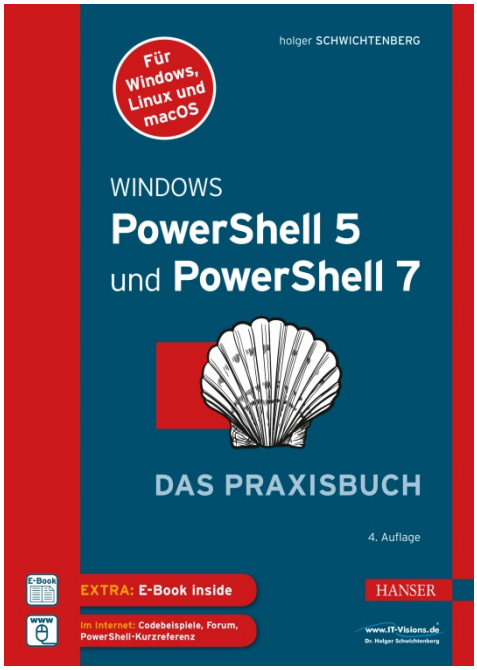
HANSER



Leseprobe

zu

„Windows PowerShell 5 und PowerShell 7“

von Holger Schwichtenberg

Print-ISBN: 978-3-446-45913-7
E-Book-ISBN: 978-3-446-46081-2
E-Pub-ISBN: 978-3-446-46507-7

Weitere Informationen und Bestellungen unter
http://www.hanser-fachbuch.de/978-3-446-45913-7
sowie im Buchhandel

© Carl Hanser Verlag, München

Inhalt

Vorwort ..... **XXV**

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 7.0 7
1.5 Motivation zur PowerShell 8
1.6 Betriebssysteme mit vorinstallierter PowerShell 11
1.7 Einflussfaktoren auf die Entwicklung der PowerShell 12
1.8 Anbindung an Klassenbibliotheken 14
1.9 PowerShell versus WSH 14

2 Erste Schritte mit der PowerShell **17**

2.1 Windows PowerShell herunterladen und auf anderen Windows-Betriebssystemen installieren 17
2.2 Die Windows PowerShell testen 21
2.3 Woher kommen die PowerShell-Befehle? 30
2.4 PowerShell Community Extensions (PSCX) herunterladen und installieren 31
2.5 Den Windows PowerShell-Editor „ISE“ verwenden 34
2.6 PowerShell 7 installieren und testen 36

3 Einzelbefehle der PowerShell **45**

3.1 Commandlets 45
3.2 Aliase 58
3.3 Ausdrücke 66
3.4 Externe Befehle (klassische Kommandozeilenbefehle) 67
3.5 Dateinamen 69

4 Hilfefunktionen ... 71
4.1 Auflisten der verfügbaren Befehle ... 71
4.2 Praxistipp: Den Standort eines Kommandozeilenbefehls suchen ... 72
4.3 Anzahl der Befehle ... 73
4.4 Volltextsuche ... 75
4.5 Erläuterungen zu den Befehlen ... 76
4.6 Hilfe zu Parametern ... 77
4.7 Hilfe mit Show-Command ... 78
4.8 Hilfefenster ... 80
4.9 Allgemeine Hilfetexte ... 81
4.10 Aktualisieren der Hilfedateien ... 82
4.11 Online-Hilfe ... 84
4.12 Fehlende Hilfetexte ... 85
4.13 Dokumentation der .NETKlassen ... 86

5 Objektorientiertes Pipelining ... 89
5.1 Befehlsübersicht ... 89
5.2 Pipeline-Operator ... 90
5.3 .NET-Objekte in der Pipeline ... 91
5.4 Pipeline Processor ... 92
5.5 Pipelining von Parametern ... 94
5.6 Pipelining von klassischen Befehlen ... 96
5.7 Zeilenumbrüche in Pipelines ... 98
5.8 Schleifen ... 99
5.9 Zugriff auf einzelne Objekte aus einer Menge ... 102
5.10 Zugriff auf einzelne Werte in einem Objekt ... 103
5.11 Methoden ausführen ... 105
5.12 Analyse des Pipeline-Inhalts ... 107
5.13 Filtern ... 122
5.14 Zusammenfassung von Pipeline-Inhalten ... 127
5.15 „Kastrierung“ von Objekten in der Pipeline ... 127
5.16 Sortieren ... 128
5.17 Duplikate entfernen ... 129
5.18 Gruppierung ... 130
5.19 Objekte verbinden mit Join-String ... 136
5.20 Berechnungen ... 137
5.21 Zwischenschritte in der Pipeline mit Variablen ... 138
5.22 Verzweigungen in der Pipeline ... 139
5.23 Vergleiche zwischen Objekten ... 141
5.24 Weitere Praxislösungen ... 142

6 PowerShell-Skripte ... 145
6.1 Skriptdateien ... 145
6.2 Start eines Skripts ... 147
6.3 Aliase für Skripte verwenden ... 149
6.4 Parameter für Skripte ... 149
6.5 Skripte dauerhaft einbinden (Dot Sourcing) ... 151
6.6 Das aktuelle Skriptverzeichnis ... 151
6.7 Sicherheitsfunktionen für PowerShell-Skripte ... 152
6.8 Skripte mit vollen Rechten (Elevation) ... 154
6.9 Blockierte PowerShell-Skripte ... 155
6.10 PowerShell-Skripte im Kontextmenü des Windows Explorers ... 156
6.11 Anforderungsdefinitionen von Skripten ... 158
6.12 Skripte anhalten ... 158
6.13 Versionierung und Versionsverwaltung von Skripten ... 158

7 PowerShell-Skriptsprache ... 161
7.1 Hilfe zur PowerShell-Skriptsprache ... 161
7.2 Befehlstrennung ... 162
7.3 Kommentare ... 162
7.4 Variablen ... 163
7.5 Variablenbedingungen ... 176
7.6 Zahlen ... 177
7.7 Zeichenketten (Strings) ... 179
7.8 Reguläre Ausdrücke ... 189
7.9 Datum und Uhrzeit ... 195
7.10 Objekte ... 196
7.11 Arrays ... 197
7.12 ArrayList ... 200
7.13 Assoziative Arrays (Hash-Tabellen) ... 201
7.14 Operatoren ... 202
7.15 Überblick über die Kontrollkonstrukte ... 206
7.16 Schleifen ... 207
7.17 Bedingungen ... 212
7.18 Unterroutinen (Prozedur/Funktionen) ... 214
7.19 Eingebaute Funktionen ... 220
7.20 Fehlerausgabe ... 220
7.21 Fehlerbehandlung ... 221
7.22 Laufzeitfehler erzeugen ... 233
7.23 Objektorientiertes Programmieren mit Klassen ... 233

8 Ausgaben ... 237
8.1 Ausgabe-Commandlets ... 237
8.2 Benutzerdefinierte Tabellenformatierung ... 240
8.3 Benutzerdefinierte Listenausgabe ... 242
8.4 Mehrspaltige Ausgabe ... 242
8.5 Out-GridView ... 243
8.6 Standardausgabe ... 245
8.7 Einschränkung der Ausgabe ... 248
8.8 Seitenweise Ausgabe ... 248
8.9 Ausgabe einzelner Werte ... 249
8.10 Details zum Ausgabeoperator ... 252
8.11 Ausgabe von Methodenergebnissen und Unterobjekten in Pipelines ... 255
8.12 Ausgabe von Methodenergebnissen und Unterobjekten in Zeichenketten ... 255
8.13 Unterdrückung der Ausgabe ... 256
8.14 Ausgaben an Drucker ... 257
8.15 Ausgaben in Dateien ... 257
8.16 Umleitungen (Redirection) ... 258
8.17 Fortschrittsanzeige ... 258
8.18 Sprachausgabe ... 259

9 Das PowerShell-Navigationsmodell (PowerShell Provider) ... 261
9.1 Einführungsbeispiel: Navigation in der Registrierungsdatenbank ... 261
9.2 Provider und Laufwerke ... 262
9.3 Navigationsbefehle ... 265
9.4 Pfadangaben ... 265
9.5 Beispiel ... 267
9.6 Eigene Laufwerke definieren ... 268

10 Fernausführung (Remoting) ... 269
10.1 RPC-Fernabfrage ohne WS-Management ... 270
10.2 Anforderungen an PowerShell Remoting ... 271
10.3 Rechte für PowerShell-Remoting ... 272
10.4 Einrichten von PowerShell Remoting ... 273
10.5 Überblick über die Fernausführungs-Commandlets ... 275
10.6 Interaktive Fernverbindungen im Telnet-Stil ... 276
10.7 Fernausführung von Befehlen ... 277
10.8 Parameterübergabe an die Fernausführung ... 281
10.9 Fernausführung von Skripten ... 282
10.10 Ausführung auf mehreren Computern ... 283
10.11 Sitzungen ... 284

10.12 Implizites Remoting ... 289
10.13 Zugriff auf entfernte Computer außerhalb der eigenen Domäne ... 290
10.14 Verwaltung des WS-Management-Dienstes ... 294
10.15 PowerShell Direct für Hyper-V ... 295
10.16 Praxislösung zu PowerShell Direct ... 297

11 PowerShell-Werkzeuge ... 301
11.1 PowerShell-Standardkonsole ... 301
11.2 Windows Terminal ... 311
11.3 PowerShell Integrated Scripting Environment (ISE) ... 313
11.4 PowerShell Script Analyzer ... 325
11.5 PowerShell Analyzer ... 330
11.6 PowerShell Tools for Visual Studio ... 331
11.7 PowerShell Pro Tools for Visual Studio ... 333
11.8 NuGet Package Manager ... 333
11.9 Visual Studio Code mit PowerShell-Erweiterung ... 334
11.10 PowerShell-Erweiterungen für andere Editoren ... 337
11.11 PowerShell Web Access (PSWA) ... 337
11.12 Azure Cloud Shell ... 343
11.13 ISE Steroids ... 343
11.14 PowerShellPlus ... 344
11.15 PoshConsole ... 347
11.16 PowerGUI ... 348
11.17 PrimalScript ... 349
11.18 PowerShell Help ... 351
11.19 CIM Explorer for PowerShell ISE ... 351
11.20 PowerShell Help Reader ... 352
11.21 PowerShell Remoting ... 353

12 Windows PowerShell Core 5.1 in Windows Nano Server ... 355
12.1 Installation ... 355
12.2 PowerShell-Skriptsprache ... 355
12.3 Werkzeuge ... 356
12.4 Fehlende Funktionen ... 356

13 PowerShell 7 für Windows, Linux und macOS ... 357
13.1 Motivation für den Einsatz der PowerShell 7 auf Linux und macOS ... 357
13.2 Basis der PowerShell 7 ... 358
13.3 Identifizierung der PowerShell 7 ... 359
13.4 Funktionsumfang der PowerShell 7 ... 359
13.5 Entfallene Befehle in PowerShell 7 ... 362

13.6 Erweiterungsmodule nutzen in PowerShell 7 ... 367
13.7 Geänderte Funktionen in PowerShell 7 ... 372
13.8 Neue Funktionen der PowerShell 7 ... 374
13.9 PowerShell 7-Konsole ... 378
13.10 VSCode-PowerShell als Editor für PowerShell 7 ... 378
13.11 Verwendung von PowerShell 7 auf Linux und macOS ... 383
13.12 PowerShell-Remoting via SSH ... 389
13.13 Dokumentation zur PowerShell 7 ... 393
13.14 Quellcode zur PowerShell 7 ... 394

Teil B: PowerShell-Aufbauwissen ... 397

14 Verwendung von .NET-Klassen ... 399
14.1 .NET versus .NET Core ... 399
14.2 Ermitteln der verwendeten .NET-Version ... 400
14.3 .NET-Bibliotheken ... 401
14.4 Microsoft Developer Network (MSDN) ... 403
14.5 Überblick über die Verwendung von .NET-Klassen ... 404
14.6 Erzeugen von Instanzen ... 404
14.7 Parameterbehaftete Konstruktoren ... 406
14.8 Initialisierung von Objekten ... 408
14.9 Nutzung von Attributen und Methoden ... 408
14.10 Statische Mitglieder in .NET-Klassen und statische .NET-Klassen ... 410
14.11 Generische Klassen nutzen ... 414
14.12 Zugriff auf bestehende Objekte ... 415
14.13 Laden von Assemblies ... 415
14.14 Liste der geladen Assemblies ... 418
14.15 Verwenden von Nuget-Assemblies ... 419
14.16 Objektanalyse ... 421
14.17 Aufzählungstypen (Auflistungen/Enumerationen) ... 422

15 Verwendung von COM-Klassen ... 427
15.1 Unterschiede zwischen COM und .NET ... 427
15.2 Erzeugen von COM-Instanzen ... 428
15.3 Abruf der Metadaten ... 428
15.4 Nutzung von Attributen und Methoden ... 429
15.5 Liste aller COM-Klassen ... 430
15.6 Holen bestehender COM-Instanzen ... 431
15.7 Distributed COM (DCOM) ... 431

16 Zugriff auf die Windows Management Instrumentation (WMI) .. 433
16.1 Einführung in WMI 433
16.2 WMI in der PowerShell 459
16.3 Open Management Infrastructure (OMI) 461
16.4 Abruf von WMI-Objektmengen 461
16.5 Fernzugriffe 462
16.6 Filtern und Abfragen 463
16.7 Liste aller WMI-Klassen 466
16.8 Hintergrundwissen: WMI-Klassenprojektion mit dem PowerShell-WMI-Objektadapter 467
16.9 Beschränkung der Ausgabeliste bei WMI-Objekten 471
16.10 Zugriff auf einzelne Mitglieder von WMI-Klassen 472
16.11 Werte setzen in WMI-Objekten 473
16.12 Umgang mit WMI-Datumsangaben 475
16.13 Methodenaufrufe 476
16.14 Neue WMI-Instanzen erzeugen 477
16.15 Instanzen entfernen 478
16.16 Commandlet Definition XML-Datei (CDXML) 479

17 Dynamische Objekte 483
17.1 Erweitern bestehender Objekte 483
17.2 Komplett dynamische Objekte 485

18 Einbinden von C# und Visual Basic .NET 487

19 Win32-API-Aufrufe 489

20 Benutzereingaben 493
20.1 Read-Host 493
20.2 Benutzerauswahl 494
20.3 Grafischer Eingabedialog 495
20.4 Dialogfenster 496
20.5 Authentifizierungsdialog 496
20.6 Zwischenablage (Clipboard) 498

21 Fehlersuche 499
21.1 Detailinformationen 499
21.2 Einzelschrittmodus 500
21.3 Zeitmessung 501
21.4 Ablaufverfolgung (Tracing) 502
21.5 Erweiterte Protokollierung aktivieren 503

21.6 Script-Debugging in der ISE ... 505
21.7 Kommandozeilenbasiertes Script-Debugging ... 505

22 Transaktionen ... 507
22.1 Commandlets für Transaktionen ... 507
22.2 Start und Ende einer Transaktion ... 508
22.3 Zurücksetzen der Transaktion ... 509
22.4 Mehrere Transaktionen ... 510

23 Standardeinstellungen ändern mit Profilskripten ... 511
23.1 Profilpfade ... 511
23.2 Ausführungsreihenfolge ... 513
23.3 Beispiel für eine Profildatei ... 513
23.4 Starten der PowerShell ohne Profilskripte ... 514

24 Digitale Signaturen für PowerShell-Skripte ... 515
24.1 Zertifikat erstellen ... 515
24.2 Skripte signieren ... 517
24.3 Verwenden signierter Skripte ... 519
24.4 Mögliche Fehlerquellen ... 519

25 Hintergrundaufträge („Jobs“) ... 521
25.1 Voraussetzungen ... 521
25.2 Architektur ... 521
25.3 Starten eines Hintergrundauftrags ... 522
25.4 Hintergrundaufträge abfragen ... 523
25.5 Warten auf einen Hintergrundauftrag ... 524
25.6 Abbrechen und Löschen von Aufträgen ... 524
25.7 Analyse von Fehlermeldungen ... 524
25.8 Fernausführung von Hintergrundaufträgen ... 525
25.9 Praxislösung: Einen Job auf mehreren Computern starten ... 525

26 Geplante Aufgaben und zeitgesteuerte Jobs ... 527
26.1 Geplante Aufgaben (Scheduled Tasks) ... 527
26.2 Zeitgesteuerte Jobs ... 531

27 PowerShell-Workflows ... 537
27.1 Ein erstes Beispiel ... 537
27.2 Unterschiede zu einer Function bzw. einem Skript ... 542
27.3 Einschränkungen bei Workflows ... 542
27.4 Workflows in der Praxis ... 544
27.5 Workflows in Visual Studio erstellen ... 551

28 Ereignissystem ... 569
28.1 WMI-Ereignisse ... 569
28.2 WMI-Ereignisabfragen ... 569
28.3 WMI-Ereignisse seit PowerShell 1.0 ... 571
28.4 Registrieren von WMI-Ereignisquellen seit PowerShell 2.0 ... 572
28.5 Auslesen der Ereignisliste ... 573
28.6 Reagieren auf Ereignisse ... 575
28.7 WMI-Ereignisse ab PowerShell-Version 3.0 ... 577
28.8 Registrieren von .NET-Ereignissen ... 577
28.9 Erzeugen von Ereignissen ... 578

29 Datenbereiche und Datendateien ... 581
29.1 Datenbereiche ... 581
29.2 Datendateien ... 583
29.3 Mehrsprachigkeit/Lokalisierung ... 584

30 Desired State Configuration (DSC) ... 587
30.1 Grundprinzipien ... 588
30.2 DSC für Linux ... 588
30.3 Ressourcen ... 589
30.4 Verfügbare DSC-Ressourcen ... 589
30.5 Eigenschaften einer Ressource ... 592
30.6 Aufbau eines DSC-Dokuments ... 592
30.7 Commandlets für die Arbeit mit DSC ... 593
30.8 Ein erstes DSC-Beispiel ... 593
30.9 Kompilieren und Anwendung eines DSC-Dokuments ... 594
30.10 Variablen in DSC-Dateien ... 596
30.11 Parameter für DSC-Dateien ... 597
30.12 Konfigurationsdaten ... 598
30.13 Entfernen einer DSC-Konfiguration ... 601
30.14 DSC Pull Server ... 604
30.15 DSC-Praxislösung 1: IIS installieren ... 611
30.16 DSC-Praxislösung 2: Software installieren ... 613
30.17 DSC-Praxislösung 3: Software deinstallieren ... 615
30.18 Realisierung einer DSC-Ressource ... 616
30.19 Weitere Möglichkeiten ... 616

31 PowerShell-Snap-Ins ... 617
31.1 Einbinden von Snap-Ins ... 617
31.2 Liste der Commandlets ... 621

32 PowerShell-Module ... 623
32.1 Überblick über die Commandlets ... 623
32.2 Modularchitektur ... 624
32.3 Aufbau eines Moduls ... 625
32.4 Module aus dem Netz herunterladen und installieren mit PowerShellGet ... 626
32.5 Module manuell installieren ... 632
32.6 Doppeldeutige Namen ... 633
32.7 Auflisten der verfügbaren Module ... 634
32.8 Importieren von Modulen ... 635
32.9 Entfernen von Modulen ... 638

33 Ausgewählte PowerShell-Erweiterungen ... 639
33.1 PowerShell-Module in Windows 8.0 und Windows Server 2012 ... 640
33.2 PowerShell-Module in Windows 8.1 und Windows Server 2012 R2 ... 642
33.3 PowerShell-Module in Windows 10 und Windows Server 2019 ... 644
33.4 PowerShell Community Extensions (PSCX) ... 648
33.5 PowerShellPack ... 652
33.6 *www.IT-Visions.de*: PowerShell Extensions ... 654
33.7 Quest Management Shell for Active Directory ... 654
33.8 Microsoft Exchange Server ... 656
33.9 System Center Virtual Machine Manager ... 657
33.10 PowerShell Management Library for Hyper-V (pshyperv) ... 657
33.11 Powershell Outlook Account Manager ... 658
33.12 PowerShell Configurator (PSConfig) ... 659
33.13 Weitere Erweiterungen ... 660

34 Delegierte Administration/Just Enough Administration (JEA) ... 661
34.1 JEA-Konzept ... 661
34.2 PowerShell-Sitzungskonfiguration erstellen ... 661
34.3 Sitzungskonfiguration nutzen ... 664
34.4 Delegierte Administration per Webseite ... 666

35 Unit Tests mit Pester ... 667
35.1 Einführung in das Konzept des Unit Testing ... 667
35.2 Pester installieren ... 668
35.3 Befehle in Pester ... 668
35.4 Testen einer PowerShell-Funktion ... 669
35.5 Testgenerierung ... 670
35.6 Tests starten ... 671

35.7 Prüf-Operationen ... 672
35.8 Mock-Objekte ... 672
35.9 Test von Dateisystemoperationen ... 673

36 Tipps und Tricks zur PowerShell ... 677
36.1 Alle Anzeigen löschen ... 677
36.2 Befehlsgeschichte ... 677
36.3 System- und Hostinformationen ... 678
36.4 Anpassen der Eingabeaufforderung (Prompt) ... 679
36.5 PowerShell-Befehle aus anderen Anwendungen heraus starten ... 680
36.6 ISE erweitern ... 681
36.7 PowerShell für Gruppenrichtlinienskripte ... 682
36.8 Einblicke in die Interna der Pipeline-Verarbeitung ... 685

Teil C: PowerShell im Praxiseinsatz ... 687

37 Dateisystem ... 689
37.1 Laufwerke ... 690
37.2 Ordnerinhalte ... 695
37.3 Dateieigenschaften verändern ... 699
37.4 Eigenschaften ausführbarer Dateien ... 700
37.5 Kurznamen ... 702
37.6 Lange Pfade ... 702
37.7 Dateisystemoperationen ... 703
37.8 Praxislösungen: Dateien umorganisieren ... 703
37.9 Praxislösung: Zufällige Dateisystemstruktur erzeugen ... 705
37.10 Praxislösung: Leere Ordner löschen ... 706
37.11 Einsatz von Robocopy ... 707
37.12 Dateisystemkataloge ... 710
37.13 Papierkorb leeren ... 711
37.14 Dateieigenschaften lesen ... 711
37.15 Praxislösung: Fotos nach Aufnahmedatum sortieren ... 712
37.16 Datei-Hash ... 713
37.17 Finden von Duplikaten ... 714
37.18 Verknüpfungen im Dateisystem ... 716
37.19 Komprimierung ... 721
37.20 Dateisystemfreigaben ... 724
37.21 Überwachung des Dateisystems ... 735
37.22 Dateiversionsverlauf ... 736
37.23 Windows Explorer öffnen ... 737
37.24 Windows Server Backup ... 737

38 Festplattenverschlüsselung mit BitLocker ... 739
38.1 Übersicht über das BitLocker-Modul ... 740
38.2 Verschlüsseln eines Laufwerks ... 741

39 Dokumente ... 743
39.1 Textdateien ... 743
39.2 CSV-Dateien ... 745
39.3 Analysieren von Textdateien ... 748
39.4 INI-Dateien ... 751
39.5 XML-Dateien ... 751
39.6 HTML- und Markdown-Dateien ... 761
39.7 JSON-Dateien ... 764
39.8 Binärdateien ... 774
39.9 Grafikdateien ... 775

40 Microsoft Office ... 777
40.1 Allgemeine Informationen zur Office-Automatisierung per PowerShell ... 777
40.2 Praxislösung: Terminserien aus Textdateien anlegen in Outlook ... 778
40.3 Praxislösung: Outlook-Termine anhand von Suchkriterien löschen ... 780
40.4 Praxislösung: Grafiken aus einem Word-Dokument (DOCX) extrahieren ... 781

41 Datenbanken ... 785
41.1 ADO.NET-Grundlagen ... 785
41.2 Beispieldatenbank ... 791
41.3 Datenzugriff mit den Bordmitteln der PowerShell ... 792
41.4 Hilfsfroutinen für den Datenbankzugriff (DBUtil.ps1) ... 804
41.5 Datenzugriff mit den PowerShell-Erweiterungen ... 807
41.6 Datenbankzugriff mit SQLPS ... 811
41.7 Datenbankzugriff mit SQLPSX ... 811

42 Microsoft-SQL-Server-Administration ... 813
42.1 PowerShell-Integration im SQL Server Management Studio ... 814
42.2 SQL-Server-Laufwerk „SQLSERVER:“ ... 815
42.3 Die SQLPS-Commandlets ... 818
42.4 Die SQL Server Management Objects (SMO) ... 820
42.5 SQLPSX ... 823
42.6 Microsoft-SQL-Server-Administration mit der PowerShell in der Praxis ... 831

43 ODBC-Datenquellen ... 837
43.1 ODBC-Treiber und -Datenquellen auflisten ... 838
43.2 Anlegen einer ODBC-Datenquelle ... 839
43.3 Zugriff auf eine ODBC-Datenquelle ... 840

