted Platform Module
 siehe TPM
 Trustee 976
 Try...Catch 1055
 Try-Catch-Finally 231, 237
 T-SQL 828f.
 Tuva 1125
 Typ 172, 417, 1366, 1381
 – Namensgebung 1381
 Typadapter 172, 206
 Typbezeichner 172
 Type Cast *siehe* Typkonver-
 tierung 178
 types.ps1xml 76f., 128, 131
 Typisierung 171
 Typkennzeichner *siehe*
 Typbezeichner 172
 Typkonvertierung 138, 178
 Typname *siehe* Typbezeichner
 172

U

- UAC 30, 155, 160, 321, 919
 Überladung 228
 Ubuntu 13, 53, 1137,
 1141f., 1175
 ufw 403
 Umgebungsvariable 451
 – Linux 400
 Umlaut 752
 Umleitung 268
 Unblock-File 162
 Undefined 159
 Undo 316
 Undo-Transaction 378, 522,
 524f.
 UniformGrid 1254
 Uninstall-Package 353, 894
 Uninstall-WindowsFeature
 896
 Unit Test 1261
 Universal Coordinated Time
 456, 492
 Unix 3f., 99f., 168, 691, 725
 Unlock-BitLocker 749
 Unregister-PSSessionConfigu-
 ration 295, 298, 677
 Unrestricted 159
 Unterbrechungsfreie Strom-
 versorgung *siehe* USV
 Unternamensraum 1379
 Unterordner 121, 700
 Unteroutine 222
 Unterschlüssel 271
 until 214
 Update 452, 672
 – Einstellungen 870
 – installieren 869
 – suchen 868
 UpdateColl 867
 Update-Database 353
 Update-Help 93
 Update-Module 38, 1262
 UseBasicParsing 952
 UsePropertyCache 1016

- User 603, 1009, 1019, 1033
 user32.dll 505
 User Account Control *siehe*
 Benutzerkontensteuerung
 UserDomainName 975
 UserName 975
 User Settings 398
 UseTestCertificate 358
 UseTransaction 523
 using 560
 USV 875
 UTF8 752
 UWP 889
- V**
- ValidateCount 1278
 Validate-CustomerID 1285
 ValidateLength 182, 1278,
 1344
 ValidateNotNull 1278, 1344
 ValidatePattern 182, 1278,
 1344
 ValidateRange 183, 1278
 ValidateScript 182, 1278
 ValidateSet 183
 ValueFromPipeline 1278,
 1319, 1321, 1326
 ValueFromPipelineByProperty-
 Name 1278, 1321
 ValuesCollection 1007
 ValueType 211
 Variable 44, 119, 148, 170,
 181, 260, 271
 - Auflösung 188
 - eingebaut 181, 391
 - vordefiniert 181, 391
 - Workflow 561
 Variablenauflösung 188f.,
 260
 Variablenkennzeichner 148,
 170
 VariablenTypisierung 171
 VB 503, 505
 VBA 788
 Verb 787
 Verbindungszeichenfolge
 425, 802, 810
 Verbose 65f., 1329
 VerbosePreference 66, 181,
 1329
 VerbsCommon 1342
 VerbsCommunications 1343
 VerbsData 1343
 VerbsDiagnostic 1343
 VerbsLifeCycle 1343
 VerbsSecurity 1343
 Vererbung 460, 1370, 1383
 Vererbungsdiagramm 1370
 Vererbungshierarchie 473,
 1044, 1370
 - WMI 460
 Vergleich 149
 Vergleichsoperator 132
 Verifikation 1378
 VeriSign 531
 Verknüpfung 724
 Verzeichnisattribut 1015
 Verzeichnisdienst 130, 462,
 668, 1017, 1029
 Verzeichnisdienstklasse
 1017
 Verzeichnisobjekt 1013,
 1018
 Verzweigung 147
 VHD 1111f., 1123, 1129
 VHDX 1108, 1112, 1123
 Video 1257
 View 256
 Vim 356
 VirtualHardDisk 1112
 Virtualisierung 1099, 1152
 VirtualizingStackPanel 1254
 Virtual Machine Platform
 1143
 Virtuelle Maschine *siehe* VM
 Virtuelles System 1099
 Virus 159
 Visual Basic 503, 1373
 Visual Basic 6.0 1378
 Visual Basic for Applications
 siehe VBA
 Visual Basic .NET 4, 1298
 Visual Studio 348ff., 353,
 566, 1137, 1182, 1259,
 1298f., 1326
 - Container 1179
 - Erweiterung *siehe* 1299
 Visual Studio Code 56, 153,
 354, 396, 1157
 Visual Studio Team Services
 siehe VSTS
 Visual Web Developer Express
 1299
 VMBus 305
 VMGUID 305
 VMName 305
 void 267, 447
 VolumeLabel 427
 Volume Shadow Copy Service
 siehe VSS 744
 VSCode
 - Visual Studio Code 354
 VSCode-PowerShell 354, 396
 VSI 1299
 VSIX 1299
 VSS 744
 VSTS 167, 1224
 VT100 773
- W**
- WaitForAll 631
 WaitForAny 631
 WaitForSome 631
 Wait-Job 536, 539
 Wait-Process 922
 Walter Skinner 1004
 WarningAction 65, 238
 WarningVariable 65
 Warnung 65
 WAS 925
 WBEM 8, 449
 WCOW 1137
 WDAC 846
 WebAdministration 1088,
 1090, 1135
 Webanwendung 1377
 Web Based Enterprise
 Management 449f.
 WebClient 965
 Webdienst 959
 Weblog 950, 1384
 Webserver 273, 1093
 Webservice 959, 961
 Web Service Description
 Language 960