44 Registrierungsdatenbank (Registry) ... 843
44.1 Schlüssel auslesen ... 843
44.2 Schlüssel anlegen und löschen ... 844
44.3 Laufwerke definieren ... 844
44.4 Werte anlegen und löschen ... 845
44.5 Werte auslesen ... 846
44.6 Praxislösung: Windows-Explorer-Einstellungen ... 846
44.7 Praxislösung: Massenanlegen von Registry-Schlüsseln ... 847

45 Computer- und Betriebssystemverwaltung ... 849
45.1 Computerinformationen ... 849
45.2 Versionsnummer des Betriebssystems ... 851
45.3 Zeitdauer seit dem letzten Start des Betriebssystems ... 851
45.4 BIOS- und Startinformationen ... 852
45.5 Windows-Produktaktivierung ... 853
45.6 Umgebungsvariablen ... 853
45.7 Schriftarten ... 856
45.8 Computername und Domäne ... 857
45.9 Herunterfahren und Neustarten ... 857
45.10 Windows Updates installieren ... 858
45.11 Wiederherstellungspunkte verwalten ... 862

46 Windows Defender ... 863

47 Hardwareverwaltung ... 865
47.1 Hardwarebausteine ... 865
47.2 Plug-and-Play-Geräte ... 867
47.3 Druckerverwaltung (ältere Betriebssysteme) ... 867
47.4 Druckerverwaltung (seit Windows 8 und Windows Server 2012) ... 869

48 Softwareverwaltung ... 871
48.1 Softwareinventarisierung ... 871
48.2 Installation von Anwendungen ... 875
48.3 Deinstallation von Anwendungen ... 875
48.4 Praxislösung: Installationstest ... 876
48.5 Praxislösung: Installierte .NET Core SDKs aufräumen ... 877
48.6 Windows 10 Apps verwalten ... 883
48.7 Installationen mit PowerShell Package Management („OneGet“) ... 885
48.8 Versionsnummer ermitteln ... 888
48.9 Servermanager ... 890
48.10 Windows-Features installieren auf Windows-Clientbetriebssystemen ... 900

48.11 Praxislösung: IIS-Installation ... 903
48.12 Softwareeinschränkungen mit dem PowerShell-Modul „AppLocker“ ... 905

49 Prozessverwaltung ... 911
49.1 Prozesse auflisten ... 911
49.2 Prozesse starten ... 912
49.3 Prozesse mit vollen Administratorrechten starten ... 913
49.4 Prozesse unter einem anderen Benutzerkonto starten ... 914
49.5 Prozesse beenden ... 915
49.6 Warten auf das Beenden einer Anwendung ... 916

50 Windows-Systemdienste ... 917
50.1 Dienste auflisten ... 917
50.2 Dienstzustand ändern ... 920
50.3 Diensteigenschaften ändern ... 920
50.4 Dienste hinzufügen ... 921
50.5 Dienste entfernen ... 922

51 Netzwerk ... 923
51.1 Netzwerkkonfiguration ... 923
51.2 DNS-Client-Konfiguration ... 928
51.3 DNS-Namensauflösung ... 932
51.4 Erreichbarkeit prüfen (Ping) ... 933
51.5 Windows Firewall ... 934
51.6 Remote Desktop (RDP) einrichten ... 941
51.7 E-Mails senden (SMTP) ... 942
51.8 Auseinandernehmen von E-Mail-Adressen ... 943
51.9 Abruf von Daten von einem HTTP-Server ... 943
51.10 Praxislösung: Linkprüfer für eine Website ... 949
51.11 Aufrufe von SOAP-Webdiensten ... 953
51.12 Aufruf von REST-Diensten ... 955
51.13 Aufrufe von OData-Diensten ... 957
51.14 Hintergrunddatentransfer mit BITS ... 958

52 Ereignisprotokolle (Event Log) ... 963
52.1 Protokolleinträge auslesen ... 963
52.2 Ereignisprotokolle erzeugen ... 965
52.3 Protokolleinträge erzeugen ... 965
52.4 Protokollgröße festlegen ... 965
52.5 Protokolleinträge löschen ... 965

53 Leistungsdaten (Performance Counter) ... 967
53.1 Zugriff auf Leistungsindikatoren über WMI ... 967
53.2 Get-Counter ... 968

54 Sicherheitseinstellungen ... 971
54.1 Aktueller Benutzer ... 971
54.2 Grundlagen ... 972
54.3 Zugriffsrechtelisten auslesen ... 977
54.4 Einzelne Rechteeinträge auslesen ... 978
54.5 Besitzer auslesen ... 980
54.6 Benutzer und SID ... 980
54.7 Hinzufügen eines Rechteeintrags zu einer Zugriffsrechteliste ... 983
54.8 Entfernen eines Rechteeintrags aus einer Zugriffsrechteliste ... 985
54.9 Zugriffsrechteliste übertragen ... 987
54.10 Zugriffsrechteliste über SDDL setzen ... 987
54.11 Zertifikate verwalten ... 988

55 Optimierungen und Problemlösungen ... 991
55.1 PowerShell-Modul „TroubleshootingPack“ ... 991
55.2 PowerShell-Modul „Best Practices“ ... 995

56 Active Directory ... 997
56.1 Benutzer- und Gruppenverwaltung mit WMI ... 998
56.2 Einführung in System.Directory Services ... 999
56.3 Basiseigenschaften ... 1010
56.4 Benutzer- und Gruppenverwaltung im Active Directory ... 1012
56.5 Verwaltung der Organisationseinheiten ... 1020
56.6 Suche im Active Directory ... 1021
56.7 Navigation im Active Directory mit den PowerShell Extensions ... 1028
56.8 Verwendung der Active-Directory-Erweiterungen von www.IT-Visions.de ... *1029*
56.9 PowerShell-Modul „Active Directory“ (ADPowerShell) ... 1031
56.10 PowerShell-Modul „ADDSDeployment“ ... 1060
56.11 Informationen über die Active Directory-Struktur ... 1063

57 Gruppenrichtlinien ... 1067
57.1 Verwaltung der Gruppenrichtlinien ... 1068
57.2 Verknüpfung der Gruppenrichtlinien ... 1069
57.3 Gruppenrichtlinienberichte ... 1071
57.4 Gruppenrichtlinienvererbung ... 1073
57.5 Weitere Möglichkeiten ... 1074

58 Lokale Benutzer und Gruppen ... 1075
58.1 Modul „Microsoft.PowerShell.LocalAccounts“ ... 1075
58.2 Lokale Benutzerverwaltung in älteren PowerShell-Versionen ... 1077

59 Microsoft Exchange Server ... 1079
59.1 Daten abrufen ... 1079
59.2 Postfächer verwalten ... 1080
59.3 Öffentliche Ordner verwalten ... 1081

60 Internet Information Services (IIS) ... 1083
60.1 Überblick ... 1083
60.2 Navigationsprovider ... 1085
60.3 Anlegen von Websites ... 1087
60.4 Praxislösung: Massenanlegen von Websites ... 1088
60.5 Ändern von Website-Eigenschaften ... 1091
60.6 Anwendungspool anlegen ... 1091
60.7 Virtuelle Verzeichnisse und IIS-Anwendungen ... 1092
60.8 Website-Zustand ändern ... 1093
60.9 Anwendungspools starten und stoppen ... 1093
60.10 Löschen von Websites ... 1094

61 Virtuelle Systeme mit Hyper-V ... 1095
61.1 Das Hyper-V-Modul von Microsoft ... 1096
61.2 Die ersten Schritte mit dem Hyper-V-Modul ... 1098
61.3 Virtuelle Maschinen anlegen ... 1102
61.4 Umgang mit virtuellen Festplatten ... 1108
61.5 Konfiguration virtueller Maschinen ... 1111
61.6 Dateien kopieren in virtuelle Systeme ... 1115
61.7 PowerShell Management Library for Hyper-V (für ältere Betriebssysteme) ... 1116

62 Windows Nano Server ... 1119
62.1 Das Konzept von Nano Server ... 1119
62.2 Einschränkungen von Nano Server ... 1121
62.3 Varianten des Nano Servers ... 1123
62.4 Installation eines Nano Servers ... 1123
62.5 Docker-Image ... 1124
62.6 Fernverwaltung mit PowerShell ... 1125
62.7 Windows Update auf einem Nano Server ... 1127
62.8 Nachträgliche Paketinstallation ... 1127
62.9 Abgespeckter IIS unter Nano Server ... 1129
62.10 Nano-Serververwaltung aus der Cloud heraus ... 1130

63 Docker-Container ... 1131
63.1 Docker-Varianten für Windows ... 1132
63.2 Docker-Installation auf Windows 10 ... 1133
63.3 Docker-Installation auf Windows Server ... 1135
63.4 Installation von „Docker for Windows“ ... 1136
63.5 Docker-Registries ... 1138
63.6 Docker-Images laden ... 1138
63.7 Container starten ... 1138
63.8 Container-Identifikation ... 1140
63.9 Container mit Visual Studio ... 1141
63.10 Befehle in einem Container ausführen ... 1143
63.11 Ressourcenbeschränkungen für Container ... 1146
63.12 Dateien zwischen Container und Host kopieren ... 1146
63.13 Dockerfile ... 1146
63.14 Docker-Netzwerke ... 1147
63.15 Container anlegen, ohne sie zu starten ... 1148
63.16 Container starten und stoppen ... 1148
63.17 Container beenden und löschen ... 1148
63.18 Images löschen ... 1149
63.19 Images aus Containern erstellen ... 1149
63.20 .NET Core-Container ... 1150
63.21 Images verbreiten ... 1152
63.22 Azure Container Service (ACS) ... 1154

64 Microsoft Azure ... 1155
64.1 Azure Konzepte ... 1155
64.2 Kommandozeilenwerkzeuge für die Azure-Verwaltung ... 1156
64.3 Benutzeranmeldung und Informationsabfrage ... 1160
64.4 Azure Ressourcen-Gruppen ... 1161
64.5 Azure Web Apps ... 1161
64.6 Azure SQL Server ... 1163
64.7 Azure Kubernetes Services (AKS) ... 1164
64.8 Azure DevOps ... 1188

65 Grafische Benutzeroberflächen (GUI) ... 1203
65.1 Einfache Nachfragedialoge ... 1203
65.2 Einfache Eingabe mit Inputbox ... 1205
65.3 Komplexere Eingabemasken ... 1205
65.4 Universelle Objektdarstellung ... 1207
65.5 WPF PowerShell Kit (WPK) ... 1209
65.6 Direkte Verwendung von WPF ... 1217

Teil D: Profiwissen – Erweitern der PowerShell 1219

66 Entwicklung von Commandlets in der PowerShell-Skriptsprache 1221

66.1 Aufbau eines skriptbasierten Commandlets 1221
66.2 Verwendung per Dot Sourcing 1223
66.3 Parameterfestlegung 1224
66.4 Fortgeschrittene Funktion (Advanced Function) 1230
66.5 Mehrere Parameter und Parametersätze 1234
66.6 Unterstützung für Sicherheitsabfragen (-whatif und -confirm) 1236
66.7 Kaufmännisches Beispiel: Test-CustomerID 1237
66.8 Erweitern bestehender Commandlets durch Proxy-Commandlets 1241
66.9 Dokumentation 1246

67 Entwicklung eigener Commandlets mit C# 1249

67.1 Technische Voraussetzungen 1250
67.2 Grundkonzept der .NET-basierten Commandlets 1251
67.3 Schrittweise Erstellung eines minimalen Commandlets 1253
67.4 Erstellung eines Commandlets mit einem Rückgabeobjekt 1262
67.5 Erstellung eines Commandlets mit mehreren Rückgabeobjekten 1263
67.6 Erstellen eines Commandlets mit Parametern 1268
67.7 Verarbeiten von Pipeline-Eingaben 1270
67.8 Verkettung von Commandlets 1273
67.9 Fehlersuche in Commandlets 1277
67.10 Statusinformationen 1280
67.11 Unterstützung für Sicherheitsabfragen (-whatif und -confirm) 1285
67.12 Festlegung der Hilfeinformationen 1287
67.13 Erstellung von Commandlets für den Zugriff auf eine Geschäftsanwendung 1292
67.14 Konventionen für Commandlets 1293
67.15 Weitere Möglichkeiten 1295

68 PowerShell-Module erstellen 1297

68.1 Erstellen eines Skriptmoduls 1297
68.2 Praxislösung: Umwandlung einer Skriptdatei in ein Modul 1299
68.3 Erstellen eines Moduls mit Binärdateien 1299
68.4 Erstellen eines Moduls mit Manifest 1300
68.5 Erstellung eines Manifest-Moduls mit Visual Studio 1307

69 Hosting der PowerShell 1309

69.1 Voraussetzungen für das Hosting 1310
69.2 Hosting mit PSHost 1311
69.3 Vereinfachtes Hosting seit PowerShell 2.0 1314

Anhang A: Crashkurs Objektorientierung **1317**

Anhang B: Crashkurs .NET **1325**

B.1 Was ist das .NET Framework? 1327
B.2 Was ist .NET Core? 1328
B.3 Eigenschaften von .NET 1329
B.4 .NET-Klassen 1330
B.5 Namensgebung von .NET-Klassen (Namensräume) 1330
B.6 Namensräume und Softwarekomponenten 1332
B.7 Bestandteile einer .NET-Klasse 1333
B.8 Vererbung 1334
B.9 Schnittstellen 1334

Anhang C: Literatur **1335**

Anhang D: Weitere Informationen im Internet **1339**

Anhang E: Abkürzungsverzeichnis **1341**

Stichwortverzeichnis **1367**

Vorwort

Liebe Leserin, lieber Leser,

willkommen zur aktuellen Auflage meines PowerShell-Buchs! Es handelt sich hierbei um die vierte Auflage des Windows PowerShell 5-Buches und die achte 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 7.0 von Microsoft wie auch ergänzende Werkzeuge von Microsoft und Drittanbietern (z. B. PowerShell Community Extensions).

Das Buch ist aber auch dann für Sie geeignet, wenn Sie noch Windows PowerShell 2.0/3.0/4.0/5.0 oder PowerShell Core 6.0/6.1/6.2 einsetzen. Welche Funktionen neu hinzugekommen sind, wird jeweils erwähnt.

Wer bin ich?

Mein Name ist Holger Schwichtenberg, ich bin derzeit 47 Jahre alt und habe im Fachgebiet Wirtschaftsinformatik promoviert. Ich lebe (in Essen, im Herzen des Ruhrgebiets) davon, dass mein Team und ich im Rahmen unserer Firma *www.IT-Visions.de* anderen Unternehmen bei der Entwicklung von .NET-, Web- und PowerShell-Anwendungen beratend und schulend zur Seite stehen. Zudem entwickeln wir im Rahmen der 5Minds IT-Solutions GmbH & Co. KG *(www.5Minds.de)* Software im Auftrag von Kunden in zahlreichen Branchen.

Es ist mein Hobby und meine Nebentätigkeit, IT-Fachbücher zu schreiben. Dieses Buch ist, unter Mitzählung aller nennenswerten Neuauflagen, das 82. Buch, das ich allein oder mit Co-Autoren geschrieben habe. Meine weiteren Hobbys sind Mountain Biking, Fotografie und Reisen.

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

Wer sind Sie?

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

Sind Sie Systemadministrator in einem Windows-Netzwerk?	○ Ja	○ Nein
Laufen die für Sie relevanten Computer mit den von PowerShell unterstützten Betriebssystemen? (Windows 7/8/8.1/10, Windows Server 2008/2008 R2/2012/2012 R2/2016/2019, macOS, Linux)	○ Ja	○ Nein
Sie besitzen zumindest rudimentäre Grundkenntnisse im Bereich des (objektorientierten) Programmierens?	○ Ja	○ Nein
Wünschen Sie einen kompakten Überblick über die Architektur, Konzepte und Anwendungsfälle der PowerShell?	○ Ja	○ Nein
Sie können auf Schritt-für-Schritt-Anleitungen verzichten?	○ Ja	○ Nein
Sie können auf formale Syntaxbeschreibungen verzichten und lernen lieber an aussagekräftigen Beispielen?	○ Ja	○ Nein
Sie erwarten nicht, dass in diesem Buch **alle** Möglichkeiten der PowerShell detailliert beschrieben werden?	○ Ja	○ 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 1200 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?	○ Ja	○ Nein

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

Was ist neu in diesem Buch?

Gegenüber der vorherigen, siebten Auflage zur PowerShell 5.1/PowerShell Core 6.1 (die der Verlag leider nur als E-Book veröffentlicht hat) wurde das Buch um die neuen Commandlets, Funktionen und Operationen in PowerShell Core 6.2 und 6.3 sowie PowerShell 7.0 erwei-

tert. Zu Dateisystemoperationen gibt es weitere Beispiele aus meiner Praxis. Bei den Werkzeugen ist das neue Windows Terminal sowie das Unit Testing mit Pester hinzugekommen.

Im Bereich der Anwendungsgebiete ist neu hinzugekommen:

- Verarbeitung von Markdown
- JavaScript Object Notation (JSON)
- Automatisierung von Microsoft Office
- Verwaltung der Universal Windows Platform (UWP) Apps
- Verwaltung von Microsoft Azure, insbesondere Azure Kubernetes Service (AKS) und Azure DevOps

Zudem wurden die bestehenden Inhalte des Buchs an vielen Stellen erweitert und didaktisch optimiert.

Sind in diesem Buch alle Features der PowerShell beschrieben?

Die PowerShell umfasst mittlerweile 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.

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

Dies hat natürlich auch Auswirkungen auf ein etabliertes Fachbuch wie das vorliegende. Leider kann man ein gedrucktes Buch nicht so schnell ändern wie Software. Verlage definieren erhebliche Mindestauflagen, die abverkauft werden müssen, bevor neu gedruckt werden darf. Das E-Book ist keine echte Alternative. Die Verkaufszahlen zeigen, dass nur eine kleine Gruppe 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 oder Web-Links nicht mehr funktionieren. 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 Verlagsdruckerei an den Buchhandel rund vier Monate liegen.

Welche PowerShell-Versionen werden besprochen?

Das Buch bespricht sowohl die Windows PowerShell 5.1 als auch die PowerShell 7.0. Häufig wird auch erwähnt, dass Änderungen zwischen diesen beide Versionen schon in PowerShell Core 6.x eingeführt werden.

Bei der Windows PowerShell 5.1 wird die RTM-Version besprochen, die Microsoft in Windows 10 1909 bzw. Windows Server 1909 mitliefert.

Bei PowerShell 7.0 wird die Release Candidate 1 (RC1)-Version besprochen. Dass hier nicht die RTM-Version besprochen wird, liegt daran, dass der Verlag als Manuskriptabgabetermin den 6. Januar 2020 festgelegt hat. Ich musste die Texte daher schon im Dezember 2019 mit der RC1-Version schreiben.

Da es zu diesem Zeitpunkt noch eine bruchstückhafte Dokumentation der PowerShell 7.0 gab, kann es sein, dass nicht alle neuen Features in diesem Buch erwähnt sind. Ich habe in den letzten Wochen so viel wie möglich die PowerShell 7.0 eingesetzt, um möglichst viele Neuerungen in der Praxis zu bemerken. Ich werde Neuerungen in PowerShell 7.0 daher auch in meinem Weblog *(www.dotnet-doktor.de)* verfolgen.

Warum behandelt das Buch auch noch Version 5.1 und nicht nur Version 7.0?

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

Die PowerShell 7.0 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 7.0 der Windows PowerShell 5.1 funktionell immer noch nicht ganz ebenbürtig ist. Einige Befehle sind weiterhin nur in der Windows PowerShell verfügbar.

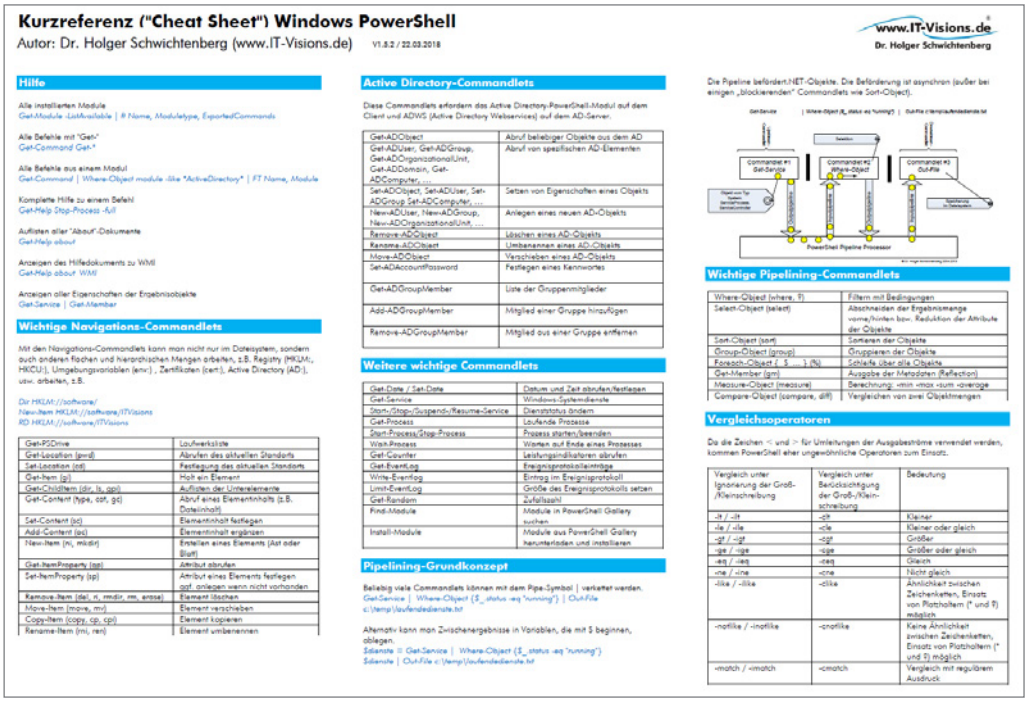
Daher wird die Windows PowerShell 5.1 auch weiterhin eine große Bedeutung haben und daher in diesem Buch auch weiterhin behandelt.

Woher bekommt man die Beispiele aus diesem Buch?

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

In diesem Portal können Sie

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



Kurzreferenz ("Cheat Sheet") Windows PowerShell

Autor: Dr. Holger Schwichtenberg (www.IT-Visions.de) V1.5.2 / 22.03.2018

www.IT-Visions.de
Dr. Holger Schwichtenberg

Hilfe

Alle installierten Module
Get-Module -listAvailable | # Name, Moduletype, ExportedCommands

Alle Befehle mit "Get-"
*Get-Command Get-**

Alle Befehle aus einem Modul
Get-Command | Where-Object module -like "ActiveDirectory" | FT Name, Module

Komplette Hilfe zu einem Befehl
Get-Help Stop-Process -full

Auflisten aller "About"-Dokumente
Get-Help about

Anzeigen des Hilfedokuments zu WMI
Get-Help about WMI

Anzeigen aller Eigenschaften der Ergebnisobjekte
Get-Service | Get-Member

Wichtige Navigations-Commandlets

Mit den Navigations-Commandlets kann man nicht nur im Dateisystem, sondern auch anderen flachen und hierarchischen Mengen arbeiten, z.B. Registry (HKLM:, HKCU:), Umgebungsvariablen (env:) , Zertifikaten (cert:), Active Directory (AD:), usw. arbeiten, z.B.

Dir HKLM:// software/
New-Item HKLM://software/ITVisions
RD HKLM://software/ITVisions

Get-PSDrive	Laufwerksliste
Get-Location (pwd)	Abrufen des aktuellen Standorts
Set-Location (cd)	Festlegung des aktuellen Standorts
Get-Item (gi)	Holt ein Element
Get-ChildItem (dir, ls, gci)	Auflisten der Unterelemente
Get-Content (type, cat, gc)	Abruf eines Elementinhalts (z.B. Dateiinhalt)
Set-Content (sc)	Elementinhalt festlegen
Add-Content (ac)	Elementinhalt ergänzen
New-Item (ni, mkdir)	Erstellen eines Elements (Ast oder Blatt)
Set-ItemProperty (sp)	Attribut abrufen
Set-ItemProperty (sp)	Attribut eines Elements festlegen ggf. anlegen wenn nicht vorhanden
Remove-Item (del, ri, rmdir, rm, erase)	Element löschen
Move-Item (move, mv)	Element verschieben
Copy-Item (copy, cp, cpi)	Element kopieren
Rename-Item (rni, ren)	Element umbenennen

Active Directory-Commandlets

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

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

Weitere wichtige Commandlets

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

Pipelining-Grundkonzept

Beliebig viele Commandlets können mit dem Pipe-Symbol | verkettet werden.
Get-Service | Where-Object {$_.status -eq "running"} | Out-File c:\temp\laufendedienste.txt

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

Die Pipeline befördert .NET-Objekte. Die Beförderung ist asynchron (außer bei einigen „blockierenden" Commandlets wie Sort-Object).

Wichtige Pipelining-Commandlets

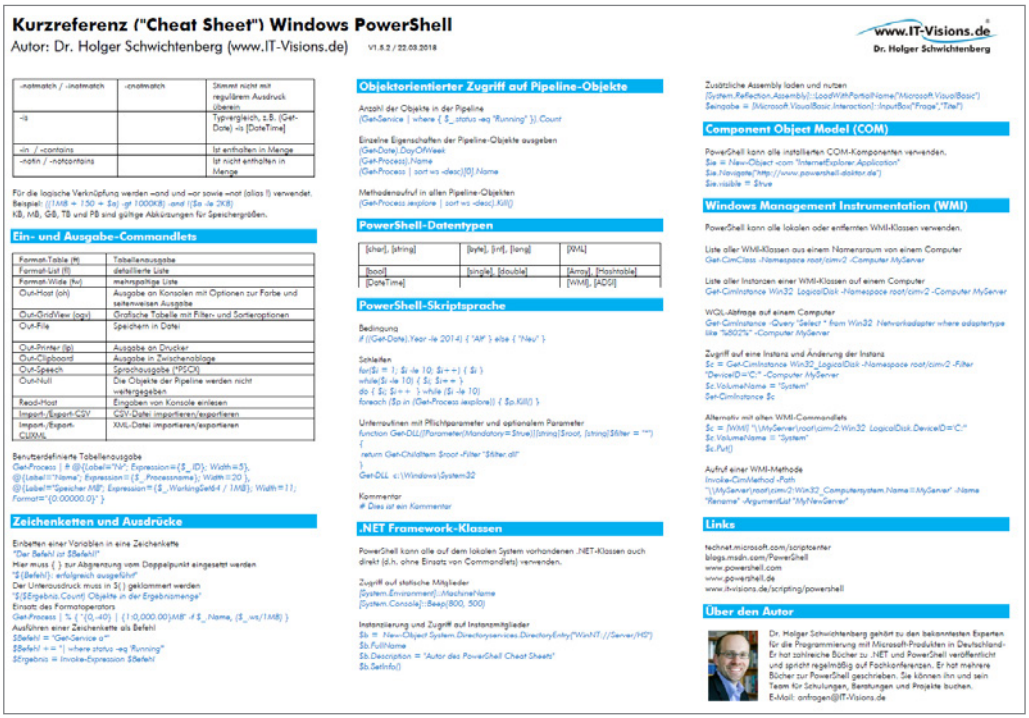
Where-Object (where, ?)	Filtern mit Bedingungen
Select-Object (select)	Abschneiden der Ergebnismenge vorne/hinten bzw. Reduktion der Attribute der Objekte
Sort-Object (sort)	Sortieren der Objekte
Group-Object (group)	Gruppieren der Objekte
Foreach-Object { $_ } (%)	Schleife über alle Objekte
Get-Member (gm)	Ausgabe der Metadaten (Reflection)
Measure-Object (measure)	Berechnung: -min -max -sum -average
Compare-Object (compare, diff)	Vergleichen von zwei Objektmengen

Vergleichsoperatoren

Da die Zeichen < und > für Umleitungen der Ausgabeströme verwendet werden, kommen PowerShell eher ungewöhnliche Operatoren zum Einsatz.

Vergleich unter Ignorierung der Groß-/Kleinschreibung	Vergleich unter Berücksichtigung der Groß-/Kleinschreibung	Bedeutung
-lt / -llt	-clt	Kleiner
-le / -lle	-cle	Kleiner oder gleich
-gt / -lgt	-cgt	Größer
-ge / -lge	-cge	Größer oder gleich
-eq / -leq	-ceq	Gleich
-ne / -lne	-cne	Nicht gleich
-like / -llike	-clike	Ähnlichkeit zwischen Zeichenketten, Einsatz von Platzhaltern (* und ?) möglich
-notlike / -lnotlike	-cnotlike	Keine Ähnlichkeit zwischen Zeichenketten, Einsatz von Platzhaltern (* und ?) möglich
-match / -lmatch	-cmatch	Vergleich mit regulärem Ausdruck

Bild 1 Vorderseite der PowerShell-Kurzreferenz



Kurzreferenz ("Cheat Sheet") Windows PowerShell

Autor: Dr. Holger Schwichtenberg (www.IT-Visions.de) V1.5.2 / 22.03.2018

www.IT-Visions.de
Dr. Holger Schwichtenberg

-notmatch / -lnotmatch	-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

Für die logische Verknüpfung werden –and und –or sowie –not (alias !) verwendet.
Beispiel: *((1MB -le 150 -or $a) -gt 1000KB) -and !($a -le 2KB)*
KB, MB, GB, TB und PB sind gültige Abkürzungen für Speichergrößen.

Ein- und Ausgabe-Commandlets

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

Benutzerdefinierte Tabellenausgabe
Get-Process | ft @{Label="Id"; Expression={$_.ID}; Width=5},
@{Label="Name"; Expression={$_.ProcessName}; Width=20 },
@{Label="Speicher MB"; Expression={ ($_.WorkingSet64 / 1MB); Width=11; Format="{0:00000.0}" }

Zeichenketten und Ausdrücke

Einbetten einer Variablen in eine Zeichenkette
"Der Befehl ist $Befehl!"
Hier muss { } zur Abgrenzung vom Doppelpunkt eingesetzt werden
"${Befehl}: erfolgreich ausgeführt"
Der Unterausdruck muss in $() geklammert werden
"$($Ergebnis.Count) Objekte in der Ergebnismenge"
Einsatz des Formatoperators
Get-Process | % { "{0,-40} | {1:0,000.00}MB" -f $_.Name, ($_.ws/1MB) }
Ausführen einer Zeichenkette als Befehl
$Befehl = "Get-Service a"*
$Befehl += " | where status -eq 'Running'"
$Ergebnis = Invoke-Expression $Befehl

Objektorientierter Zugriff auf Pipeline-Objekte

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

Einzelne Eigenschaften der Pipeline-Objekte ausgeben
(Get-Date).DayOfWeek
(Get-Process).Name
(Get-Process | sort ws -desc)[0].Name

Methodenaufruf in allen Pipeline-Objekten
(Get-Process explore | sort ws -desc).Kill()

PowerShell-Datentypen

[char], [string]	[byte], [int], [long]	[XML]	
[bool]	[single], [double]	[Array] [Hashtable]	
[DateTime]		[WMI], [ADSI]	

PowerShell-Skriptsprache

Bedingung
if ((Get-Date).Year -le 2014) { "Alt" } else { "Neu" }

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

Unterroutinen mit Pflichtparameter und optionalem Parameter
function Get-DLL([Parameter(Mandatory=$true)][string]$root, [string]$filter = "")*
{
return Get-ChildItem $root -Filter "$filter.dll"
}
Get-DLL -c:\Windows\System32

Kommentar
 # Dies ist ein Kommentar

.NET Framework-Klassen

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

Zugriff auf statische Mitglieder
[System.Environment]::MachineName
[System.Console]::Beep(800, 500)

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

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

Component Object Model (COM)

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

Windows Management Instrumentation (WMI)

PowerShell kann alle lokalen oder entfernten WMI-Klassen verwenden.

Liste aller WMI-Klassen aus einem Namensraum von einem Computer
Get-CimClass -Namespace root/cimv2 -Computer MyServer

Liste aller Instanzen einer WMI-Klassen auf einem Computer
Get-CimInstance Win32_LogicalDisk -Namespace root/cimv2 -Computer MyServer

WQL-Abfrage auf einem Computer
*Get-CimInstance -Query "Select * from Win32_NetworkAdapter where adaptertype like '%802%' -Computer MyServer*

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

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

Aufruf einer WMI-Methode
Invoke-CimMethod -Path "\\MyServer\root\cimv2:Win32_Computersystem.Name='MyServer'" -Name "Rename" -ArgumentList "MyNewServer"

Links

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

Über den Autor

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

Bild 2 Rückseite der PowerShell-Kurzreferenz

Alle registrierten Leser erhalten auch meinen Newsletter (2 bis 4 × im Jahr) mit aktuellen Produktinformationen, Einladungen zu kostenlosen Community-Veranstaltungen sowie Vergünstigungen bei unseren öffentlichen Seminaren zu .NET und zur PowerShell.

Wie sind die Programmcodebeispiele organisiert?

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

Buchteilname\Kapitelname\Dateiname

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

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

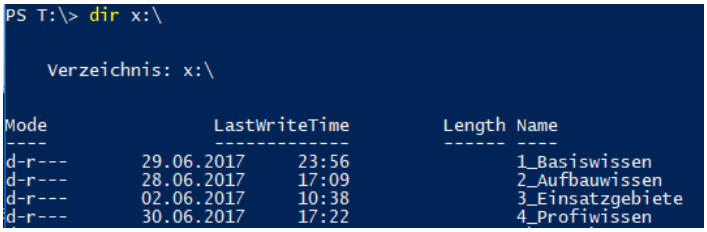


Bild 3 Verzeichnisstruktur der Beispielsammlung mit vier Hauptordnern entsprechend den vier Buchteilen

Bild 4 Inhalt eines der Hauptordner aus der vorherigen Bildschirmabbildung, d. h. eines Buchteils

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

Wie wurde die Qualität gesichert?

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

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

- Eine abweichende Systemkonfiguration (in der heutigen komplexen Welt der vielen Varianten und Versionen von Betriebssystemen und Anwendungen nicht unwahrscheinlich). Es ist einem Fachbuchautor nicht möglich, alle Konfigurationen durchzutesten.
- Änderungen, die sich seit der Erstellung der Beispiele ergeben haben (mittlerweile gibt es sehr regelmäßig umfangreiche Breaking Changes in den Microsoft-Produkten, insbesondere beim Versionsnummernwechsel an der ersten Stelle, d. h. Windows PowerShell 5.1 und PowerShell 6.0 sowie PowerShell 6.2 und PowerShell 7.0).
- Schließlich sind auch menschliche Fehler des Autors möglich. Bitte bedenken Sie, dass das Fachbuchschreiben nur ein Hobby ist. Es gibt nur sehr wenige Menschen in Deutschland, die hauptberuflich als Fachbuchautor arbeiten und so professionell Programmcodebeispiele erstellen und testen können wie kommerziellen (bezahlten) Programmcode.

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

Wo kann man Verbesserungsvorschläge melden?

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

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

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

Wann wird die nächste Auflage erscheinen?

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

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

Wo kann man sich schulen oder beraten lassen?

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

Wem ist zu danken?

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

- meinem Kollegen Peter Monadjemi, der rund 100 Seiten mit Beispielen zu der 3. Auflage dieses Buchs beigetragen hat und dessen Inhalte zum Teil noch im Buch enthalten sind (Themen: Workflows, Bitlocker, ODBC, Hyper-V, DNS-Client, Firewall und Microsoft SQL Server-Administration),
- meinem Kollegen André Krämer, der die PowerShell 7.0 auf macOS getestet hat, da ich selbst kein macOS-Gerät besitze,
- Frau Sylvia Hasselbach, die mich schon seit 20 Jahren als Lektorin begleitet und die dieses Buchprojekt beim Carl Hanser Verlag koordiniert und vermarktet,
- Frau Sandra Gottmann (6. Auflage) und Herrn Matthias Bloch (7. Auflage), die meine Tippfehler gefunden und sprachliche Ungenauigkeiten eliminiert haben,
- meiner Frau und meinen Kindern dafür, dass sie mir das Umfeld geben, um neben meinem Hauptberuf an Büchern wie diesem zu arbeiten.

Zum Schluss dieses Vorworts …

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

Dr. Holger Schwichtenberg

Essen, im Januar 2020

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 nächstes Kapitel).

5.3 .NET-Objekte in der Pipeline

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

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

In einer Pipeline wie

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

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

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

Im obigen Beispiel legt `Get-Process` ein .NET-Objekt der Klasse `System.Diagnostics.Process` für jeden laufenden Prozess in die Pipeline. `System.Diagnostics.Process` ist eine Klasse aus der .NET-Klassenbibliothek. Commandlets können aber jedes beliebige .NET-Objekt in die Pipeline legen, also auch einfache Zahlen oder Zeichenketten, da es in .NET keine Unterscheidung zwischen elementaren Datentypen und Klassen gibt. Eine Zeichenkette in die Pipeline zu legen, wird aber in der PowerShell die Ausnahme bleiben, denn der typisierte Zugriff auf Objekte ist wesentlich robuster gegenüber möglichen Änderungen als die Zeichenkettenauswertung mit regulären Ausdrücken.

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

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

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

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

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

5.4 Pipeline Processor

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

Wie Bild 5.1 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 PowerShell aber im Standard nur mit 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.

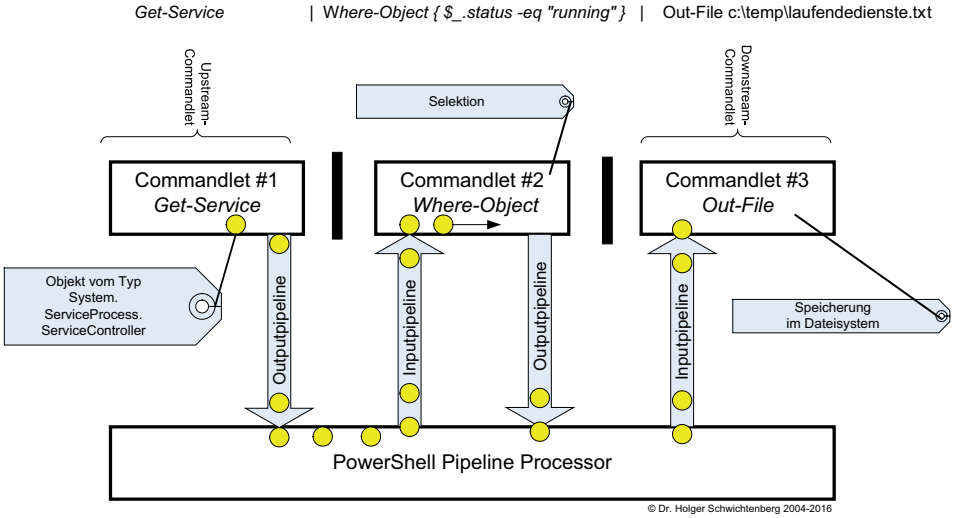


Bild 5.1 Der Pipeline Processor befördert die Objekte vom Downstream-Commandlet zum Upstream-Commandlet. Die Verarbeitung ist in der Regel asynchron.

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

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

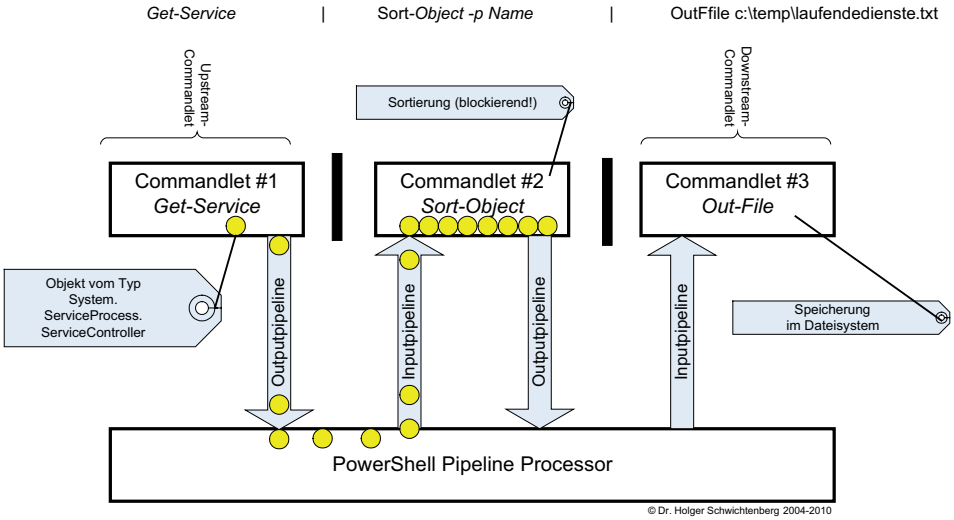


Bild 5.2 Sort-Object blockiert die direkte Weitergabe. Erst wenn alle Objekte angekommen sind, kann das Commandlet sortieren.

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

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

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

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

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

5.5 Pipelining von Parametern

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

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

```
-Include <string[]>
    Retrieves only the specified services. The value of this parameter qualifie
    s the Name parameter. Enter a name element or pattern, such as "s*". Wildca
    rds 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 retriev
    ed. Enter a variable that contains the objects, or type a command or expres
    sion that gets the objects. You can also pipe a service object to Get-Servi
    ce.

    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 perm
    itted. 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 ser
    vice. By default, Get-Service gets all services.

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

Bild 5.3 Hilfe zu den Parametern des Commandlets Get-Service

Bild 5.3 zeigt einige Parameter des Commandlets `Get-Service`. Diese Liste erhält man durch den Befehl `Get-Help Get-Service -Parameter *`.

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

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

Im Fall von

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

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

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.

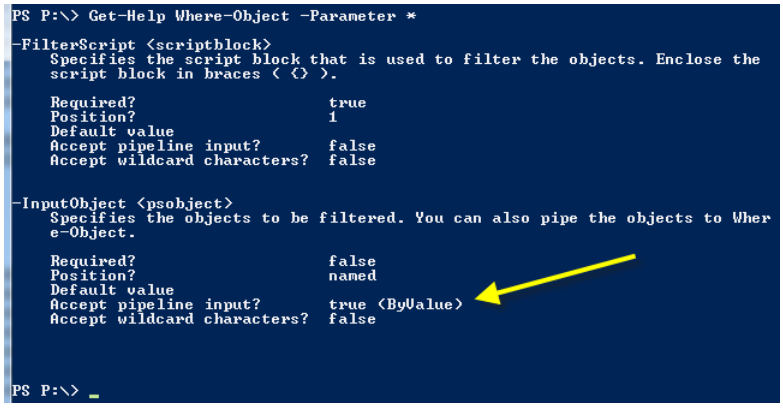


Bild 5.4 Parameter des Commandlets Where-Object

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.

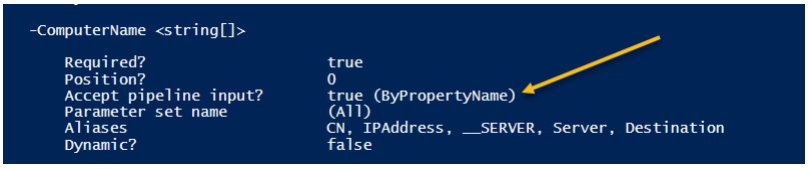


Bild 5.5 Hilfe zum Parameter ComputerName des Commandlets Test-Connection

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 in Kapitel 7 *„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 ab PowerShell 7.0.

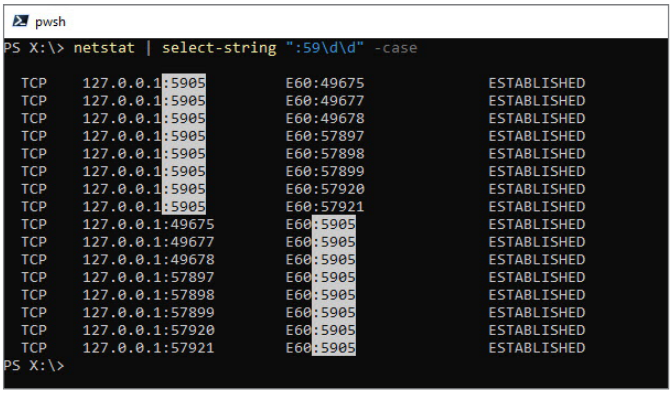


Bild 5.6 Einsatz von Select-String zur Filterung von Ausgaben klassischer Kommandozeilenwerkzeuge

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
```

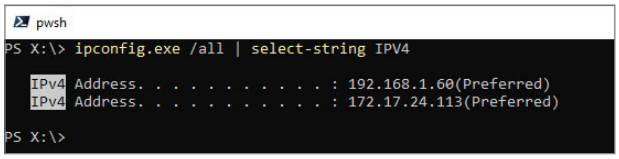


Bild 5.7
Abbildung: Ausführung des obigen Befehls

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 leider nicht zu sehen, da der Verlag leider nicht farbig drucken kann) 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 [`], ASCI-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 WorkingsSet |
>> Format-Table id,name,WorkingSet

   Id Name            WorkingSet
   -- ----            ----------
10828 powershell       92942336
15340 powershell_ise  220946432
 1804 powershell       83664896
 4040 powershell       76177408


PS T:\>
```

Bild 5.8
Zeilenumbruch nach Pipeline-Symbol

5.8 Schleifen

Ein wichtiges Commandlet ist

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

Alias:

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

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

5.8.1 Notwendigkeit für Foreach-Object

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

Beispiele:

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

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

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

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

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

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

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

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

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

Richtig ist:

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

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

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

5.8.2 Parallelisierung mit Multithreading

In PowerShell 1.0 bis 6.2 erfolgt die Ausführung im Haupthread 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 ab 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
```

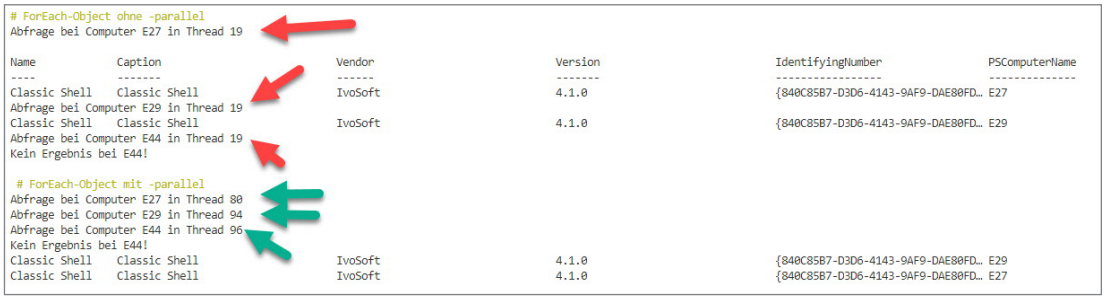


Bild 5.9 Parallelität bei Foreach-Object in PowerShell 7

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]
```

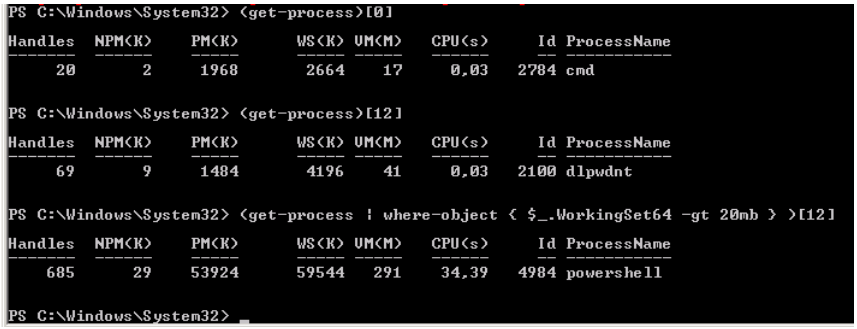


Bild 5.10 Zugriff auf einzelne Prozessobjekte

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
cmd
PS C:\Windows\System32> (get-process)[0].WorkingSet64
2727936
PS C:\Windows\System32> (get-process)[0].WorkingSet64 / 1MB
2,6015625
PS C:\Windows\System32>
```

Bild 5.11 Ausgabe zu den obigen Beispielen

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

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

Bild 5.12
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.

```
pwsh
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()
InvalidOperation: You cannot call a method on a null-valued expression.
PS X:\> $nichtinitialisierteVariable.Substring(10,5)
InvalidOperation: 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 `$_` 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.

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

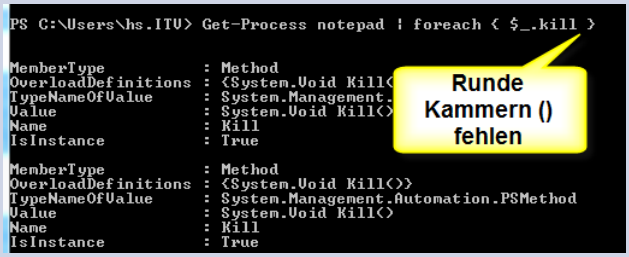


Bild 5.14 Folgen des vergessenen Klammernpaars

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

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

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

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

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

auch schreiben:

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

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

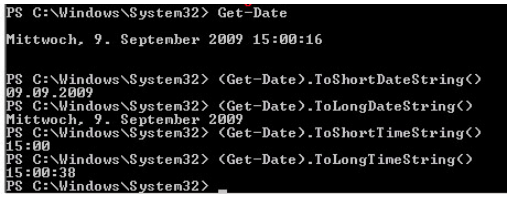


Bild 5.15
Ausgaben der Methoden der Klasse DateTime

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

Im Folgenden werden zwei hilfreiche Commandlets sowie zwei Methoden 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).

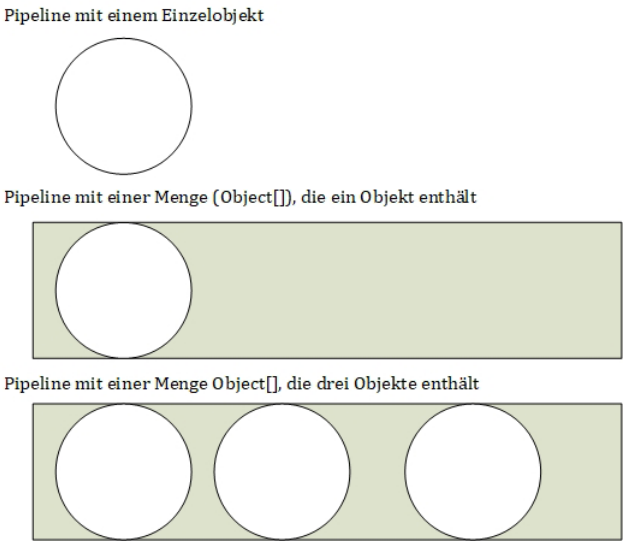


Bild 5.16 Einzelobjekt versus Menge

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 idle).Count
(Get-Date).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 20 MB Hauptspeicher (RAM) verbrauchen?

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

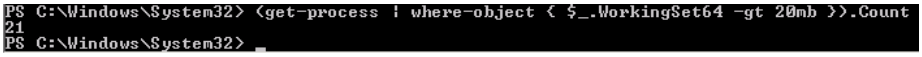


Bild 5.17 Aufruf von Count für eine Pipeline

Es gibt aber (mindestens) einen Fall, in dem `Count` auf einem Einzelobjekt nicht funktioniert. Dieser Fall, der nicht dokumentiert, mir aber in der Praxis aufgefallen 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 = , $firma1
$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 zeigt die folgende Bildschirmabbildung:

```
PS C:\Users\HS> (Get-Date).GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     DateTime                                 System.ValueType

PS C:\Users\HS> (Get-Process).GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     Object[]                                 System.Array

PS C:\Users\HS> (Get-Process)[0].GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     False    Process                                  System.ComponentModel.Component

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

Bild 5.18 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.Service.ServiceController, die das Commandlet `Get-Service` liefert).