- Web Services Description Language *siehe* WSDL 959
- Website 956, 1093, 1098
- Well-Known GUID 1013
- Well-Known Object 1013
- Well-Known Security Principal 985
- WellKnownSidType 986
- Werkzeug 309
- Wertemenge 205
- Wertkopie 211f.
- WhatIf 63, 65f., 181, 708, 878, 1283, 1334
- WhatIfPreference 66
- Where() 134f.
- WHERE 468
- Where-Object 70f., 98ff., 115, 131, 133ff., 146, 150, 267, 924, 1302, 1313, 1326
- while 214
- Whistler 458
- whoami.exe 323
- Width 250
- Wiederherstellungspunkt 871
- Win32 453
- Win32_Account 1005
- Win32_ACE 736
- Win32-API 505
- Win32_Battery 875
- Win32_Bios 860
- Win32_BootConfiguration 860
- Win32_CDRomdrive 873
- Win32_CDRomDrive 488
- Win32_CodecFile 881
- Win32_ComponentCategory 469
- Win32_ComputerShutdown-Event 466, 470, 585
- Win32_Computersystem 858
- Win32_Currenttime 204
- Win32/Desktop 1005
- Win32_Diskdrive 873
- Win32_Group 1005
- Win32_Keyboard 874
- Win32_LocalTime 204
- Win32_LogicalDisk 459f., 470, 493, 692, 694f., 1313
- Win32_MappedLogicalDisk 696
- Win32_MemoryDevice 873
- Win32_NetworkAdapter 873
- Win32_NetworkAdapterConfiguration 469, 929, 935f.
- Win32_NTLogEvent 469, 471, 586
- Win32_OpenSSH 406
- Win32_OperatingSystem 858f.
- Win32_OSRecoveryConfiguration 861
- Win32_PerfRawData 972
- Win32_PerfRawData_PerfOS_Processor 972
- Win32_PerfRawData_PerfProc_Process 972
- Win32_PingStatus 939f.
- Win32_PointingDevice 874
- Win32_PowerManagement-Event 470, 585
- Win32_Printer 875, 877, 929
- Win32_Printjob 875f.
- Win32_Process 586
- Win32_Process 873f., 1119
- Win32_ProcessStartTrace 470, 585
- Win32_Product 109, 879, 881ff.
- Win32_Quickfixengineering 881
- Win32_SecurityDescriptor 736
- Win32_Service 469, 471, 586, 923
- Win32_Share 733f.
- Win32_SoundDevice 873
- Win32_SystemConfiguration-ChangeEvent 466, 470, 585
- Win32_Tapedrive 873
- Win32_TCPIPPrinterPort 875f., 929
- Win32_Trustee 736
- Win32_USBController 874
- Win32_UserAccount 149, 459, 1005
- Win32_VideoController 479, 488, 873, 875
- Win32_Volume 696
- Win32_WindowsProduct-Activation 861
- window 1254
- Windows – Rolle 896, 1128
- Windows 8 483
- Windows 8.1 656
- Windows 9x 461
- Windows 10 5, 12, 20, 305, 310, 658 – Anniversary Update 5
- Windows 2000 458
- Windows Activation Service *siehe* WAS
- WindowsApps 50
- Windows-Authentifizierung 832
- Windows Communication Foundation 452
- Windows-Container 1137
- Windows Container *siehe* Docker 1138
- Windows Data Access Components *siehe* WDAC
- Windows Defender 872, 1129
- Windows Driver Model 462
- Windows Explorer 745, 855
- Windows Firewall 673, 940, 946 – Im Netzwerk abfragen 946 – Per PowerShell konfigurieren 940
- Windows Forms 350, 1244, 1246, 1248
- WindowsIdentity 975
- Windows Installer 462, 911
- Windows Management Framework 20, 450, 877
- Windows Management Instrumentation 15
- Windows ME 461
- Windows Nano Server 12, 1125, 1154
- Windows PowerShell XXIV, 3, 6