Listing 53 [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
```

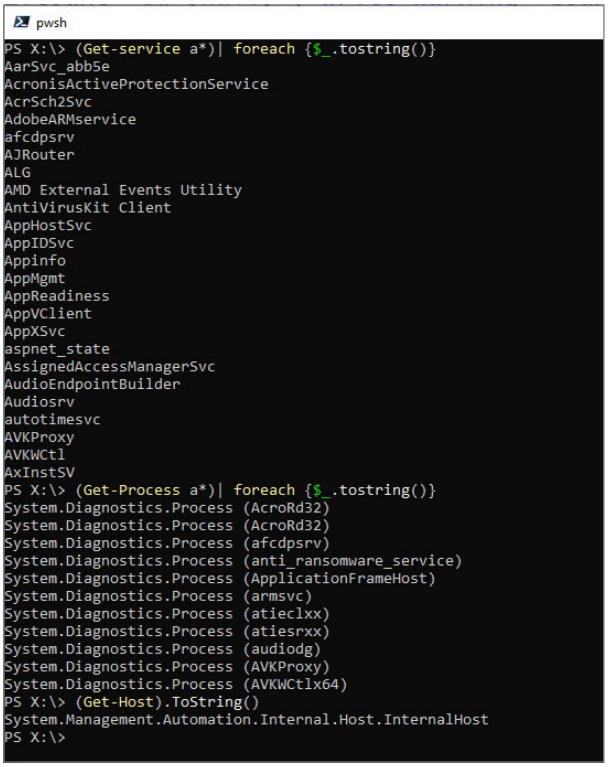


Bild 5.19 Anwendung von ToString() auf Instanzen verschiedener Klassen

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.

5.12.4 Get-PipelineInfo

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

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

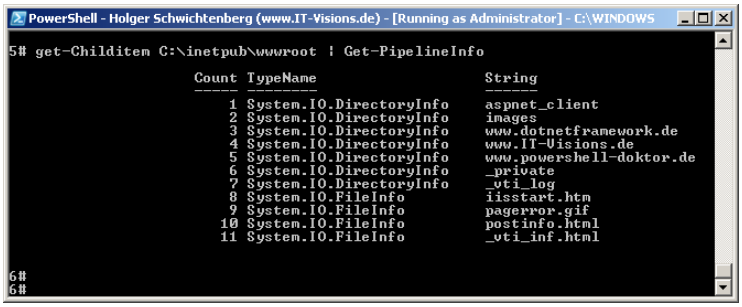


Bild 5.20 Get-PipelineInfo liefert Informationen, dass sich in dem Dateisystemordner elf Objekte befinden. Davon sind sieben Unterordner (Klasse DirectoryInfo) und vier Dateien (Klasse FileInfo).

Das Stichwort Zeichenkettenrepräsentation (Spalte „String“ in der Bildschirmabbildung) 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.

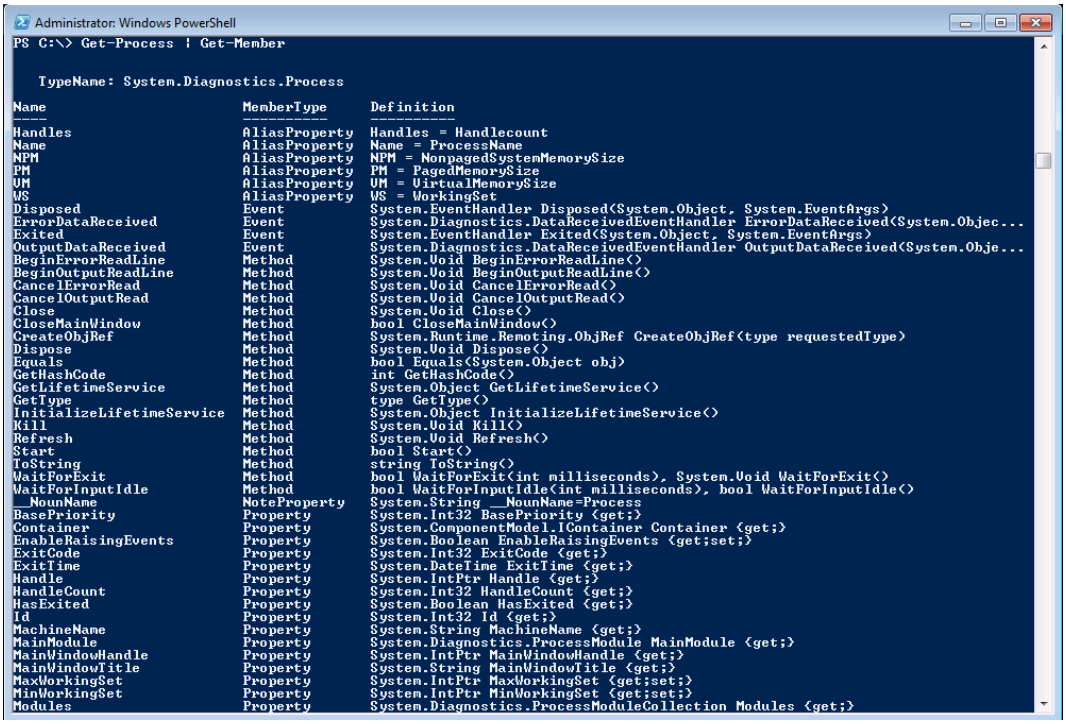


Bild 5.21 Teil 1 der Ausgabe von Get-Process | Get-Member

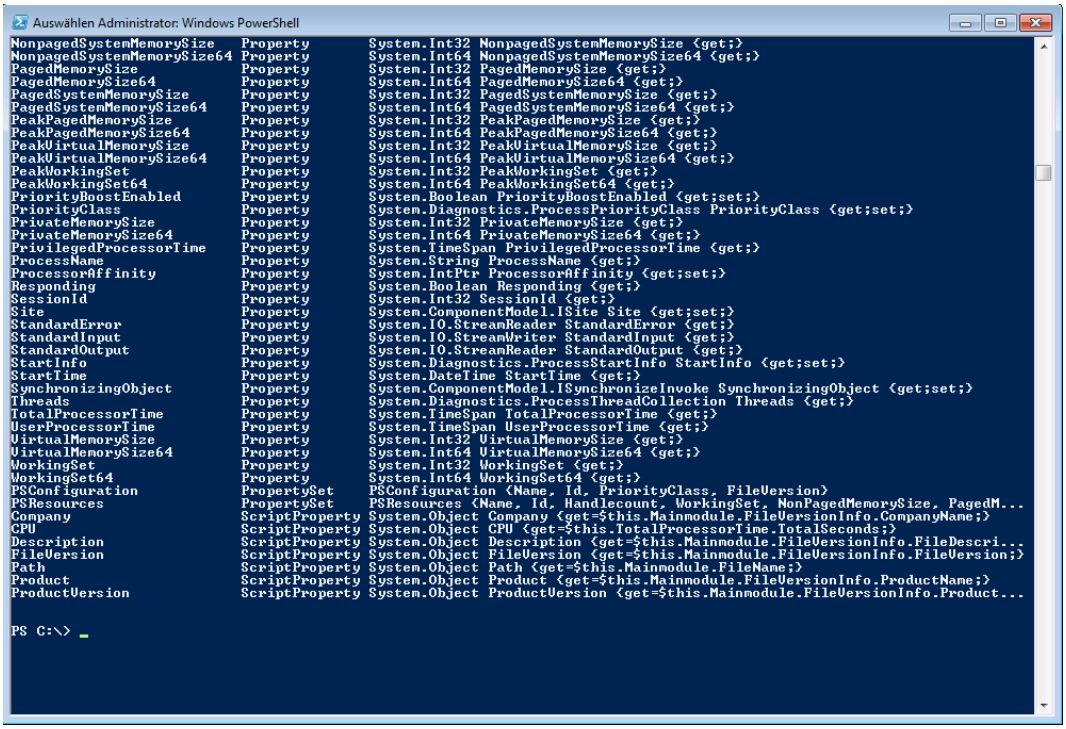


Bild 5.22 Teil 2 der Ausgabe von Get-Process | Get-Member

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 ausgeben lässt. Diese erreicht man über den Parameter `-Membertype` (kurz: `-m`). Der folgende Befehl listet nur die Properties auf:

```
Get-Process | Get-Member -Membertype Properties
```

Außerdem ist eine Filterung beim Namen möglich:

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

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

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

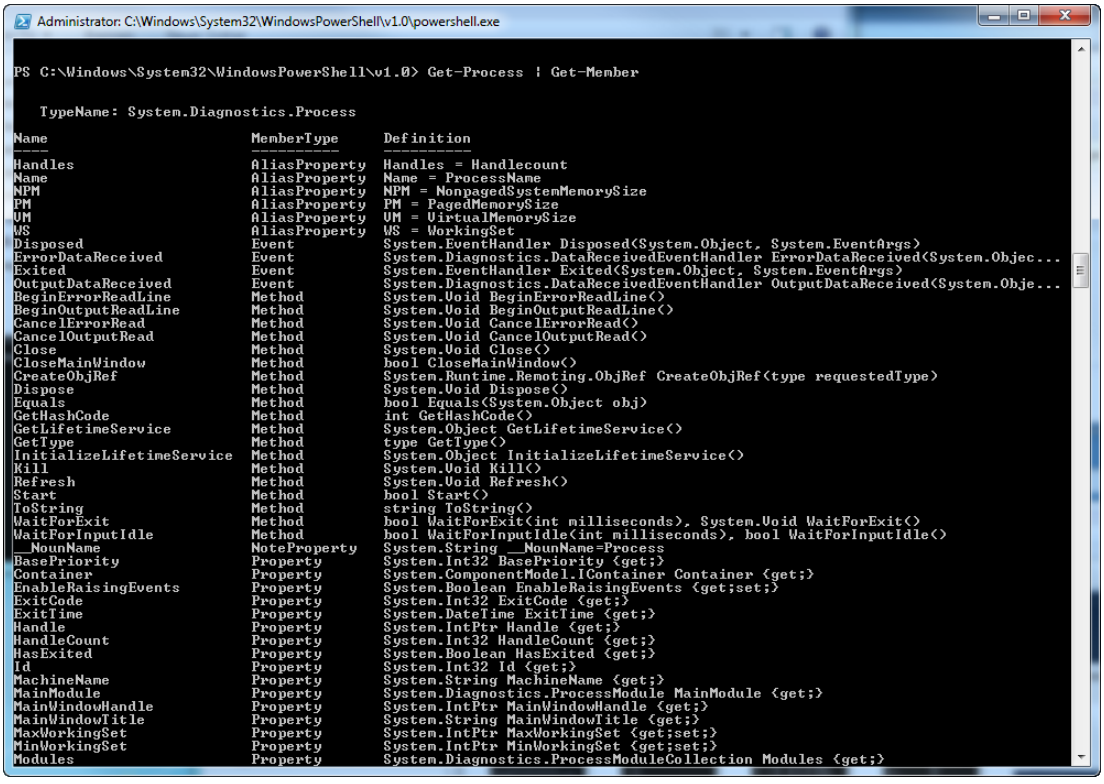


Bild 5.23 Anzeige der Getter und Setter in PowerShell 1.0

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"
```

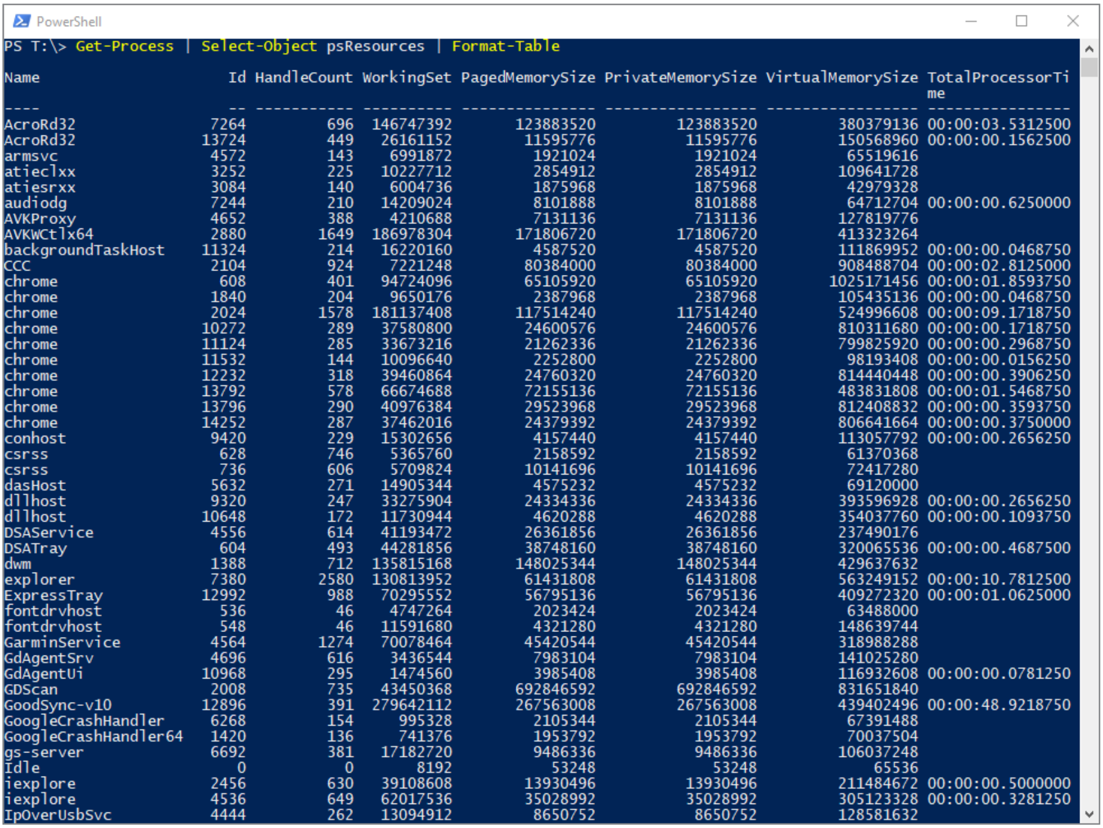
Hier geht nur die o. g. Schreibweise mit `Foreach-Object`.

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.



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

Name	Id	HandleCount	WorkingSet	PagedMemorySize	PrivateMemorySize	VirtualMemorySize	TotalProcessorTime
AcroRd32	7264	696	146747392	123883520	123883520	380379136	00:00:03.5312500
AcroRd32	13724	449	26161152	11595776	11595776	150568960	00:00:00.1562500
armsvc	4572	143	6991872	1921024	1921024	65519616	
atieclxx	3252	225	10227712	2854912	2854912	109641728	
atiesrxx	3084	140	6004736	1875968	1875968	42979328	
audiodg	7244	210	14209024	8101888	8101888	64712704	00:00:00.6250000
AVKProxy	4652	388	4210688	7131136	7131136	127819776	
AVKWCtlx64	2880	1649	186978304	171806720	171806720	413323264	
backgroundTaskHost	11324	214	16220160	4587520	4587520	111869952	00:00:00.0468750
CCC	2104	924	7221248	80384000	80384000	908488704	00:00:02.8125000
chrome	608	401	94724096	65105920	65105920	1025171456	00:00:01.8593750
chrome	1840	204	9650176	2387968	2387968	105435136	00:00:00.0468750
chrome	2024	1578	181137408	117514240	117514240	524996608	00:00:09.1718750
chrome	10272	289	37580800	24600576	24600576	810311680	00:00:00.1718750
chrome	11124	285	33673216	21262336	21262336	799825920	00:00:00.2968750
chrome	11532	144	10096640	2252800	2252800	98193408	00:00:00.0156250
chrome	12232	318	39460864	24760320	24760320	814440448	00:00:00.3906250
chrome	13792	578	66674688	72155136	72155136	483831808	00:00:01.5468750
chrome	13796	290	40976384	29523968	29523968	812408832	00:00:00.3593750
chrome	14252	287	37462016	24379392	24379392	806641664	00:00:00.3750000
conhost	9420	229	15302656	4157440	4157440	113057792	00:00:00.2656250
csrss	628	746	5365760	2158592	2158592	61370368	
csrss	736	606	5709824	10141696	10141696	72417280	
dasHost	5632	271	14905344	4575232	4575232	69120000	
dllhost	9320	247	33275904	24334336	24334336	393596928	00:00:00.2656250
dllhost	10648	172	11730944	4620288	4620288	354037760	00:00:00.1093750
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	70295552	56795136	56795136	409272320	00:00:01.0625000
fontdrvhost	536	46	4747264	2023424	2023424	63488000	
fontdrvhost	548	46	11591680	4321280	4321280	148639744	
GarminService	4564	1274	70078464	45420544	45420544	318988288	
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
iexplore	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.ps1ml*

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 Kapitel 17 *„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
```

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

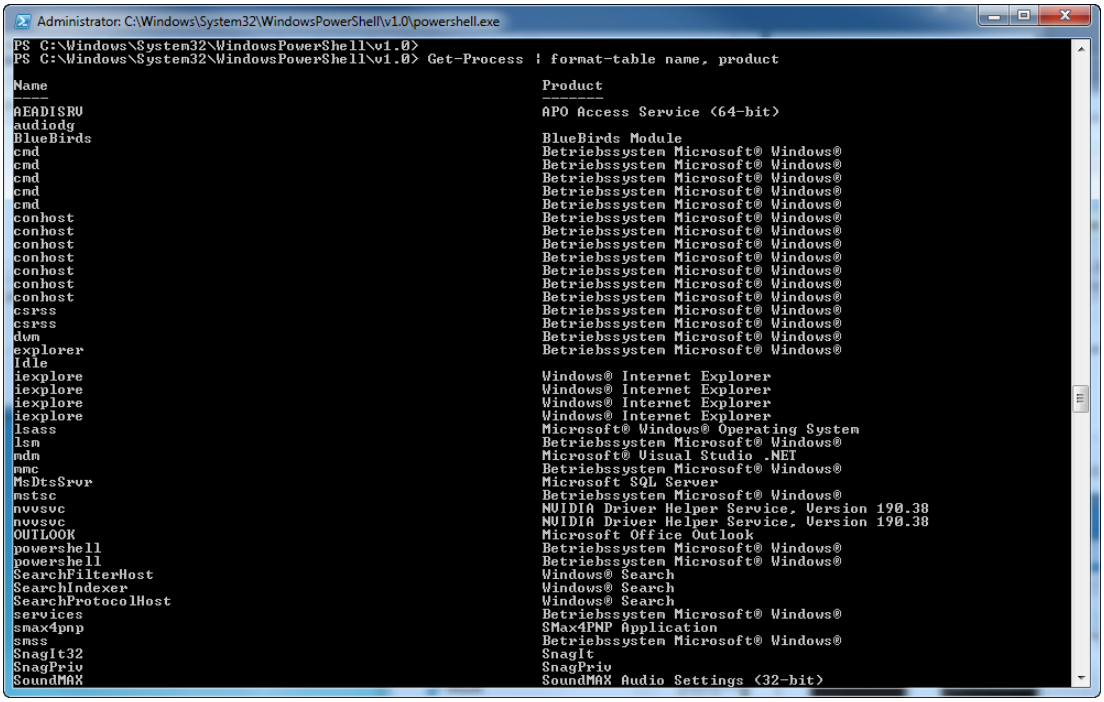


Bild 5.26 Auflistung der berechneten Eigenschaft „Product“

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

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

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

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

Bild 5.27 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 Kapitel 17 *„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 Object members are inconsistently named, provide an insufficient set of public members, or provide insufficient capability. ETS resolves this issue by introducing the ability to extend the .NET object with additional members.“ [MSDN54] Dies heißt im Klartext, dass das PowerShell-Team mit der Arbeit des Entwicklungsteams der .NET-Klassenbibliothek nicht ganz zufrieden ist.

Das 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 `WhereObject` (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:**

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

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

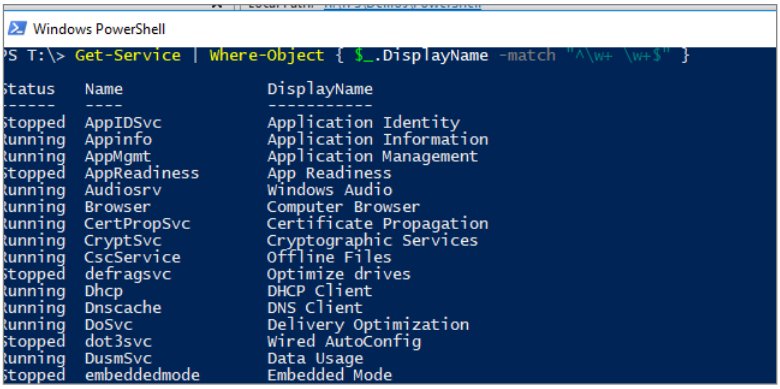


Bild 5.28 Ausgabe zu obigem Beispiel

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

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

Bild 5.29 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 befremdlich.

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

So sind folgende Befehle **nicht** erlaubt:

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

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

Korrekt muss es heißen:

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

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

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

5.13.3 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 5.30
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`).

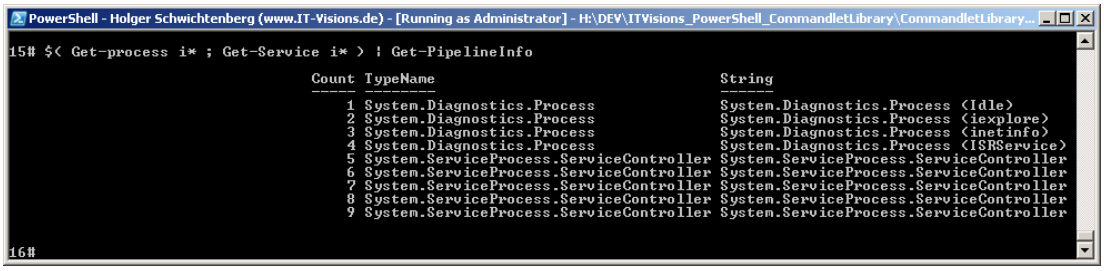


Bild 5.31 Anwendung von Get-PipelineInfo auf eine heterogene Pipeline

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 Bildschirmabbildung). Wie das Bild zeigt, ist das „Kastrieren“ mit zwei Wermutstropfen verbunden:

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

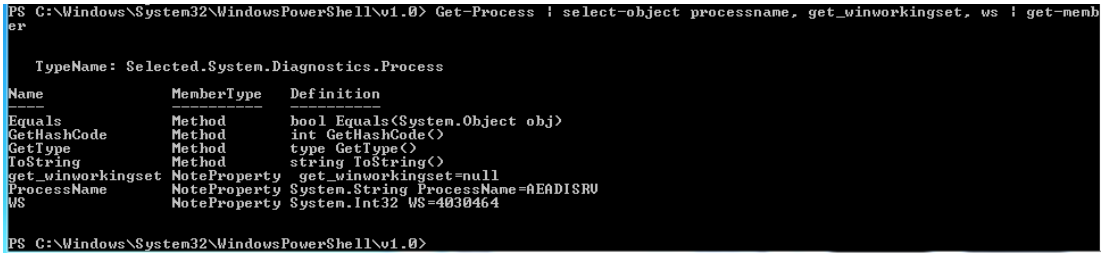


Bild 5.32 Wirkung der Anwendung von Select-Object

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]$_ }
```

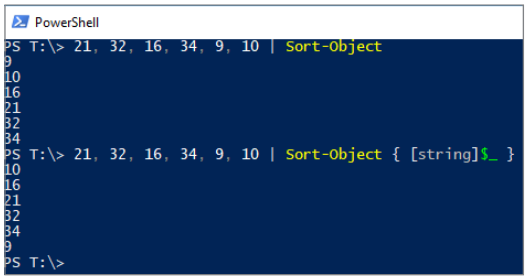


Bild 5.33
Numerische versus alphabetische Sortierung von sechs Zahlen

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
```

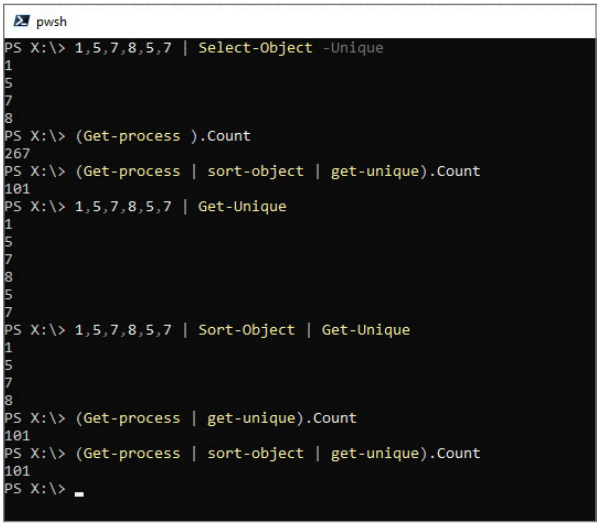


Bild 5.34 Einsatz von Get-Unique und Select-Object -unique

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

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

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

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

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

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

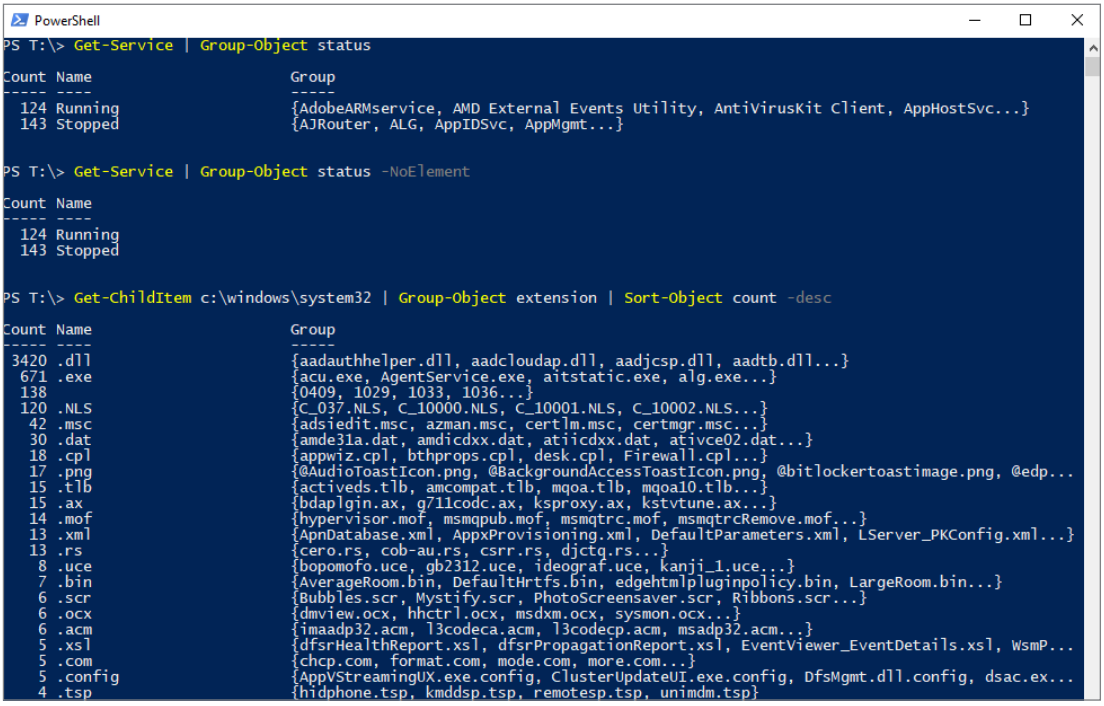


Bild 5.35 Einsatz von Group-Object

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

Count Name                      Group
----- ----                      -----
   46 False                     {Ascd_tmp.ini, bfsvc.exe, bootstat.dat, DtcInstall.log...}
    2 True                      {explorer.exe, WindowsUpdate.log}
```

Bild 5.36 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 54 Finde doppelte Zeichenketten.ps1

```
# Finde doppelte Zeichenketten
# Eingabemenge: Zeichenketten (eingbettet 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 liefern lässt, so kann man diese weiterverwenden, indem man über die Eigenschaft `group` mit `Foreach-Object` iteriert.

Beispiel: Ermittle aus dem Verzeichnis `System32` alle Dateien, die mit dem Buchstaben „b“ beginnen. Beschränke die Menge auf diejenigen Dateien, die größer als 40 000 Byte sind, und 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.

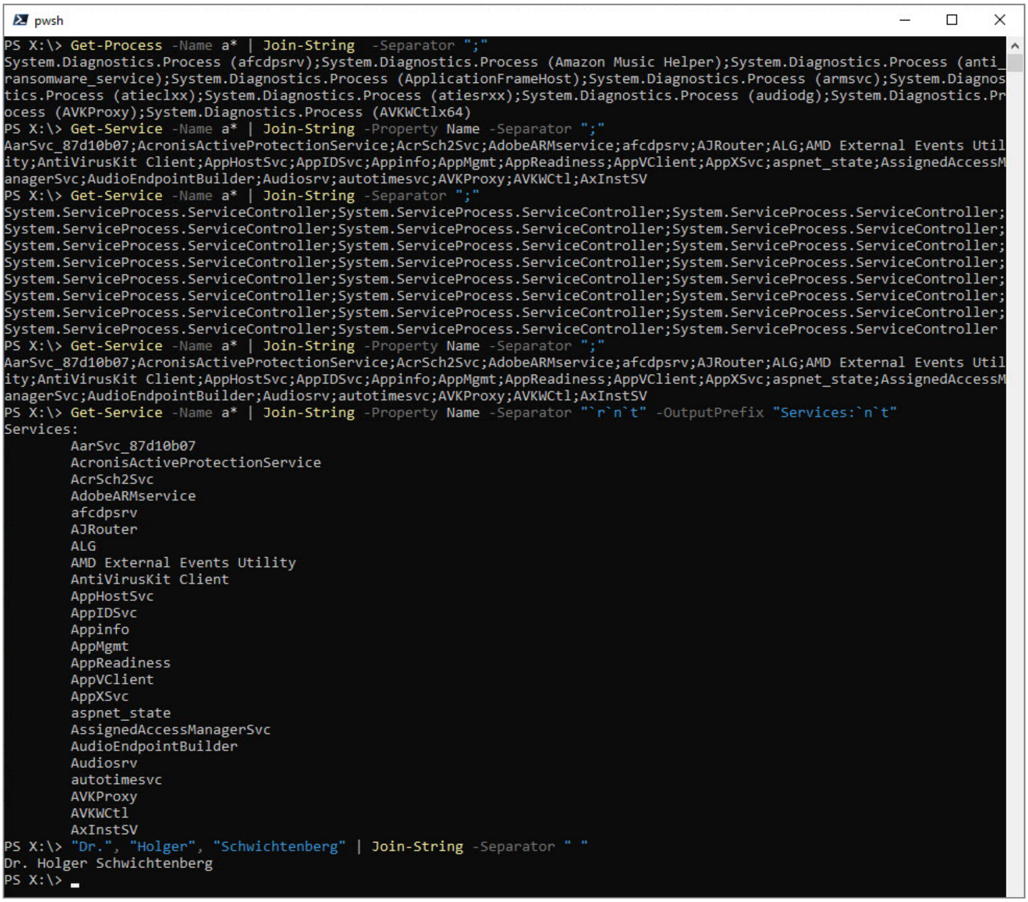


Bild 5.37 Einsatzbeispiele für Join-String

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    : 2590
Average  : 465515,188030888
Sum      : 1205684337
Maximum  : 26575296
Minimum  : 35
Property : length


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

Bild 5.38 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 5.39 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
```

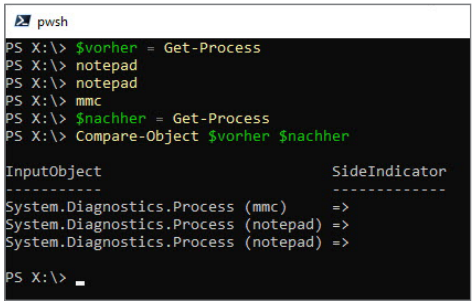


Bild 5.40
Vergleich von zwei Pipelines

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.

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

- Gruppiere die Einträge im System-Ereignisprotokoll nach Benutzernamen.

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

- Zeige die letzten zehn Einträge im System-Ereignisprotokoll.