- Windows PowerShell Community Extensions 662
 Windows Pre Installation Environment *siehe* WinPE
 Windows Presentation Foundation *siehe* WPF
 Windows Remote Management 279
 Windows Remote Management *siehe* WinRM 279
 Windows Script Host 4, 17, 158
 Windows Server 1709 12
 Windows Server 2003 4, 458, 461, 1028
 Windows Server 2012 483, 654, 1067
 Windows Server 2012 R2 362, 656
 Windows Server 2016 5, 12, 20, 305, 310, 658
 Windows Server 2019 12
 Windows Server 2022 12, 34, 324
 Windows Server Container 1138
 Windows Server Core 331, 1154
 Windows Subsystem for Linux *siehe* WSL 1138
 Windows Troubleshooting Platform 997
 Windows Universal Platform *siehe* UWP
 Windows Update 867, 870
 - Agent API 1133
 - Nano Server 1133
 Windows Vista 1374
 Windows Workflow Foundation 378
 Windows XP 15, 458, 468
 WinMgmt.exe 461 f.
 WinPE 747
 WinPSCompatSession 385
 WinRM 279, 281 f., 284, 461, 536, 866
 WinSCP 964
 WITHIN 468, 585
 WKGUID 1013
- WMI 3, 8, 15, 204, 279, 378, 449, 453, 460, 480, 1313, 1317, 1330
 - Class Explorer 473
 - Command Shell 15
 - Data Query 469
 - Ereignis 466
 - Event Query 468, 470
 - Klasse 473
 - Namespace 457
 - Object Browser 472 f.
 - Query Language 468, 482
 - Repository 461, 466, 484
 - Schema 460, 469
 - Schema-Query 469
 - Steuerung 461
 WMI API 474
 WMIClass 449, 480
 WMI Object Browser 473
 WMI Query Language *siehe* WQL
 WMISEARCHER 449, 480, 482
 Word 139, 447, 792
 Workflow 552, 557 ff., 566
 - Designer 567
 - Einschränkungen 557
 - Persistenz 564
 - Verschachtelt 562
 WorkflowInfo 566
 WorkingSet 130
 WorkingSet64 101
 Work Item 1230
 Workspace Settings 398
 World Wide Wings 1341
 Wörterbuch 139
 WPF 87, 89 f., 253, 331, 350, 366, 391, 435, 556, 666, 1244, 1249, 1257, 1378
 WPF PowerShell Kit 666, 1249
 WPK 666, 1249
 WQL 468, 483, 584, 1313
 WrapPanel 1254
 Write-BZip2 730
 Write-Clipboard 513
 WriteDebug() 1330
- Write-Error 259
 WriteError() 1330, 1333
 Write-EventLog 280, 378, 969, 971
 Write-GZip 730
 Write-Host 88, 247, 259, 361, 559
 WriteObject 1306, 1315
 WriteObject() 1302, 1313
 Write-Output 78
 Write-Progress 269, 497
 Write-Tar 34, 730
 WriteVerbose() 1330, 1333
 Write-Warn 259
 WriteWarning() 1330, 1333
 Write-Zip 730
 WScript.Shell 724
 WSDL 959
 WSH 1010
 WSL 373, 1138, 1143
 WSMan 273, 303
 WS-Management 279 f., 282, 303, 461, 474, 477, 479
 WT_SESSION 328
 Wurzelnamensraum 1379
 www.IT-Visions.de 653, 668, 679, 692, 818

X

- x64 1099
 x86 1099
 XAML 556, 566, 1257
 XamlReader 1258
 XCopy-Deployment 1378, 1381
 xDscDiagnostics 625
 xDscWebService 620
 X-Files 1004
 XFileServer 1004
 XML 92, 508, 634, 741, 759 f., 768 f., 962, 1336
 XML Application Markup Language *siehe* XAML
 XmlAttribute 763
 XmlDocument 769
 XmlElement 763
 XML-Schema 761
 XML-Webservice 1379

XPathDocumentNavigator 763	Zeichenkette 187f., 196, 260, 1323, 1343	Zertifikatsspeicher 3, 271, 994
XslCompiledTransform 770	- ersetzen 195 - leer 128 - Operation 193 - trennen 195 - verbinden 196	Zertifikatsverwaltung 531f., 535
Y		
YAML 1180, 1237	Zeichensatz 752	ZIP 728ff., 792
Year 112	Zeilenumbruch 68, 107	ZipFile 729, 731
YesNo 1244	- Pipeline 107	zsh 373
YesNoCancel 1244	Zeitmessung 516	Zufallszahl 185f.
Z		
Zahl 183	Zeitplandienst 452	Zugriffsrechteleiste 975, 981
Zahlenliteral 184	Zertifikat 358, 531, 994 - selbst signiert 532	Zugriff verweigert 492
	Zertifikatsdatei 534	Zuweisungsoperator 211
		Zwischenablage 513
		Zwischencode 1376
		Zwischenschritt 146
		Zwischenspeicher 800