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

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

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

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

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

- Ermittle aus dem Verzeichnis `System32` alle Dateien, die mit dem Buchstaben „a“ beginnen. Beschränke die Menge auf diejenigen Dateien, die größer als 40 000 Byte sind, und 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 {$_.Length -gt 40000}
| Group-Object Extension | Sort-Object count -desc | Select-Object -first 1 |
Select-Object group | foreach {$_.group} | Select-Object name,length | Format-
Table –autosize
```

Stichwortverzeichnis

Symbole

?? 169
??= 169
& 68
&& 221
% 99
> 258
>> 258
|| 221
$_ 90 f., 99, 105, 174, 486
$? 221 f., 224
$ErrorView 220 f.
$null 104, 150, 797
$PSItem 99
$psUnsupportedConsole Applications 319
$PSVersionTable 24
-and 125
-as 171
-band 205, 425
-Bit 18
-bnot 205, 425
-bor 205, 425
-bxor 205
-cmatch 189
-cnotmatch 189
-expression 685
-force 54
-imatch 189
-inotmatch 189
-ItemsSource 1217
-Join 188
-match 189
-notmatch 189
-or 125
-Parameter 77
-Split 187 f.
-Verbose 53
-whatif 175
.cat 710
.dll 49, 159, 416, 700, 1301 f.
.exe 159, 333
.NET 3, 15, 91, 174, 196, 406, 496, 1203, 1249, 1277, 1292, 1325
– Bibliothek 399
– Klasse 399, 1330
– Runtime Host 13
.NET API Portability Analyzer 1122
.NET CLI 43
.NET Core 15, 38, 362, 1138, 1150, 1325, 1328
.NET Core 3.1 358, 365, 1325
.NET Core SDK 44
.NET Data Provider 785 f.
.NET Framework 4, 13, 271, 537, 617, 890, 1325, 1327, 1331
– 4.0 17
.NET Standard 402, 1329
.nupkg 420
.pfx 518
.pkg 41
.ps1 26, 64, 145, 1302
.psd1 585, 625, 1224, 1302, 1307
.psm1 1224, 1297 ff., 1302, 1307
.psproj 331
.yml 1151
32-Bit 18, 791
64-Bit 314, 791, 850
[Type] 412

A

Ablaufverfolgung 3, 502 f.
About 161
Absent 601
AbsoluteTimerInstruction 449
abstract 1323
Accelerator 165
Accent Grave
– Gravis 55
AccessControl 971, 976
Access Control Entry 972
Access Control List 972, 983
Access Control Type 973
Access Mask 972
AccessMask 724
AccountDisabled 1016
Account Manager 639, 658
AceFlags 973
ACL 975, 987
ACS 1154
Active Directory 3, 264, 436, 591, 639, 654, 1005, 1012, 1016, 1031, 1061, 1063, 1225
– PowerShell 1031
– Struktur 1063
– Suche 1021
Active Directory Application Mode 1063
Active Directory Domain Services 1060
Active Directory Service Interface *siehe* ADSI
ActiveScriptEventConsumer 450
Active-Scripting 152
ActiveX Data Objects 785, 796, 1002
ADAccount 1037
Adapted Type System *siehe* ATS
ADComputer 1037
Add() 415
Add-ADGroupMember 1041, 1059
AddCommand() 1315
Add-Computer 857
Add-Content 744, 774
Add-DirectoryEntry 654
Add-DistributionGroupMember 1080
Add-Feature 740
Add-JobTrigger 532
Add-LDAPObject 1029, 1226 f.
Add-LocalGroupMember 1076
Add-Member 122, 483, 486, 1235
Add-ODBCDSN 837
Add-PSSnapin 362, 1253, 1258

AddScript() 1312, 1315
ADDSDeployment 1060, 1062
Add-Type 417, 487, 495, 712, 821
Add-VirtualHardDisk 657
Add-VMDisk 1118
Add-VMDrive 1118
Add-VMHardDiskDrive 1097, 1109
Add-VMNIC 1118
Add-VMSwitch 1097
Add-WBSystemState 738
Add-WindowsCapability 901
Add-WindowsFeature 737, 890, 894 ff., 1060 f.
Administration
– delegiert 661
– webbasiert 337, 666
Administrator 154, 296, 972
Administratorrechte 272, 292, 296, 307, 309 f., 474, 629, 731, 913, 971, 1135, 1138
ADODB.Connection 1003
ADO.NET 785, 793, 1002, 1022
ADPowerShell 1031, 1037
ADSI 999, 1003 f., 1007, 1009, 1077
– Bindung 1005 f.
– COM 1003, 1009
– Container 1011
– .NET 997, 999, 1003
– Pfad 1005
AdsPath 1021
ADUser 1037
Advanced Function 1221, 1230
ADWS 1033
AgentPC 999
AKS XXVII, 1164
Akte X 998
Aktivierung 659
Aktivität 544, 548
Alias 45, 58, 261, 689
Aliaseigenschaft 115, 121
AliasInfo 58
AllNodes 598
AllowClobber 32, 628
AllowEmptyCollection 1230
AllowEmptyString 1230
AllowNull 1230
AllSigned 152, 515
Alvin Kersh 998
Amazon Web Service *siehe* AWS
Änderungshistorie 790
Animation 1217
Ankerelement 191
Anwendungspool 1091, 1093
Anzeigesprache 584
Apache 1138
AppDomain 418, 821
AppendChild(). 756
Apple Software Package 41
AppLockerPolicy 906 f.
appSettings 340
AppX 883
Args 150, 174, 281
Array 197, 201, 203, 1334
ArrayList 200 f.
Artifact 1189
AsJob 522, 541
ASP.NET 436, 1141
ASP.NET Core 357
Assembly 401, 404, 415 f., 617, 623, 630, 700, 1258, 1288
– verbreiten 1332
AssocClass 919
ASSOCIATORS OF 451
Assoziation 443
– WMI 440, 443
Asynchronous 959
ATS 121, 468, 1004
Attribut 1317, 1322, 1333
– indiziert 1333
Audio 410
Aufgabe
– geplant 527
Aufzählung 424
Aufzählungstyp 304
Ausdruck 66
– Regulär 189
Ausdruckauflösung 181
Ausdrucksmodus 66
Ausführungsrichtlinie 152
Ausgabe
– mehrspaltig 242
– unterdrücken 256
Ausgabeobjekt 1262
Auslagerungsdatei 659
Authentifizierung 496, 1016, 1060
AuthorizationRuleCollection 977 f.
AutoUpdate 858
AWS 371
Az 1157
az aks 1170
az.cmd 1158, 1169
az extension 1170
Azure 67, 357
– Kontext 1160
– Kubernetes Services 1164
– Resource Group 1155, 1161
– SQL 1163
– SQL Server 1163
– Subscription 1155, 1160
– Web App 1161
– Web Portal 1155
Azure CLI 1158, 1169
Azure Cloud Shell 343
Azure Container Registry 1138
Azure Container Service *siehe* ACS
Azure DevOps XXVII, 160, 1188 f.
Azure DevOps CLI 1189
Azure Kubernetes Services *siehe* AKS
AzureRM 1157

B

BackgroundColor 175, 320, 342
Background Intelligent Transfer Service 958
Background Intelligent Transfer Service *siehe* BITS
Backspace 183
Backup 737 f., 820
Backup-GPO 1069
Backup-SqlDatabase 818, 820
Base 1021, 1044
bash 305, 343, 357, 381, 383
BasicHtmlWebResponseObject 946
Basisauthentifizierung 269
Basisimage 1150 f.
Basisklasse 789
Batterie 867
Bedingung 212
Beep 183
Beep() 413
Befehl
– Extern 45, 67
Befehls-Add-On 79
Befehlseingabefenster 34
Befehlsgeschichte 677
Befehlsmodus 66
Befehlsobjekt 794
Begin 714
BeginProcessing() 1252, 1256
Benutzer 435, 1026, 1318, 1321
– Active Directory 1012
– anlegen 1015
– lokal 1075
– löschen 1017
– umbenennen 1017
– verschieben 1018
Benutzerabmeldung 682
Benutzeranmeldung 682
Benutzerdaten lesen 1046
Benutzer-DSN 839
Benutzereingabe 493
Benutzergruppe 1059
Benutzerkennwort 1016
Benutzerkontensteuerung *siehe* UAC
Benutzerkonto 1046, 1336
Benutzername 496
Benutzerschnittstelle 434
Berechnung 137
Best Practice 995
Betriebssystembasis-Image 1119
Bezeichner 1332
Beziehung 1321

Bibliothek 1328
Big Endian 1021
Bild 653
Bildschirmschoner 489, 999
Binärdatei 774
Binäre Operation 205
Binärmodul 1297
Bindung
– ADSI 1005
– serverlos 1006
– WMI 458
Bing 1212
BIOS 435
Bitflag 205, 424
BitLocker 739
– Überblick 739
Bitmap 712 f.
BITS 47, 958, 961
Bitweise Operation 205
Bitweises NOT 205
Bitweises ODER 205
Bitweises UND 205
Blatt 1003
Blockierung 155
BMC 434
Board 1189
Boolean 221
Boot-Konfiguration 435
Bootstrap 421
break 206, 208, 217, 225
Build 1202
bxor 699, 1077
Bypass 152
ByPropertyName 95 f.
Byte 178
ByValue 95 f.
BZIP2 722

C

C# 4, 161, 487, 489, 1189, 1221, 1249 f., 1256, 1325
C++ 1329 f.
C++/CLI 1249
CAB 83
Canvas 1214
Carriage Return 183
cat 383, 385
CategoryView 220
CATID 452
CD 415
Certificate
– Zertifikat 988
ChangeAccess 731
Checkpoint-Computer 862
Checkpoint-VM 1096, 1111
Children 1002
ChildSession 286
Chkdsk() 476
CHKDSK 436
chmod 384
Chocolatey 888
Chocolatey.org 886
chown 388
Chrome 591, 887
CIL 13, 1327
CIM 8, 434, 437
– Repository 444
CimClass 457 f., 460, 469
CimClassProperties 469
CIM Explorer 351
CimInstance 457 f., 460, 469
CimInstanceProperties 469
CimProperty 469
CIM Query Language *siehe* CQL
Cisco 434
City 1038 f.
class 234
ClassCreationEvent 570
ClassDeletionEvent 570
ClassModificationEvent 570
Clear-BitLockerAutoUnlock-Funktion 741
ClearCase 160
Clear-Content 744
Clear-DnsClientCache 932
Clear-EventLog 270, 965
Clear-History 678
Clear-Host 220, 677
Clear-Item 689
Clear-RecycleBin 711
Clear-Variable 173
Click 1215
Clipboard *siehe* Zwischenablage
CliXml 757
Close() 1214
Cloud 1155
CLR 13, 24, 1301, 1327
cmd.exe 91
Cmdlet 1285
CmdletBinding 1230, 1236
Cmdlet Help Editor 1288
cn 1012
Codeausschnitt 316, 335
Codeeigenschaft 115, 121
Color 869
COM 15, 427 f., 711, 1330
– Kategorie 452
– Klasse 430
– Komponente 435
– Moniker 1005
– Sicherheit 448
Commandlet 3, 45, 57, 67, 71, 91, 267
– binär 1249
– erstellen 1221, 1249
– Klasse 1251
– Konvention 1275, 1293
– Provider 262
– Proxy 1241
– Verkettung 1273
Command Line Event Consumer 451
Command Mode 66
Comma-Separated Values 748
CommitChanges() 410, 1000, 1009 f., 1016
Common Information Model *siehe* CIM
Common Intermediate Language *siehe* CIL
Common Language Runtime *siehe* CLR
Common Language Specification 1327
Common Management Information Protocol 434
Common Parameter 51
Common Type System 1327
compare 89
Compare-Object 89, 141, 698
Compare-VM 1096, 1113 f.
CompatiblePSEditions 369
Complete-BITSTransfer 959
Complete-Transaction 362, 507 f., 510
Component Object Model 4
Component Object Model *siehe* COM
Compress-Archive 721
Computer 442, 1026, 1321
Computergruppe 340
ComputerInfo 849
Computername 149, 374, 857, 869
Computerrichtlinie 504
Computerverwaltung 849, 913
ConciseView 220
configuration 592
ConfigurationData 599
ConfigurationID 609
ConfigurationNamingContext 1035
confirm 53 f.
Confirm 52, 54, 870, 927, 1043, 1046, 1236 f., 1285
ConfirmPreference 54, 1237
Connect-AzAccount 1160
Connection 792
Connect-VMNetworkAdapter 1097
ConsolePaneBackgroundColor 320
Console.WriteLine() 1280, 1294
Container 1011, 1070, 1123, 1131
Container-Klasse 1003
continue 53, 217
Continue 206, 208, 225, 229
ConvertFrom-JSON 764
ConvertFrom-Markdown 761 f.

ConvertFrom-String 749
ConvertFrom-StringData 582
Convert-Html 761
Convert-String 748
ConvertTo-ContainerImage 1149
ConvertTo-CSV 748
ConvertTo-DataTemplate 1217
ConvertTo-JSON 764, 773
ConvertTo-SecureString 498, 1060
ConvertTo-WebApplication 1093
ConvertTo-XML 759
Convert-VHD 1097, 1108, 1111
Convert-Xml 759
copy 703
Copy-ContainerFile 1146
Copy-GPO 1068
Copy-Item 225, 286, 689, 703, 707 f., 844, 990
Copy-NetFirewallRule 935
Copy/Paste 303
Copy-ToZip 722 f.
Copy-VMFile 1115
CORBA 1330
Count 109, 199, 218
Country 1039
CPU 142
CQL 451, 466
Create() 477, 1241
CreateCommand() 794
CreateElement() 756
CreateInstance() 868
CreateObject() 431
CreationTime 712, 1322
Credential 1060
Credentials 1041
CSV 64, 493, 745 ff., 847, 1055, 1088
CSV-Datei 142
CultureInfo 1262
CurrentThread 101
Cursor 789 f.
CustomerID 1239
CVS 160

D

Dana Scully 998
DataReader 788 ff., 794, 796, 798 f.
DataRow 121 f.
DataSet 788 f., 796, 799 ff.
Data Source Name *siehe* DSN
DataTable 799
Datei 237, 436, 1318
– Eigenschaft 699, 711
– kopieren 703
– löschen 47
– Rechte 436
– verschieben 703
Dateiname 45
Dateinamenerweiterung 142, 145
Dateisystem 3, 706, 975, 1321, 1332
Dateisystemfreigabe 591, 724
Dateisystemkatalog 710
Dateisystemoperation 673
Dateisystemstruktur 705
Dateiversionsverlauf 736
Datenabfrage 452
Datenbank 444, 654, 785
Datenbankmanagementsystem 792
Datenbankverbindung 792
Datenbankzeile 121
Datenbankzugriff 785
Datenbereich 581
Datenbindung 1217
Datendatei 581
Datenmenge 261
Datenquelle 837 f.
Datenquellensteuerelement 788
Datentyp 164, 174, 201, 1007, 1273, 1333
– .NET 92
– PowerShell 164
– WMI 439
Datenzugriff 793
DateTime 104, 107, 111, 412
Datum 195
Day 104
DB2 787, 837
dBase 837
DbCommand 794
DbDataReader 796
DBG 505
DbNull 797
DbProviderFactories 787
DCOM 431, 444, 460, 859, 1268, 1281
– Konfiguration 435
dcpromo 1060
DDL 451
debug 1280
Debug 53 f.
Debugger 36
Debugging 3, 35, 335, 505, 551
Debug-Modus 505
DebugPreference 54, 1280
Decimal 177 f.
Deep Throat 998
Default Domain Policy 1073
DefaultNamingContext 1035
Deinstallation 875 f.
Delete() 1322
Deleting 869
Delimiter 745
Deployment 1332
Deployment Image Servicing and Management *siehe* DISM
Description 1013, 1039, 1246
DESCRIPTION 1240
Deserialisierung 278
Desired State Configuration *siehe* DSC
Desktop 435
Desktop-Anwendungen 1328
Desktop Management Task Force *siehe* DMTF
Destruktor 1318, 1334
Deutsche Telekom 515
Dezimalzahl 177
DHCP 923 f., 926
Diagnose 502
Dialogfenster 496, 1203
diff 89
Digest 269
dir 387
Directory 975
DirectoryEntry 121 f., 167, 406 f., 1001 ff., 1007 ff., 1012, 1015, 1018
DirectoryInfo 113, 127, 483, 699, 975, 1322 f.
Directory Management Objects 1063
DirectorySearcher 167, 1022
DirectorySecurity 977
DirectoryString 1013
Disable-ComputerRestore 862
Disable-ExperimentalFeature 376
Disable-JobTrigger 532
Disable-Mailbox 1080
Disable-NetFirewallRule 935
Disable-PnpDevice 867
Disable-PSRemoting 275
Disable-PSSessionConfiguration 285, 287
Disable-VMIntegrationService 1100
Disable-WindowsOptionalFeature 901, 903
Disk Quotas 436
DISM 901
Dismount-VHD 1108
DisplayName 1039
Distinguished Name 1005, 1010, 1013, 1035, 1039
Distributed COM 269
Distributed Component Object Model *siehe* DCOM
Distributed COM *siehe* DCOM
Distributed File System 436
Distributed Managements Objects *siehe* DMO
DML 451
DMO 820
DMTF 269, 434
DNS 297, 932, 1060, 1122 f.
DnsClient 925, 932
DNS-Client 928

DNSClient 926, 928
DNS-Konfigurationseinstellungen
– Per WMI abfragen 929
DNS-Server 436, 926, 931
do 206
Docker 44, 67, 358, 1119, 1122, 1124, 1131, 1133, 1135 f.
Docker Compose File 1141
docker.exe 1131
Dockerfile 1146, 1151
Docker Hub 1138
Docker-Image 1138
DockPanel 1214
DOCX 781
Dokument 743
Dokumentation
– Active Directory 1015
– .NET 86
Dollarzeichen 138, 181
Domain 935
Domain Controller 1061
Domain Specific Language *siehe* DSL
Domäne 857, 1005, 1063, 1318, 1321
– Beitritt 297, 857
– hinzufügen 857
DOS 4
dotnet.exe 43, 223
DotNetTypes.Format.ps1xml 237, 245 f.
Dot Sourcing 64, 151, 514, 617, 1223, 1227
Double 177
DownloadString() 409, 943
DriveInfo 412
Driver 840
DriveType 422
Druckauftrag 868
– löschen 868
Drucker 237, 257, 435, 869
– verwalten 867, 869
Druckerport 868
Druckerverwaltung 867, 869, 923
DSC 587
DSC Pull Server 604, 609
DSL 668
DSN 837, 839 f.
DuplexingMode 869
DVD 415, 1114, 1118

E

echo 66
Echo 66
Edit-NanoServerImage 1123
Eigenschaft 115 f.
Eigenschaftenzwischenspeicher 1000
Eigenschaftssatz 115, 118
Eingabe 493
Eingabeaufforderung 679 f.
Eingabedialog 495
Eingabemaske 1205
Eingabeobjekt 1270
Eingabesteuerelement 1217
Eingabeunterstützung 315, 335
Einzelobjekt 109 f.
Einzelschrittmodus 500
elevated 913
Elevated 154, 158, 913, 971, 1138
Elevation 154
Else 212
Emacs 146, 305, 337
E-Mail 943, 954
– Adresse 943
– EmailEvent 454
– senden 942
EmailAddress 1039
EmailEvent 454
Enable-BitLocker 741
Enable-ComputerRestore 862
Enable-ExperimentalFeature 376
Enable-JobTrigger 532
Enable-NetFirewallRule 935, 941
Enable-ODBCPerfCounter 837
Enable-PnpDevice 867
Enable-PSRemoting 273, 315, 869, 941
Enable-PSSessionConfiguration 285, 287
Enable-PSSsessionConfiguration 287
Enable-VMIntegrationService 1100
Enable-WindowsOptionalFeature 901, 903
Encoding 745
End 714
EndProcessing() 1252, 1256
endregion 324
Enter-PSSession 275 f., 285, 290, 391, 505, 1125
Enum 423
EnumerateCollection 1264
Enumeration 304
Enumerationsklasse 422
env 383, 855
Environment 589
Ereignis 1317, 1334
– PowerShell 569, 578
– WMI 449, 569
Ereignisabfrage 453
Ereigniskonsument 449 ff.
– permanent 449
– temporär 449
Ereignisprotokoll 142, 362, 435 f., 445, 451, 503, 591, 963
– Überwachung 454, 571
Ereignisprovider 449
Ereignissystem 569
Error 175, 221, 232, 869
ErrorAction 53 f., 221, 229 ff., 707
ErrorActionPreference 54, 175, 221, 230, 1048
ErrorBackgroundColor 30
ErrorRecord 225, 228, 231 f.
ErrorVariable 53, 231
Ethernet 925
ETS 1295
Event 438
EventConsumer 438
Event *siehe* Ereignis
EventViewerConsumer 450
Example 1246
EXAMPLE 1240
Exception 225, 228, 233, 1276, 1282
Exchange Management Shell 656, 1079
Exchange Server 86, 436, 445, 1079
ExecuteNonQuery() 794
ExecuteReader() 794, 796
ExecuteRow() 794
ExecuteScalar() 794
ExecutionPolicy 27 f.
EXIF 712
Exists() 1008
exit 206
Exit-PSSession 277, 285
Expand-Archive 721
explorer.exe 737
Export-Alias 63 f.
Export-CliXml 493, 757
Export-Console 619, 1259
Export-Counter 969
Export-CSV 127, 374, 493, 748
Export-ModuleMember 624
Export-PfxCertificate 990
Export-VM 1096, 1113
Export-VMSnapshot 1112
Express 814
Expression 240
Expression Mode 66
Extended Reflection 91
Extensible Application Markup Language *siehe* XAML
Extrinsic Event 449

F

facsimileTelephoneNumber 1039
false 48, 54, 166, 174
Fax 1039
FBI 998, 1018
Feature 890
FeatureOperationResult 896

Fehler 53
Fehlerausgabe 220
Fehlerbehandlung 221, 1329
Fehlerklasse 206, 228
Fehlermeldung 48
Fehlerstatus 221
Fehlersuche 499, 1277
Fehlertext 206
Fernaufruf 270
Fernausführung 149, 269
– Hintergrundauftrag 525
Fernverwaltung 269
Fernzugriff 269
Festplatte
– virtuell 1108
Festplattenverschlüsselung 739
Fibre-Channel 1096
Field 116
File 975, 1320
File History 736
FileInfo 113, 127, 483, 699 f., 975, 1322 f.
FileInformation 909
FileSecurity 975, 977
FileSystem 652, 689
FileSystemAccessRule 978
FileSystemInfo 1322
FileSystemObject 1320
FileSystemRights 424
FileSystemWatcher 577
FileVersionInfo 888
filter 695
Find() 1002
Find-Module 630 f.
Find-Package 887
Firefox 591
Firewall 386, 659
Firewall-Regel 938
First 102
fish 357
For 207, 210
Force 52, 697, 1043
foreach 142
Foreach 105, 210, 549, 1002, 1266
Foreach-Object 99, 102, 105, 199, 210 f., 217, 501, 1088, 1264
ForEach-Object 548
Foregroundcolor 77
Forest 1063
Format 240
Formatkennzeichner 251
Format-List 90, 238, 977
Format-Table 103, 118, 142, 238, 240, 247 f., 977
Format-Wide 237 f., 242 f.
Format-Xml 753 f.
Form Feed 183
Fortschrittsanzeige 259
Fox Mulder 998
FoxPro 837
Framework Class Library 399
Freigabe 733
FTP 374
FullAccess 731
FullName 110, 1322
function 45, 206, 214, 220, 261
Funktion 45, 214, 261
– eingebaut 220
– fortgeschritten 1230

G

GAC 416 f.
Ganzzahl 177
Gast 1095
Gateway 926
GeneralizedTime 1013
Geplante Aufgabe 527
Gesamtstruktur 1063 f.
Geschäftsanwendung 1292
GetAccessRules() 977 ff.
GetAssemblies() 418
GetDrives() 411
GetFactoryClasses() 787
GetLongDateString() 107
GetLongTimeString() 107
GetNames() 423
GetObject() 431
GetRelated() 919
GetShortDateString() 107
GetTempName() 430
Getter 116 f., 1333
GetTimestamp() 395
GetType() 110, 174, 255, 369
Get-Acl 971, 975, 977, 987
Get-ADComputer 1041
Get-ADDomain 1064
Get-ADDomainController 1064
Get-ADForest 1064
Get-ADGroup 1041, 1059
Get-ADGroupMember 1041, 1059
Get-ADObject 49, 997, 1028, 1041 ff.
Get-ADOptionalFeature 1064
Get-ADOrganizationalUnit 1041, 1045
Get-ADPrincipalGroupMembership 1059
Get-ADRootDSE 1064
Get-ADUser 1041, 1046 f.
Get-Alias 58 f.
Get-AppLockerFileInformation 906
Get-AppLockerPolicy 906 f.
Get-AuthenticodeSignature 519
Get-AzAks 1169, 1172
Get-AzAppServicePlan 1160
Get-AzLocation 1160
Get-AzResource 1160
Get-AzResourceGroup 1160
Get-AzSqlServer 1160
Get-AzSubscription 1160
Get-AzWebApp 1160, 1163
Get-BitLockerVolume 740, 742
Get-BITStransfer 960
Get-BITSTransfer 959
Get-BPAModel 995
Get-BPAResult 995
Get-CDRomDrive 654, 865
Get-ChildItem 46 ff., 50, 90, 127, 131, 136, 142 f., 261, 695 f., 721, 843, 873
– BitLocker 742
Get-CimAssociatedInstance 459
Get-CimClass 433, 459, 466
Get-CimInstance 362, 433, 454, 459, 461 ff., 477, 866, 869
Get-Clipboard 498, 633, 649
Get-Command 56, 71 f., 636
Get-Computerinfo 857
Get-ComputerInfo 849, 1262
Get-Computername 1255
Get-ComputerRestorePoint 862
Get-Container 1148 f.
Get-Containerimage 1138
Get-ContainerImage 1140, 1149, 1153
Get-Content 689, 707, 743, 774, 847, 872, 1088, 1218
Get-Counter 270, 967 f.
Get-Credential 78, 293, 496, 662, 914, 943
Get-Culture 186
Get-DataRow 654, 808
Get-DataTable 654, 807
Get-Date 107, 195 f.
Get-DHCPServer 31, 924
Get-DirectoryChildren 654
Get-DirectoryEntry 181, 654, 1275
Get-DirSize 1221
Get-Disk 690, 692, 865, 1263 f., 1268, 1270, 1272 f., 1281
Get-DisplaySetting 867
Get-DnsClient 928 f.
Get-DnsClientCache 932
Get-DnsClient-Funktion
– Beispiel 929
Get-DnsClientServerAddress 928 f.
Get-DomainController 31, 33, 1006
Get-DSCConfiguration 603
Get-DVDDrive 657
Get-Error 221, 232
Get-Event 573
Get-EventLog 23, 142, 270, 362, 963 f.
Get-ExCommand 1079
Get-ExecutionPolicy 27
Get-ExperimentalFeature 376

Get-ExportedType 700
Get-Filecatalog 86
Get-FileHash 713 f.
Get-FileVersionInfo 700, 888
Get-FirewallRule-Funktion 937
Get-FloppyDrive 657
Get-Flug 1292
Get-Flugziele 1292
Get-Font 856
Get-FreeDiskSpace 692, 694
Get-GPInheritance 1073
Get-GPO 1068 f.
Get-GPOReport 1071
Get-GPPermissions 1074
Get-GPPrefRegistryValue 1074
Get-GPRegistryValue 1074
Get-GPResultantSetOfPolicy 1071
Get-GPStarterGPO 1068
Get-Help 75 ff., 80 ff., 85, 161, 374, 378, 1221, 1240, 1248, 1287
Get-History 378, 677
Get-Host 378, 678
Get-HotFix 270
Get-Item 711, 843, 1086, 1091
Get-ItemProperty 712, 843
Get-Job 521, 523
Get-JobTrigger 532
Get-Keyboard 866
Get-LDAPChildren 1029, 1226
Get-LDAPObject 1029, 1226 f.
Get-LocalGroupMember 1076
Get-LocalUser 1075
Get-Location 57, 689
Get-LogicalDiskInventory 690
Get-LoremIpsum 705
Get-Mailbox 1079 f.
Get-Mailboxdatabase 1079
Get-MarkdownOption 761
Get-Member 89, 108, 113, 115 f., 127 f., 412, 421, 428, 469, 1004, 1018, 1264
Get-MemoryDevice 654, 865
Get-Module 240, 623, 634, 1297
Get-MountPoint 31
Get-MPComputerStatus 863
Get-MultiTouchMaximum 867
Get-NanoServerPackage 1123
Get-NetAdapter 925
Get-NetAdapterBinding 925
Get-NetFirewallAddressFilter 935
Get-NetFirewallAddressFilter-Funktion 937
Get-NetFirewallApplicationFilter 935
Get-NetFirewallInterfaceFilter 935
Get-NetFirewallInterfaceTypeFilter 935
Get-NetFirewallPortFilter 935
Get-NetFirewallProfile 935 f.
Get-NetFirewallRule 935, 937, 939
Get-NetIPInterface 926
Get-NetworkAdapter 654, 866
Get-ODBCDriver 837
Get-ODBCDSN 837
Get-OSVersion 851
Get-Package 887 f.
Get-PackageProvider 886
Get-PackageSource 887
Get-Passagier 1292
Get-PipelineInfo 108, 113, 127, 1274
Get-PnpDevice 867
Get-PnpDeviceProperty 867
Get-PointingDevice 866
Get-PowerShellDataSource 1217
Get-Printer 869 f.
Get-PrintJob 869 f.
Get-Process 34, 46 f., 49 f., 57 f., 64, 90, 95, 105 f., 111, 113, 115, 127, 138, 142, 222, 248, 257, 270, 374, 383, 501, 911, 1273, 1275
Get-Processor 654, 865 f.
Get-PSBreakpoint 506
Get-PSDrive 690
Get-PSProvider 264
Get-PSReadLineKeyHandler 306
Get-PSRepository 631
Get-PSSession 285
Get-PSSessionConfiguration 285, 287, 664
Get-PSSnapIn 620
Get-PswaAuthorizationRule 340
Get-Random 126, 178
Get-ReparsePoint 720
Get-Service 78, 95, 107 f., 111, 127, 150, 257, 270, 374, 376, 393, 917 f., 1241
Get-SHA1 714
Get-ShortPath 702
GetShortTimeString() 107
Get-SmbShare 725, 731
Get-SmbShareAccess 732
Get-SoundDevice 865
Get-SqlData 826
Get-Storagegroup 1079
Get-Tapedrive 865
Get-TargetResource 616
Get-TerminalSession 31
Get-TraceSource 502
Get-Transaction 362, 507, 510
Get-Unique 75, 129
Get-Uptime 395, 851
Get-USB 866
Get-USBController 654, 866
Get-Variable 164, 167, 175
Get-VHD 1108
Get-VideoController 654, 865
Get-VirtualHardDisk 657
Get-VM 1097 f., 1107
Get-VMBIOS-VM 1099
Get-VMBuildScript 1117 f.
Get-VMHost 657, 1097
Get-VMMemory 1099
Get-VMProcessor 1099
Get-VMSnapshot 1112
Get-VMSummary 1118
Get-VMThumbnail 1117 f.
Get-WBBackupSet 738
Get-WBPolicy 738
Get-WBSummary 738
Get-WebApplication 1086
Get-WebitemState 1093
Get-Website 1086, 1091
Get-WebvirtualDirectory 1086
Get-WindowsCapability 902
Get-WindowsEdition 851
Get-WindowsFeature 890, 892, 894
Get-WindowsOptionalFeature 901 f.
Get-WinEvent 270
Get-WmiObject 14, 270, 362, 415, 433, 459, 461, 463, 465, 695, 865, 871, 918, 920, 923, 967, 998
Get-xDscOperation 611
Gigabyte 177
Git 67, 97, 160
git.exe 223
Github 32, 394
Gitternetz 1214
GivenName 1038 f.
Gleichheitszeichen 203
global 172 f., 1214
Global Assembly Cache *siehe* GAC
GlobalSign 515
Global Unique Identifier 1010, 1330
gm 89
Go 1189
Google 1212
GPMC 1067
Grafikkarte 435, 462
Grant-SmbShareAccess 732
Gravis 55, 98, 162, 183
grep 357
Grid 1214 f.
GridView 243
Group 130, 589, 1003
GROUP BY 451, 570
Group-Object 89, 130, 142, 964
Gruppe 1026, 1041, 1055 f.
– Active Directory 1012
– anlegen 1019
– auflisten 1018
– lokal 1075
– Mitglied aufnehmen 1019
Gruppenmitglieder 1041
Gruppenmitgliedschaft 1020

Gruppenrichtlinie 504, 682, 1067ff., 1073
– Vererbung 1073
Gruppierung 130
GUID 695
Gültigkeitsbereich 165, 172
GZIP 722

H

Haltepunkt 35, 317f., 505
Hardlink 718
Hardware 435, 654
Hardwareverwaltung 865
Hash-Tabelle 201f., 408, 477
Hashtable 201, 203, 408, 598f., 616
Hashwert 710
HAVING 451, 570
Heimatordner 174
Help-Info 83
HelpMessage 1230
Herausgeber 516, 519
Here-String 179
Herunterfahren 857
Hexadezimalzahl 177
hidden 235
Hilfe 71
Hilfetext 81
Hintergrundauftrag 521
Hintergrunddatentransfer 958
Hintergrundübertragungsdienst 47
History 677
Hit Refresh 357
HKCU 261
HKEY_CURRENT_USER 844
HKEY_LOCAL_MACHINE 844
HKLM 261
Home 174
HomeDirectory 1039
HomeDrive 1039
Host 175, 320, 342, 678, 1095
Hosting 1309
hostname.exe 857
Hotfix 435
HTML 761f., 1090, 1205
HtmlWebResponseObject 945
HTTP 269, 374, 604
HTTPS 269, 374
Hyper-V 31, 296, 639, 657, 1095, 1115, 1119, 1122f.
– Überblick 1095
Hyper-V-Container *siehe* Docker
Hyper-V-Integrationsdienste 1100
Hypervisor 1095
Hyper-V-Modul
– Überblick 1096

I

i 204
IADs 1000f., 1003
IADsComputer 1003
IADsContainer 1001, 1003
IADsGroup 1003
IADsUser 1003, 1016
idempotent 587f.
Identität 1092
Identity 972
IdentityReference 980
IDL 446
IEnumerable 1002
if 206, 212
IIS 31, 263, 337, 1005, 1122f., 1146
– Internet Information Services 1083
– Nano 1129
IISAdmin 919
IISAdministration 1083, 1129
IIS-Anwendung 1092
IIS Management Service 1129
ILSpy 421
Impersonifizierung 496, 1007
– WMI 448
Implizites Remoting 289
Import-Alias 64
Import-AzAksCredential 1169, 1172
Import-CliXml 493, 678, 758
Import-Counter 969
Import-CSV 142, 493, 698, 747, 1056, 1088
Import-GPO 1069
Import-INIFile 751
Import-Module 551, 624, 630, 636, 1303f.
Import-PSSession 290
Import-VM 1097, 1113f.
IncludeUserName 912
Index 198
Indexer 1333
Informix 787, 837
Ingres 787
Inheritance Flags 973
INI-Datei 751
inlinescript 545f.
Innertext 756
Input 174
Inputbox 1205
InputBox 417, 495, 1205
InputBox() 495, 1205
InputObject 95, 421, 1275, 1277
inquire 53, 229
Install-ADDSDomainController 1062
Install-ADDSForest 1062
Installation 875
Installationsordner 18, 120, 174
Installationstechnologie 875
Install-Module 32, 344, 627, 630f., 668
Install-Package 419
Install-PswaWebApplication 339
installutil.exe 1253, 1258
Install-WindowsFeature 338, 605, 890
InstanceCreationEvent 570f., 573
InstanceDeletionEvent 438, 449, 570
InstanceModificationEvent 449, 570
Instanz 408, 1318
Instanziierung 1319
Instanzmitglied 1334
int 165
Int32 165
Int64 177
INTEGER 1013
Integrated Scripting Environment 34
Integrated Scripting Environment *siehe* ISE
IntelliJ 337
IntelliSense 34, 46, 331
Interface 1320
Interface Definition Language 446
InternalHost 678
InternalHostUserInterface 494
Internet Control Message Protocol 933
Internet Information Server 436
Internet Information Services 435, 437, 604, 611, 903, 919, 1005, 1083
Interpretermodus 306
IntervallTimerInstruction 449
Intrinsic Event 449
InvalidOperationException 1002
Invoke-BPAModel 995
Invoke-CimMethod 362, 459, 477
Invoke-Command 275, 277, 279, 281ff., 293, 296, 505, 522, 1125
Invoke-ContainerImage 1138, 1140
Invoke-DbCommand 807
Invoke-DBMaint 823
Invoke-Expression 205
Invoke-History 677
InvokeMethod() 458
Invoke-Query 826
Invoke-SqlBackup 834
Invoke-SqlCmd 811, 814, 818ff., 831, 834
Invoke-SqlCommand 654
Invoke-Webrequest 946, 958
Invoke-WebRequest 374, 945, 949, 955

Invoke-WmiMethod 362, 459, 476
IP
– Adresse 141, 167, 297, 452, 591, 923, 926, 1163
– Konfiguration 452
ipconfig 67, 97
IPHostEntry 932
IP Routing 436
IRQ 435
Is64BitOperatingSystem 850
Is64BitProcess 850
ISA 451
IsCoreCLR 375
ise 1309
ISE 146, 259, 319, 378, 1125
– Debugging 317, 505
– Integrated Scripting Environment 313
IsePack 652
ISE Steroids 343
IsInRole() 972
IsLinux 375
IsmacOS 375
ISO 1095, 1102, 1105, 1114
IsWindows 375
Item() 1001
Iteration 99

J

Java 1189, 1330
JavaScript 421
JEA 661
Jeffrey Snover 147
Job
– zeitgesteuert 531
Job-Trigger 531
– Zeitgesteuerte Jobs 531
John Doggett 998
Join 185
Join() 188
Join-String 136, 188
JPEG 712
JScript .NET 487
JSON XXVII, 312, 764, 955, 1202
Junction 719
Junction Point 718 f.
Just-In-Time-Compiler 1327

K

Kennwort 496 f., 999, 1016
Kerberos 269, 448
Kill() 105 f.
Kilobyte 177
Klammer 49
– rund 191
Klammeraffe 66
Klasse 164, 410, 438, 442, 483, 1318, 1320 f., 1333
– CIM 437
– COM 404, 428
– Commandlet 1252, 1272, 1294
– .NET 233, 399, 1251, 1330, 1332
– PowerShell 233 f.
– statisch 412
– WMI 435 f.
Klassendiagramm 1322, 1324
Klassenhierarchie 1322
Klassenmitglied 410, 1334
Klassenname 406
Known Host 392
Kommandomodus 306
Kommandozeilenbefehl 67
Kommentar 162
Komponentenorientierung 1327 f.
Komponententest *siehe* Unit Test
Komposition 1217
Komprimierung 721 ff.
Konstante 778
Konstruktor 1318, 1334
Konstruktorfunktion 405, 407, 428
Kontakt 1026
Konvention 1293
Kopieren/Einfügen 303
Kosinus 412
Kreuzzuweisung 205
ksh 357
Kubernetes Command Line Client 1169

L

Label 240
Language Server Protocol *siehe* LSP
LastAccessTime 1322
LastExitCode 175, 221, 223
Laufwerk 261, 263, 268, 690, 844
– virtuell 1108
LDAP 997, 1004 f., 1021
– Suchanfrage 1001, 1021
– Suche 1027
LDAP-Query 1022
Leaf 1011
Least Privilege 661
Leistung 967
Leistungsindikator 967 f.
Lenght 109
Length 109, 483
Limit-EventLog 270, 963, 965
LinearGradientBrush 1215
Linie 1322
LINK 1240
Linux 7, 22, 36 f., 40, 44, 264, 357, 365, 383, 391, 427, 1132, 1136, 1330
– andere Distributionen 41
– Container 1150 f.
– Dateisystem 387
Linux Foundation 357
Literal 251
Little Endian 1021
Lizensierung 659
Load-ContainerImage 1152
LoadFrom() 416
LoadWithPartialName() 416
Logarithmus 412
Log File Event Consumer 451
Logoff 682
Logon 682
Lokalisierung 442, 584
Loopback 272
ls 383
LSP 335

M

Machine.config 787
MachineName 271, 1256
macOS 7, 22, 36, 42, 264, 357, 365, 383, 427, 1330
– Dateisystem 387
MailAddress 943
MailMessage 942
makecert.exe 516
MAML 82, 1287
man 383
Manage-Bde 740
Managed Code 999
Managed Object 434
Managed Object Format 446
Managed Provider 785, 1022
ManagedThreadId 101
ManagementBaseObject 457
ManagementClass 121 f., 167, 457 f., 460, 463, 468, 1264
ManagementEventWatcher 571
Management Infrastructure API 457 f.
ManagementObject 122, 167, 457 f., 460, 463, 467, 469 f., 476, 919, 1264
ManagementObjectCollection 467, 1264
ManagementObjectSearcher 167, 463, 1264
ManagementScope 571
Mandatory 1230, 1268
Manifest 625
Manifestmodul 1297
Markdown XXVII, 761
MarkdownInfo 762
Maschinencode 1327
Match 123
MaximumDriveCount 268
MaximumErrorCount 175

maxSessionsAllowedPerUser 340
md 707, 713, 847
measure 137
Measure-Command 501
Measure-Object 89, 137, 142
Measure-VM 1097
Megabyte 177
Mehrsprachigkeit 584
Menge 109 f.
Mercurial 160
Merge-VHD 1108
Message 225
MessageBox 496, 1203
MessageBoxButtons 1204
MessageBoxDefaultButton 1204
MessageBoxIcon 1204
Metamodell 444
Metaobjekt 1010
Methode 115, 1317, 1322, 1334
– Getter 1333
– Setter 1333
Method *siehe* Methode
Microsoft Access 801
– Treiber 791
Microsoft-Access 18
Microsoft Access Driver 840
Microsoft.ACE.OLEDB 791
Microsoft Azure 1155
Microsoft Command Shell 4
Microsoft Developer Network 86, 403
Microsoft Developer Network *siehe* MSDN
Microsoft Excel 1056
Microsoft Exchange 1061
Microsoft Exchange Server 591, 656, 1079
Microsoft.GroupPolicy 1068
Microsoft.Jet.OLEDB 791
Microsoft.Management.Infrastructure.CimClass 469
Microsoft.Management.Infrastructure.CimClassProperties 469
Microsoft.Management.Infrastructure.CimInstance 469
Microsoft.Management.Infrastructure.CimInstanceProperties 469
Microsoft.Management.Infrastructure.CimProperty 469
Microsoft Office XXVII, 437, *777*
Microsoft Outlook 916
Microsoft Print Ticket XML 869
Microsoft Shell 4
Microsoft SQL Server 1163
Microsoft SQL Server *siehe* SQL Server
Microsoft System Center 454
Microsoft.Update 858
Microsoft.Vhd.PowerShell 1108
Microsoft.VisualBasic 495, 1205
Microsoft.VisualBasic.Interaction 431, 495, 1205
Microsoft.Win32 874
Microsoft.Win32.RegistryKey 843
Microsoft Word 431
Minute 104
Mitglied
– .NET 1333
– statisch 1334
– WMI 472
MMC 448
Mock-Objekt 672
Modul 623, 632
Module Browser 629
Modulo 203
MOF 446, 597, 599
Monad 4
Monica Reyes 998
Moniker 1005
Mono 1330
Month 104
more 220, 249
Mount-SpecialFolder 691
Mount-VHD 1108
move 703
Move-ADObject 1041 f.
Move-Item 689, 703, 713
Move-Mailbox 1080
Move-VM 1097
MSCL 13
mscorlib.dll 1332
MSDN 203, 403, 1210
MSDN Library 86, 403
MSFT_Printer 869
MSFT_PrintJob 869
MSFT_SmbShare 731
MSFT_SmbShareAccessControlEntry 732
MSFT_WUOperationsSession 858, 861, 1127
MSFT_WUSettings 858
MSFT_WUUpdate 858
MSH *siehe* Microsoft Shell
MSI 613, 871, 873, 875, 883, 1253
MTA 1209
Multithreading 100 f., 1329
MySQL 787, 798, 811, 823, 1138
MySqlConnection 798
MySqlLib 811

N

Nachkommastelle 177, 412
Name 95, 1039
Namensauflösung 932
Namensraum 439, 441 ff., 975, 1063, 1330
– ADSI 1005
– .NET 1330, 1332
– WMI 441, 443
Namensraumhierarchie 1331
NamespaceCreationEvent 570
NamespaceDeletionEvent 570
Namespace-ID 1005
NamespaceModificationEvent 570
Nano Server 12, 355, 1119, 1122
– IIS 1129
– Installation 1123
– Paketinstallation 1123, 1127
NanoWbem 433
NativeObject 1002, 1004
Navigation 261
Navigation Provider 262
Navigationsbefehl 265
Navigationsmodell 689
Navigationsparadigma 261
Navigationsprovider 997
Negation 205, 425
NetAdapter 925 f.
NetSecurity 935
NetSecurity-Modul
– Überblick 934
Netsh 928, 930, 938 f.
Netsh
– Per PowerShell aufrufen 930
netstat 67
NetTCPIP 925 f.
NetworkInterface 1262
Network Load Balancing 436
Netzlaufwerk 435
Netzlaufwerksverbindung 436
Netzwerkadapter 1095
Netzwerkcenter 274
Netzwerkkarte 435, 452, 923, 1318
– Geschwindigkeit 928
Netzwerkkartenprofil 941
Netzwerkkonfiguration 659, 923
Netzwerkmanagement 433
Netzwerkprofil
– Per PowerShell setzen 941
Netzwerkverbindung 436, 591, 925 f.
Neustart 297, 896
Neustarten 857
new() 405, 407, 428
New-ADGroup 1041, 1059
New-ADObject 1042
New-ADOrganizationalUnit 1041, 1045
New-ADUser 1041, 1046, 1049
New-AppLockerPolicy 906, 909
New-AzAks 1169, 1171
New-AzSqlDatabase 1163
New-AzSqlServer 1163
New-AzSqlServerFirewallRule 1163
New-AzWebApp 1162
New-Buchung 1292

New-Button 77, 1213, 1215
New-CheckBox 1217
New-CimInstance 459, 477
New-CimSession 462
New-CimSessionOption 462
New-ComboBox 1217
New-Container 1148
New-DSCChecksum 610
New-Elippse 1217
New-Event 578
New-EventLog 270, 963, 965
New-FileCatalog 710
New-Fixture 670
New-GPLink 1069
New-GPO 1068
New-GPStarterGPO 1068
New-Grid 1215
New-Guid 405
New-Hardlink 719
New-HardwareProfile 657
New-IISSite 1083, 1129
New-Image 1217
New-Int64Animation 1217
New-Item 262, 689, 716, 744, 843 f., 847
New-Itemproperty 845, 847
New-JobTrigger 532, 534
New-Junction 720 f.
New-Label 1210, 1215
New Line 183
New-Line 1217
New-ListBox 1217
New-LocalUser 26
New-Mailbox 1080
New-Mailboxdatabase 1080
New-MediaElement 1217
New-Menu 1217
New-Module 623
New-NanoServerImage 1123 f.
New-NetFirewallRule 935, 938
New-NetIPAddress 926 f.
New-Object 404 ff., 413, 415, 428, 777, 1235, 1334
New-PasswordBox 1215
New-ProgressBar 1217
New-PSDrive 268, 844, 873, 1036
New-PSSession 273, 275, 285, 288 f., 293, 296, 664
New-PSSessionConfigurationFile 661
New-RadioButton 1217
New-Rectangle 1217
New-RichTextBox 1217
New-ScheduledJobOption 535
New-ScrollBar 1217
New-SelfSignedCertificate 516
New-Service 917, 921
New-Shortcut 717
New-Slider 1217
New-SmbShare 53, 730 f.
New-StatusBar 1217
New-Storagegroup 1080
New-Storyboard 1217
New-TextBlock 1217
New-TextBox 1215
New-TimeSpan 196
New-TreeView 1217
New-UrlShortcut 718
New-VHD 1104, 1108 f.
New-ViewBox 1217
New-VirtualDVDDrive 657
New-VirtualNetworkAdapter 657
New-VM 657, 1097, 1102, 1118
New-VMSwitch 1097
New-WebApplication 1093
New-WebAppPool 1091
New-WebServiceProxy 953 ff.
New-Website 1083, 1087, 1129
New-WebVirtualDirectory 1092
New-Window 1215
New-Zip 722
NoAccess 731
Node 756
node.js 1138
nodeJS 1189
NoElement 131
Non-Terminating Error 224, 1281, 1284
NoProfile 514
NormalView 220
Northwind 814, 835
Notation 1322
– umgekehrt polnische 1021
Notepad 146
NoteProperty 751
Notes 1246
NOTES 1240
Notizeigenschaft 115, 119, 128
Novell 1005
Now 412
Nslookup 928, 932
NtAccount 980
NTAccount 976, 979 f.
NT Event Log Event Consumer 451
NTFS 445
NTLM 269
NTSecurityDescriptor 1012
Nuget 402, 419
NuGet 44, 1189
NuGet Package Manager 333
null 138, 150, 169, 211, 215, 256
Null Assignment Operator 169
Null Coalescing Assignment 169
Null Coalescing Operator 169
Null Conditional Operator 169 f., 377
Null-Wert 104

O

Object 1323
Object[] 109, 218
ObjectCategory 1012, 1025, 1044
ObjectClass 1012, 1025, 1038 f., 1044
ObjectGUID 1013, 1039
ObjectiveC 1189
Object Linking and Embedding Database 796
ObjectSecurity 975
ObjectSecurityDescriptor 1012
ObjectSid 1012
ObjectVersion 1013
Objekt 196, 749, 1294, 1317, 1320 f.
– Dynamisch 483, 1221
– .NET 1207
– WMI 436
Objektadapter 122, 467 f., 788
Objektassoziation
– WMI 443
Objektbaum 1321
Objektidentifikation
– ADSI 1005 f.
Objektmenge 698
Objektorientierte Programmierung *siehe* OOP
Objektorientierung 91
Objektorientierung *siehe* OO
Objekt-Pipeline 698
Objekttyp 1318
OData 604
ODBC 785 f., 837, 840
Office 1039
OFS 174
ogv 237
OK 1204
OKCancel 1204
OLEDB 785 f., 796, 1002
– Provider 801, 1002, 1022
OleDbCommand 794, 801
OleDbConnection 792, 796, 801
OleDbDataAdapter 801
OMI 433, 461
OMI *siehe* Open Management Infrastructure
On_Click 1215
OneGet 885
OneLevel 1044
ONELEVEL 1021
OO 1317, 1327 f.
OOP 1317
OOP *siehe* COP
Open() 792
Open Database Connectivity
– Einstellung 436
Open Management Infrastructure 389
– OMI 461

Open Source 5
OpenSSH 389
OpenView 454
Operator 124, 188, 198, 202
Optimize-VHD 1108
Oracle 786, 798
OracleCommand 794
OracleConnection 792
Ordner 47, 69, 265, 695, 702 f., 706
– Dateisystem 436
Ordnerstatistik 696
Organisationseinheit 1026, 1055 f.
– anlegen 1020
OSS
– Open Source 5
OutBuffer 53, 94
Out-Default 237, 245, 1312, 1314
Out-File 237, 257
Out-GridView 78, 237 f., 243, 245
Out-Host 237, 248 f.
Outlook 130, 639, 658, 778
Outlook.Application 777
Out-Null 237 f., 256, 417
Out-Printer 237, 257, 868
OUTPUTS 1240
Out-Speech 237, 259, 630
Out-SqlScript 823
OutVariable 53, 140
ov *siehe* OutVariable

P

Packaged Script 333
PackageManagement 626
Page File 659
Paketinstallation
– Nano Server 1127
Panel 1214
PaperSize 869
Papierkorb 711
parallel 548
Parallelisierung 100 f.
Parallelität 101
Parameter 47, 96, 218, 1230, 1246, 1268, 1277
– Abkürzung 50 f.
– Skript 149
PARAMETER 1240
Parameterliste 150
ParentSession 286
ParsedHtml 946
parsen 748
Partition 1041
PascalCasing 1332
PASH 4
PassThru 140, 1041
passwd 385
Pause 869
PE 700
PercentComplete 258
Perforce 160
Performance Counter Provider 967
Performance Monitor 435, 445
PERL 161
Persistenz 550
Pester XXVII, 667 f., 1307
Petabyte 177
Pfad 442
– ADSI 1005
– Verzeichnisdienst 1005
– WMI 439, 441 f.
Pfadangabe 265
Pfeilspitze 1322
Pflichtparameter 23
PHP 161, 1189
PhysicalDeliveryOfficeName 1014, 1039
PIN 740
Ping 67, 436, 933 f., 1281
Ping-Host 31, 33, 933
Ping-VM 1118
Pipe 91
Pipeline 3, 13, 90 f., 111, 137, 256, 358, 472, 1189, 1202
– Ausgabe 1262
– Eingabe 1270
Pipeline Processor 92, 1252
PipelineVariable 53, 141, 248
Pipelining 89, 202, 261
Plattform Invoke 489
Plattformunabhängigkeit 1327
Platzhalter 49, 251
Plug-and-Play 867
Polymorphismus 1324
Port 386
Portable-Executable-Format *siehe* PE
PoshConsole 347
Position 1230, 1268
Postfach 1080
Postfix-Notation 1021
PowerGUI 146, 313
Power Management 445
PowerShell 3, 58, 89
– Extension 654
– Hosting 3
– Konsole 301 f.
– Laufwerk 261 f., 268, 844
– Remoting 353
– Skriptsprache 161
– Version 1.0 4
– Version 2.0 4
– Version 3.0 4
– Version 4.0 4
– Version 5.0 5
– Version 5.1 5
– Web Admin 666
PowerShell Analyzer 330
PowerShell Community Extensions 31, 648, 997
PowerShell Community Extensions *siehe* PSCX
PowerShell Core 7, 14, 37
– Funktionsumfang 359
– installieren 36
– Konsole 378
– Module 630
– Version 5.1 12, 355, 1122
– version 6.x 7
– Version 6.x 44, 357
– WMI 433
PowerShell Direct 295, 297, 1125
PowerShell Editor Services 335, 337
powershell.exe 57, 637, 680, 1209
powerShellExePath 380 f.
PowerShell Gallery 30, 32, 589, 626
PowerShellGet 626
PowerShell Management Library for Hyper-V 1096, 1116
PowerShellPlus 146, 313, 344, 1275
PowerShell Remoting
– Port 295
PowerShell-Remoting 337
PowerShell Remoting Protocol 269, 271, 389
PowerShell Remoting *siehe* Remoting
PowerShell Script Analyzer 325, 335
PowerShell Web Access *siehe* PSWA
PowerStudio 332
PrimalScript 349
Principal 976
Printer 237
Printing 869
PrintManagement 869
Print Ticket XML 869
Private 935
Privileg 448
Process 95, 111, 128, 140, 589, 714, 1275
ProcessRecord() 1252, 1256, 1268
Professional Developer Conference 4
profile 511
ProfilePath 1039
profile.ps1 417, 1223, 1277
Profilskript 309, 511, 514
Programmcodeanalyse 325
Programmgruppe 435
Programmiersprache 212
Programmiersprachenunabhängigkeit 1327
Prompt 679

PromptForChoice() 494
Propagation Flags 973
Property 116, 248, 1317, 1333
Property Cache 1009
PropertyCollection 1001
PropertyDataCollection 458, 468 f.
PropertyGrid 1207
PropertyNames 1001
PropertyValueCollection 1001, 1008
ProtectedFromAccidentalDeletion 1039, 1043
Protokolldatei 451
Protokollierung 503
Provider 264
– ADO.NET 785
– Dateisystem 689
– PowerShell 262
– Verzeichnisdienst 997
– WMI 437
Proxy 953
ProxyCommand 1241
Proxy-Commandlet 289, 1241
Prozedur 215
Prozess 46 f., 142, 435, 444
– auflisten 257, 911
– beenden 915
ps 383
PSAzureProfile 1160
PSBase 1004, 1010 f.
PSCmdlet 1251
PSCodeGen 653
PSComputerName 283
PSConfig 639
PSCredential 496, 914, 1060
PSCustomobject 1235
PSCustomObject 109, 128, 485, 746, 751, 1232, 1235
PSCX 33, 237, 498, 628, 630, 639, 648, 716, 722 f.
PSCX *siehe* PowerShell Community Extensions
psd1 583
PSDriveInfo 690
PSDSCRunAsCredential 592
psedit 315, 323
PSEdition 355
PSHome 174
PSHost 175, 1311
PSHostRawUserInterface 1311, 1313
PSHostUserInterface 1311, 1313
PSImageTools 653
psISE 320
PSItem 90, 486
PSModuleAutoLoadingPreference 175, 635
PSModulePath 373, 383, 624, 1298
PSObject 121, 1314 f.
PSReadline 304 f., 378
PSRemotingJob 522
PSRP *siehe* PowerShell Remoting Protocol
PSRSS 653
PSScheduledJob 527
PSScriptRoot 151
PSSession 284
PSSnapIn 1254
PSSystemTools 653, 690, 866 f.
psUnsupportedConsoleApplications 319
PSUserTools 653
PSVariable 164
psversiontable 39, 355
PSWA 337, 339
Public 935, 1268
Public Network 274
Pull-ContainerImage 1138
Pull Request 394
Punktnotation 103, 408, 472
Push-ContainerImage 1154
Put() 473 f.
pv *siehe* PipelineVariable
pwsh 38, 378
pwsh.exe 308
Python 161, 1189

Q

Quantifizierer 191
Quantor 191
QueryDialect 466
Quest 639, 654

R

Range 185
RawUI 320
RDP 298, 941
ReadAccess 731
Read-Host 493, 498, 542, 544
Receive-Job 521, 523 f., 541
Rechenleistung 142
Recovery Console 1120
recurse 48, 696, 1041
recursive 1041, 1046
Redirection *siehe* Umleitung
Redo 306
Redstone 5
REFERENCES OF 451
Referenzkopie 203 f.
Refresh() 1322
RefreshCache() 1009
RefreshFrequencyMins 609
Regel 906, 938
region 324
Region 324
Register-CimIndicationEvent 459, 577
Register-DnsClient 929
Register-Event 575
Register-ObjectEvent 735
Register-Packagesource 631, 886
Register-PSSessionConfiguration 285, 288, 661
Register-ScheduledJob 533 f.
Register-WmiEvent 572, 577
Registrierungsdatenbank 3, 261, 267 f., 436, 602, 843, 873
– Schlüssel 843
Registry 443, 445, 589, 843
RegistryKey 874, 975
RegistrySecurity 977
RegistryValueChangeEvent 449, 570
Regulärer Ausdruck 166, 189, 710
Relative Distinguished Name 1010 f.
Remote Desktop Protocol *siehe* RDP
Remote Procedure Call *siehe* RPC
Remote Server Administration Tools 997, 1033
Remoting 149, 269, 298, 391, 869, 1125
– Implizit 289, 1241
Remove-ADGroup 1059
Remove-ADGroupMember 1041, 1059
Remove-ADObject 1041 ff.
Remove-ADOrganizationalUnit 1045
Remove-ADUser 54, 1046
Remove-Alias 64
Remove-AzAks 1169, 1188
Remove-AzSqlDatabase 1164
Remove-AzSqlServer 1164
Remove-AzWebApp 1163
Remove-Buchung 1292
Remove-CimInstance 459, 478
Remove-Computer 857
Remove-Container 1148
Remove-ContainerImage 1149, 1153
Remove_DirectoryEntry 1285
Remove-DirectoryEntry 654, 1275
Remove-Event 575
Remove-EventLog 270, 963
Remove-GPLink 1070
Remove-GPO 1068
Remove-GPPrefRegistryValue 1074
Remove-GPRegistryValue 1074
Remove-Item 47, 52, 54, 64, 689, 703, 844, 847
Remove-ItemProperty 846
Remove-Job 521, 524
Remove-JobTrigger 532
Remove-LDAPObject 1029, 1226

Remove-LocalUser 1075
Remove-Module 624, 638
Remove-NetFirewallRule 935, 939
Remove-NetFirewallRule-Funktion 939
Remove-NetIPAddress 926
Remove-NetRoute 926
Remove-ODBCDsn 837
Remove-PrintJob 870
Remove-PSBreakpoint 506
Remove-PSSession 285 f.
Remove-PSSnapin 362
Remove-PswaAuthorizationRule 340
Remove-Service 922
Remove-SmbShare 52, 54, 731
Remove-Variable 173 f.
Remove-VM 1097, 1107
Remove-VMSnapshot 1112
Remove-WebApplication 1094
Remove-WebAppPool 1094
Remove-Website 1094
Remove-WebVirtualDirectory 1094
Remove-WindowsCapability 901
Remove-WindowsFeature 890, 896
Remove-WmiObject 459, 478
Rename-ADObject 1041 f.
Rename-Computer 857
Rename-Drive 694
Rename-GPO 1068
Rename-Item 703
Rename-NetAdapter 928
Rename-NetFirewallRule 935
Rename-VM 1097
Rename-VMSnapshot 1112
Repair-VM 1097
Replace 187
Replikation 435
Repo 1189
Repository 444
requires 158
Resize-VHD 1097, 1108
Resolve-Assembly 417
Resolve-DnsName 932
Resolve-DNSName-Funktion
– Beispiel 932
Resolve-DsnName 929
Resolve-Host 932
Resolve-Path 266
ResponseHeaders 409
Ressource 589
Ressource Group 1155
REST 955
Restart-Computer 270, 857 f., 896
Restart-PrintJob 869
Restart-Service 54, 292, 295, 917, 920
Restart-VM 1097
Restore-ADObject 1042
Restore-Computer 862
Restore-DscConfiguration 601
Restore-GPO 1069
Restore-VMSnapshot 1112
Restricted 152
Restricted Runspace 661
Resume-PrintJob 869
Resume-Service 917, 920
Resume-VM 1097
return 206, 267, 1252
Revoke-SmbShareAccess 732
Richtlinienergebnisbericht 1071
Robocopy 707 ff.
Rolle 890, 1122
Rollendienst 890
rootcimv2 443
RPC 270
RSAT 31, 1096
RSS 653, 944
Ruby on Rails 1138
Rückgabeobjekt 1262
RuleCollection 907
Run-ContainerImage 1139 f., 1143 f., 1146 f.
RunNow 533
Runspace 330, 661
RuntimeException 233

S

s 362
sa 822
SAM 1005
SAMAccountName 1012, 1016, 1021, 1039
Sapien 349, 351 f., 659 f.
SAPI.SPVoice 259, 428 f.
Satya Nadella 357
Save-ContainerImage 1152
Save-Help 83
Save-Module 627
Save-VM 1097
Schablone 1318 f.
Schalter 48, 1295
Schattenkopie 736
ScheduledJob 533
Scheduled Task 527
Schema 1010, 1037
– Active Directory 1015
– WMI 443
Schemaabfrage 452
SchemaNameCollection 1002
SchemaNamingContext 1035
Schleife 99, 210
Schlüssel 261
Schlüsselattribut
– WMI 438
Schnittstelle 234, 789, 1320, 1324
– .NET 1334
Schriftart 856
Schtasks.exe 528
Scope *siehe* Gültigkeitsbereich
script 172
Script 603
Script Analyzer 325
Scripting.FileSystemObject 430
ScriptPaneBackgroundColor 321
SDDL 664, 725, 987
sealed 1323
Searcher 858
SearchScope 1044
Secure Socket Layer 946
Secure String 739 f.
Security Descriptor 972
Security Descriptor Definition Language 287
Security Identifier 972, 976 f., 979 f.
Security Service Provider 448
Select
– PowerShell 127
SELECT 451, 569
– WQL 451 f., 454
SelectNodes() 754 f.
Select-Object 23, 89 f., 102, 118, 122 f., 127, 129, 131, 142, 248, 471, 755, 1277
SelectSingleNode() 754 f.
Select-String 67, 97, 710, 744
Select-Xml 754 f.
Semantic Versioning 167
Semaphore 975
Semikolon 142, 913
Send-MailMessage 942 f.
Send-SmtpMail 942
sequence 545
Serialisierung 112, 278
Seriennummer 853
Server 1041
ServerCertificateValidationCallback 955
Server Management Objects *siehe* SMO
Server Manager 1060
ServerRemoteHost 342
ServerURL 609
Service 589
ServiceController 111, 278, 917
Serviceorientierung 1328
Serviceorientierung *siehe* SOA
ServicePointManager 946
Session 431, 858
sessionState 340
Set-Acl 971, 984, 987
Set-ADAccountPassword 1041
Set-ADGroup 1059
Set-ADObject 1042 f.

Set-ADOrganizationalUnit 1045
Set-ADUser 1046, 1048
Set-Alias 63
Set-AppLockerPolicy 906
Set-AuthenticodeSignature 517
Set-AzAks 1169
Set-AZContext 1160
Set-AzWebApp 1162
Set-BPAResult 995
Set-CimInstance 459, 474
Set-Clipboard 498
Set-Content 689, 744, 774, 944
Set-DataRow 808
Set-DataTable 654, 808
Set-Date 196
Set-DistributionGroup 1080
Set-DnsClientServerAddress 926 ff.
Set-ExecutionPolicy 27 f., 145, 152, 515, 519
Set-FileTime 700
Set-FirewallProfile 936
Set-GPInheritance 1073
Set-GPLink 1070
Set-GPPermissions 1074
Set-GPPrefRegistryValue 1074
Set-GPRegistryValue 1074
SetInfo() 1000, 1009 f.
Set-Item 292, 689
Set-ItemProperty 699, 846, 941
Set-JobTrigger 532
Set-Location 57, 261, 689, 843
Set-Mailbox 1080
Set-MarkdownOption 761
Set-NetFirewallPortFilter 935
Set-NetFirewallProfile 935
Set-NetFirewallRule 935, 939
Set-NetIPInterface 926
Set-ODBCDriver 837
Set-ODBCDsn 837
Set-PrintConfiguration 869
Set-PSBreakpoint 505 f.
Set-PSDebug 168, 499 f.
Set-PSReadLineKeyHandler 306
Set-PSReadlineOption 29 f., 305, 378
Set-PSSessionConfiguration 285
Set-ScheduledJob 533
Set-Service 376, 917, 920 f.
Set-StrictMode 168
Set-TargetResource 616
Setter 116 f., 1333
Set-TraceSource 503
Set-Variable 164, 175, 1213 f.
Set-VHD 1108
Set-VM 1097, 1099
Set-VMMemory 1118
Set-VolumeLabel 694
Set-VolumneLabel 31
Set-WmiInstance 459, 474
Set-WSManQuickConfig 274
SHA256 710
Shell 3, 90
Shell.Application 711, 723
Shielded VM 1123
ShouldProcess() 1285 f.
Show() 1203
Show-Command 78 f., 316
ShowDialog() 1218
Show-EventLog 270, 964
Show-HyperVMenu 1117
Show-Markdown 761
Show-NetFirewallRule 935
Show-Service 270
Show-VMMenu 1117
ShowWindow 80
Shutdown 682
Sicherheit
– COM 448
– Dateisystem 436, 445
– PowerShell 152
– WMI 448
Sicherheitsabfrage 1236, 1285
Sicherheitsbeschreibung 972
Sicherheitseinstellung 971
Sicherheitsmodell 3
Sicherheitsrichtlinie 153
SID 972
Side-by-Side Executing 1329
Signatur
– digital 515
Signieren 516
SilentlyContinue 53, 229, 707, 1048
Simple Network Management 434 f., 445
Simple Object Access Protocol *siehe* SOAP
Sitzung 284 f., 287
Sitzungskonfiguration 287, 661, 664
Skip 102
SkipEditionCheck 368
SkipNetworkProfileCheck 275, 941
Skript 145, 147
– PowerShell 145
Skriptausführungsrechte 27
Skriptausführungsrichtlinie 27
Skriptblock 172, 277, 1213
Skriptdatei 145
Skripteigenschaft 115, 119
Skriptfenster 34
Skriptmodul 1297
Skriptsprache 1221
SMB 1122
SMO 814, 820 f., 831, 833, 835
Smoking Man 998
SMTP 942 f.
SmtpClient 942
Snap-In 362, 617, 623, 637, 1253 f., 1277, 1288
Snapshot
– Hyper-V 1111
SNA Server 445
Snippet 316
SOA 1327
SOAP 269, 445, 953
Software 435, 659
– deinstallieren 875
– installieren 613, 615, 875
– inventarisieren 871
– verwalten 871
Softwareentwickler 403
Softwareentwicklungsplattform 1328
Softwarekomponente 401, 415, 419
Softwarepaket 887
Softwarequelle 887
Software Restriction Policy 905
Sortieren 128
Sort-Object 89 ff., 93, 122, 128 f., 131, 136, 142 f., 217, 1252
Speak() 429
Speech 237
SpeechSynthesizer 259
Speicher 109
Speicherbereinigung 1329
Speicherverbrauch 790
Speicherverwaltung 1329
Spitzname 1320
Spoolerdienst 869
Spooling 869
Sprachausgabe 237, 259, 429
Sprache 584
Sprachkürzel 584
SQL 451, 840
SQLASCOMMANDLETS 813
Sqlcmd.exe 818
SqlCommand 794, 818
SqlConnection 408, 792, 796, 798
SqlDataSourceEnumerator 788
SQLPS 263, 811, 814, 818
SQLPSX 811, 814, 823 f., 831
SQL Server 357, 786, 811, 813
– Agent 831
– Laufwerk 815
SqlServerCe 786
SqlServerCmdletSnapin100 813
SQL Server Management Studio 815, 831
SSH 391
sshs 391
SSL 339, 955
SSL *siehe* Secure Socket Layer
STA 1209
StackPanel 1214
StackTrace 174

Stammzertifizierungsstelle 516, 519
Standarddrucker 257
Standardkonsole 301
Start-BITSTransfer 959
Start-Container 1148
Start-ContainerProcess 1144
Start-DscConfiguration 594
Start-Job 521 ff.
Startmenü 435
Start-Process 527, 911 f., 914
Start-PSSession 525
Start-Service 279, 917, 920
Start-Sleep 158
Start-Transaction 507 ff.
Start-Transcript 542
Startup 682
Start-VM 1097, 1105, 1118
Start-WBBackup 738
Start-WBFileRecovery 738
Start-WBHyperVRecovery 738
Start-WBSystemStateRecovery 738
Start-WBVolumeRecovery 738
Start-Webitem 1093
Start-Website 1093
static 234
Status 258, 869
Stop 53, 229
Stop-Computer 270, 857
Stop-Container 1148
Stop-Job 521, 524
Stop-Process 105, 543, 911, 915, 1275
StopProcessing() 1252
Stop-Service 47, 917, 920
Stop-VM 1097
Stopwatch 395
Stop-WBJob 738
Stop-Webitem 1093
Stop-Website 1093
Stored Procedure 800
Streaming 92
StreetAddress 1039
String 179, 187
Subnetzmaske 926
Substring() 185
SubTree 1044
SUBTREE 1021
Subversion 160
Suche
– Active Directory 1021
– Assembly 700
– LDAP 1002
– Verzeichniseintrag 1011
– XML 754
SupportsShouldProcess 1285
Surname 1039
Suspend 52
Suspend-PrintJob 869
Suspend-Service 917, 920
Suspend-VM 1097
Switch 48, 206, 212, 1095
SwitchParameter 1295
Sybase 787
Symbolic Link 718, 720
SymLink 720 f.
Synopsis 1246
SYNOPSIS 1240
Syntaxfarbhervorhebung 335
System 1331 f.
System32 136, 142 f.
System ACL 976
System.ApplicationException 233
Systemattribut
– WMI 438
System.Boolean 265
System Center Virtual Machine Manager 657
System.Collections.Hashtable 201
System.Console 413, 677
System.Data 405
System.Data.Odbc 786, 840
System.Data.OleDb 786, 793
System.Data.OLEDB 786
System.Data.OracleClient 786, 793
System.Data.SqlClient 786, 793, 818
System.Data.SqlClient.SqlConnection 408
System.Data.SqlServerCe 786
System.DateTime 195, 407, 409, 412
System.DbNull 797
System.Diagnostics.EventLog 963
System.Diagnostics.Process 92, 105, 245, 911, 1273
Systemdienst 94, 435, 917
– auflisten 452
– überwachen 454
System.DirectoryService 405, 997 ff., 1001, 1003, 1007, 1012, 1015, 1022, 1063, 1077
System.DirectoryServices.ActiveDirectory 1063
System.Directoryservices.DirectoryEntry 407, 409
System.dll 1332
System-DSN 839
Systemende 682
System.Enum 423
System.Environment 276, 850 f., 853, 855, 971, 1256 f., 1262
SystemEvent 570
System.Globalization.CultureInfo 679
System.Int32 165, 177
System.IO.Compression 723
System.IO.Directory 975
System.IO.DirectoryInfo 483, 695 f.
System.IO.DriveInfo 411, 413, 415, 422, 690, 692
System.IO.DriveType 422
System.IO.File 975
System.IO.FileInfo 483, 695 f., 711
Systemklassen
– WMI 437
Systemmanagement 433
System.Management 405, 1264
System.Management.Automation 86, 1251, 1254, 1264, 1314
System.Management.Automation.Cmdlet 1252
System.Management.Automation.PathInfo 266
System.Management.Automation.PSCustomObject 746
System.Management.Automation.PSDriveInfo 690
System.Management.ManagementObject 196
System Management Server 445
System.Math 412
System.Media.SoundPlayer 410
System.Net.Mail 942 f.
System.Net.WebClient 409, 943 f., 946, 955
System.Object 111 f., 485, 1087, 1273, 1277
SystemParametersInfo 489
System.Random 178, 407
System.Reflection 416
System.Security 976
System.Security.AccessControl 975
System.ServiceProcess.ServiceController 917, 919, 1273
Systemstart 682
System.String 179, 184, 385, 1256
System.TimeSpan 196
System.Type 110, 174, 255
System.ValueType 203
Systemwiederherstellung 862
System.Windows 1209
System.Windows.FontStyle 1212
System.Windows.Forms 712, 1203
System.Xml.Node 756
Sysvol 682

T

Tab Completion 303
Tabellenformatierung 240
TabPanel 1214
Tabulator 183
Tabulatorvervollständigung 303

Tag-ContainerImage 1153
TAR 722
TaskScheduler 652
TCP/IP 926
tcsh 357
Team Foundation Server *siehe* TFS
Tee-Object 139
Teilmenge 126
Telnet 276
Terminal Services 436
Terminating Error 224, 1281
Terrabyte 177
Test-32Bit 866
Test-64Bit 866
Test-AppLockerPolicy 906, 908
Test-Assembly 700
Test-Connection 33, 96, 933 f.
Test-CustomerID 1237
Test-DbConnection 807
Test-DscConfiguration 595
Test-FileCatalog 85, 710
Test-IsAdmin 971
Test-JSON 764, 772
Test-ModuleManifest 625
Test-Path 265
Test Plan 1189
Test-PswaAuthorizationRule 340
Test-ServiceHealth 1079
Test-SqlScript 823
Test-TargetResource 616
Test-UserGroupMembership 1020
Test-VHD 1097, 1108
Test-Xml 753
Textanzeige 1217
Textdatei 142, 743
Texteingabefeld 495, 1205
TextInfo 186
Textpad 146
TFS 160, 1189
Thawte 515
this 486, 1214
Thread 101, 548
Thread-Modell 1209
ThrottleLimit 102, 284
throw 206, 233
ThrowTerminatingError() 1281
Thumbprint 989
TIFF 712
TimeSpan 501
Tivoli 454
TLS 955
TLS *siehe* Transport Layer Security
ToLower() 186
Ton 413
ToString() 111 ff., 369, 415, 1273, 1323
TotalProcessorTime 255
ToTitleCase() 186
ToUpper() 186
TPM 739
Trace-Command 685
Tracing 502
Transaktion 362, 507
Transformation 1217
Translate() 980
Transport Layer Security 946
trap 206
Trap 221, 224, 227
Treiber 659
– ODBC 838
Trigger 531
Troubleshooting Pack 991
true 166, 174, 866
Trusted Host 292
Trusted Platform Module *siehe* TPM
Trustee 972
Try...Catch 1048
Try-Catch-Finally 221, 227
T-SQL 818 f.
Tuva 1119
Typ 165, 399, 1318, 1332
– Namensgebung 1332
Typadapter 165, 199
Typbezeichner 165
Type Cast *siehe* Typkonvertierung
types.ps1xml 64 f., 120, 122
Typisierung 164
Typkennzeichner *siehe* Typbezeichner
Typkonvertierung 129, 171
Typname *siehe* Typbezeichner

U

UAC 27, 149, 307, 913
Überladung 219
Ubuntu 40
ufw 386
Umgebungsvariable 435
– Linux 383
Umlaut 745
Umleitung 258
Unblock-File 155
Undefined 152
Undo 306
Undo-Transaction 362, 507, 509 f.
UniformGrid 1214
Uninstall-Package 888
Uninstall-WindowsFeature 890
Unit Test 667
Universal Coordinated Time 440, 475
Unix 3 f., 90 f., 161, 689, 718
Unlock-BitLocker 742
Unregister-PSSessionConfiguration 285, 289, 664
Unrestricted 152
Unterbrechungsfreie Stromversorgung *siehe* USV
Unternamensraum 1330
Unterordner 113, 696
Unterroutine 214
Unterschlüssel 262
until 206
Update 435, 659
– Einstellungen 861
– installieren 860
– suchen 858
UpdateColl 858
Update-Help 83
Update-Module 668
UseBasicParsing 946
UsePropertyCache 1009
User 589, 1003, 1012, 1025
user32.dll 489
User Account Control *siehe* Benutzerkontensteuerung
UserDomainName 971
UserName 971
User Settings 380
UseTestCertificate 339
UseTransaction 508
using 545
USV 867
UTF8 745
UWP XXVII, 883

V

ValidateCount 1230
Validate-CustomerID 1237
ValidateLength 176, 1231, 1295
ValidateNotNull 1230, 1295
ValidatePattern 176, 1231, 1295
ValidateRange 176, 1231
ValidateScript 176, 1231
ValidateSet 176
ValueFromPipeline 1230, 1270, 1272, 1277
ValueFromPipelineByPropertyName 1230, 1272
ValuesCollection 1001
ValueType 203
Variable 36, 110, 139, 163, 174, 251, 261
– eingebaut 174, 375
– vordefiniert 174, 375
– Workflow 547
Variablenauflösung 180 f., 251
Variablenkennzeichner 139, 163
Variablentypisierung 164
VB 487, 489
VBA 777
Verbindungszeichenfolge 408, 792, 800
Verbose 53 f., 595, 1280

VerbosePreference 54, 175, 1280
VerbsCommon 1293
VerbsCommunications 1294
VerbsData 1294
VerbsDiagnostic 1294
VerbsLifeCycle 1294
VerbsSecurity 1294
Vererbung 443, 1322, 1334
Vererbungsdiagramm 1322
Vererbungshierarchie 456, 1037, 1322
– WMI 443
Vergleich 141
Vergleichsoperator 123
Verifikation 1329
VeriSign 515
Verknüpfung 717
Verzeichnisattribut 1008
Verzeichnisdienst 121, 445, 654, 1010, 1022
Verzeichnisdienstklasse 1010
Verzeichnisobjekt 1007, 1012
Verzweigung 139
VHD 1107f., 1118, 1123
VHDX 1104, 1108, 1118
Video 1217
View 246
Vim 337
VirtualHardDisk 1108
Virtualisierung 1095
VirtualizingStackPanel 1214
Virtuelle Maschine *siehe* VM
Virtuelles System 1095
Virus 153
Visual Basic 487, 1325
Visual Basic 6.0 1329
Visual Basic for Applications *siehe* VBA
Visual Basic .NET 4, 1249f.
Visual Studio 331ff., 551, 1150f., 1219, 1249f., 1277
– Container 1141
Visual Studio Code 44, 146, 334, 378, 1146
Visual Studio Team Services 160
Visual Studio Team Services *siehe* VSTS
Visual Web Developer Express 1250
VM 657, 1095, 1102
VMBus 295
VMGUID 296
VMName 296
void 256
VolumeLabel 410
Volume Shadow Copy Service *siehe* VSS 736
VSCode
– Visual Studio Code 334
VSCode-PowerShell 334, 378
VSS 736
VSTS 1189
VT100 762

W

Wait 595
WaitForAll 616
WaitForAny 616
WaitForSome 616
Wait-Job 521, 524
Wait-Process 916
Walter Skinner 998
WarningAction 53, 229
WarningVariable 53
Warnung 53
WAS 919
WBEM 8, 434
WDAC 837
WebAdministration 1083, 1085, 1129
Webanwendung 1328
Web Based Enterprise Management 434
WebClient 958
Webdienst 953
Weblog 944, 1339
Webserver 263, 1088
Webservice 953, 955
Web Service Description Language 953
Web Services Description Language *siehe* WSDL
Website 949, 1088, 1094
Well-Known GUID 1006
Well-Known Object 1006
Well-Known Security Principal 981
WellKnownSidType 981
Werkzeug 301
Wertemenge 197
Wertkopie 203f.
Whatif 52f.
WhatIf 54, 703, 870, 1236, 1285
WhatIfPreference 54
Where() 125f.
WHERE 451
Where-Object 58f., 89ff., 106, 122, 124ff., 138, 142, 257, 918, 1252, 1264, 1277
while 206
Whistler 441
whoami.exe 310
Width 240
Wiederherstellungspunkt 862
Win32 437
Win32_Account 998
Win32_ACE 728
Win32-API 489
Win32_Battery 867
Win32_Bios 852
Win32_BootConfiguration 852
Win32_CDRomdrive 865
Win32_CDRomDrive 471
Win32_CodecFile 873
Win32_ComponentCategory 452
Win32_ComputerShutdownEvent 449, 570
Win32_Computersystem 850
Win32_Currenttime 196
Win32_Desktop 999
Win32_Diskdrive 865
Win32_Group 998
Win32_Keyboard 866
Win32_LocalTime 196
Win32_LogicalDisk 443, 452, 476, 690, 692f., 1263
Win32_MappedLogicalDisk 695
Win32_MemoryDevice 865
Win32_NetworkAdapter 866
Win32_NetworkAdapterConfiguration 452, 923, 929f.
Win32_NTLogEvent 452, 454, 571
Win32 OpenSSH 389
Win32_OperatingSystem 850f.
Win32_OSRecoveryConfiguration 853
Win32_PerfRawData 967
Win32_PerfRawData_PerfOS_Processor 967
Win32_PerfRawData_PerfProc_Process 967
Win32_PingStatus 933f.
Win32_PointingDevice 866
Win32_PowerManagementEvent 570
Win32_Printer 867, 869, 923
Win32_Printjob 867f.
Win32_Process 571
Win32_Processor 865f.
Win32_ProcessStartTrace 570
Win32_Product 101, 871, 873, 875
Win32_Quickfixengineering 873
Win32_SecurityDescriptor 728
Win32_Service 452, 454, 571, 917
Win32_Share 725f.
Win32_SoundDevice 865
Win32_SystemConfiguration-ChangeEvent 449, 570
Win32_Tapedrive 865
Win32_TCPIPPrinterPort 867f., 923
Win32_Trustee 728
Win32_USBController 866
Win32_UserAccount 140, 443, 998
Win32_VideoController 462, 471, 865, 867
Win32_Volume 695

Win32_WindowsProductActivation 853
window 1214
Windows
– Rolle 890, 1122
Windows 8 466
Windows 8.1 642
Windows 9x 444
Windows 10 5, 12, 17, 296, 302, 644
– Anniversary Update 5
Windows 2000 441
Windows Activation Service *siehe* WAS
Windows-Authentifizierung 822
Windows Communication Foundation 436
Windows Container *siehe* Docker
Windows Data Access Components *siehe* WDAC
Windows Defender 863, 1123
Windows Driver Model 445
Windows Explorer 737, 846
Windows Firewall 659, 934, 940
– im Netzwerk abfragen 940
– per PowerShell konfigurieren 934
Windows Forms 333, 1203, 1205, 1207
WindowsIdentity 972
Windows Installer 445, 905
Windows Management Framework 17, 434, 869
Windows Management Instrumentation 13
Windows ME 444
Windows Nano Server 12, 1119, 1133, 1138
Windows Nano Server 1709 12
Windows Nano Server 2016 12
Windows PowerShell XXV, 3, 7
Windows PowerShell Community Extensions 648
Windows Pre Installation Environment *siehe* WinPE
Windows Presentation Foundation *siehe* WPF
Windows Remote Management 269
Windows Remote Management *siehe* WinRM
Windows Script Host 4, 14, 152
Windows Server 1709 12
Windows Server 2003 4, 441, 444, 1021
Windows Server 2012 466, 640, 1060, 1083
Windows Server 2012 R2 343, 642
Windows Server 2016 5, 12, 17, 296, 302, 644
Windows Server-Containern *siehe* Docker
Windows Server Core 314, 1133, 1138
Windows Terminal 313
Windows Troubleshooting Platform 991
Windows Universal Platform *siehe* UWP
Windows Update 858, 861
– Agent API 1127
– Nano Server 1127
Windows Vista 1325
Windows Workflow Foundation 362
Windows XP 13, 441, 451
WinMgmt.exe 444 f.
WinPE 739
WinPSCompatSession 369
WinRM 269, 271, 273 f., 445, 521, 858
WITHIN 451, 570
WKGUID 1006
WMI 3, 8, 13, 196, 269, 362, 433, 436, 443, 463, 1263, 1268, 1281
– Class Explorer 456
– Command Shell 13
– Data Query 452
– Ereignis 449
– Event Query 451, 453
– Klasse 456
– Namespace 441
– Object Browser 455 f.
– Query Language 451, 465
– Repository 444, 449, 467
– Schema 443, 452
– Schema-Query 452
– Steuerung 444
WMI API 457
WMIClass 433, 463
WMI Object Browser 455
WMI Query Language *siehe* WQL
WMISEARCHER 433, 463, 465
Word 130, 431, 781
Workflow 537, 542, 544 f., 551
– Designer 552
– Einschränkungen 542
– Persistenz 550
– Verschachtelt 547
WorkflowInfo 552
WorkingSet 121
WorkingSet64 92
Work Item 1195
Workspace Settings 380
World Wide Wings 1292
Wörterbuch 130
WPF 77, 79 f., 243, 313, 333, 374, 418, 541, 652, 1203, 1209, 1217, 1329
WPF PowerShell Kit 652, 1209
WPK 652, 1209
WQL 451, 466, 569, 1264
WrapPanel 1214
Write-BZip2 722
Write-Clipboard 498
WriteDebug() 1281
Write-Error 250
WriteError() 1281, 1284
Write-EventLog 270, 362, 963, 965
Write-GZip 722
Write-Host 77, 237, 250, 342, 544
WriteObject 1257, 1266
WriteObject() 1252, 1264
Write-Output 66
Write-Progress 258 f., 480
Write-Tar 31, 722
WriteVerbose() 1281, 1284
Write-Warn 250
WriteWarning() 1281, 1284
Write-Zip 722
WScript.Shell 717
WSDL 953
WSH 1004
WSMan 263, 294
WS-Management 269 f., 273, 294, 445, 457, 460, 462
Wurzelnamensraum 1330
www.IT-Visions.de 639, 654, 666, 690, 807

X

x64 1095
x86 1095
XAML 541, 551, 1217
XamlReader 1218
XCopy-Deployment 1329, 1332
xDscDiagnostics 610
xDscWebService 605
X-Files 998
XFilesServer 998
XML 82, 493, 619, 733, 751 f., 757, 759, 955, 1287
XML Application Markup Language *siehe* XAML
XmlAttribute 755
XmlDocument 759
XmlElement 755
XML-Schema 753
XML-Webservice 1330
XPathDocumentNavigator 755
XslCompiledTransform 759

Y

YAML 1141, 1202
Year 104

YesNo 1204
YesNoCancel 1204

Z

Zahl 177
Zahlenliteral 177
Zeichenkette 179 f., 188, 251, 1274, 1294
– ersetzen 187
– Operation 185
– trennen 187
– verbinden 188
Zeichensatz 745
Zeilenumbruch 55, 99
– Pipeline 98
Zeitmessung 501
Zeitplandienst 435
Zertifikat 339, 516, 988
– selbst signiert 516
Zertifikatsdatei 518
Zertifikatsspeicher 3, 261, 988
Zertifikatsverwaltung 515 f., 519
ZIP 721 f., 781
ZipFile 723
zsh 357
Zufallszahl 178
Zugriffsrechteliste 971, 977
Zugriff verweigert 475
Zuweisungsoperator 203
Zwischenablage 498
Zwischencode 1327
Zwischenschritt 138
Zwischenspeicher 790