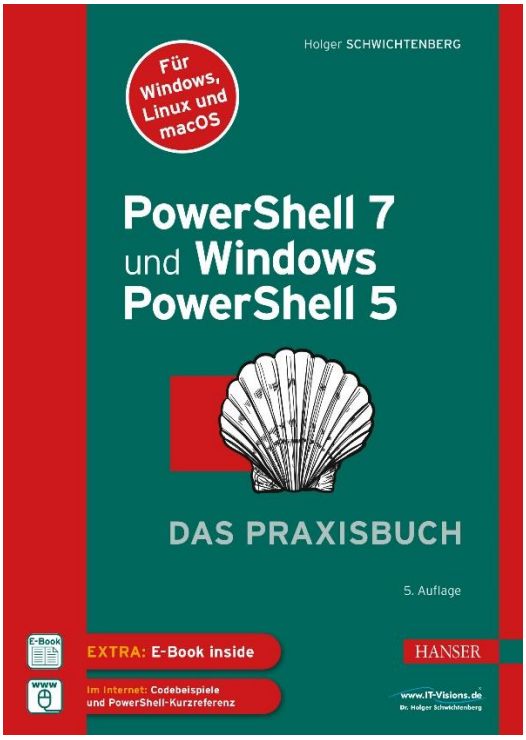
HANSER



Leseprobe

zu

PowerShell 7 und Windows PowerShell 5

von Holger Schwichtenberg

Print-ISBN: 978-3-446-47296-9
E-Book-ISBN: 978-3-446-47446-8
E-Pub-ISBN: 978-3-446-47574-8

Weitere Informationen und Bestellungen unter
https://www.hanser-kundencenter.de/fachbuch/artikel/9783446472969
sowie im Buchhandel

Inhalt

Vorwort ..... **XXIV**

Über den Autor ..... **XXXII**

Teil A: PowerShell-Basiswissen **1**

1 Fakten zur PowerShell **3**

1.1 Was ist die PowerShell? 3
1.2 Geschichte der PowerShell 4
1.3 Welche Varianten und Versionen der PowerShell gibt es? 6
1.4 Windows PowerShell versus PowerShell Core versus PowerShell 7.x 6
1.5 Motivation zur PowerShell 8
1.6 Betriebssysteme mit vorinstallierter PowerShell 11
1.7 Support der PowerShell 13
1.8 Einflussfaktoren auf die Entwicklung der PowerShell 15
1.9 Anbindung an Klassenbibliotheken 16
1.10 PowerShell versus WSH 17

2 Erste Schritte mit der PowerShell **20**

2.1 Windows PowerShell herunterladen und auf anderen Windows-Betriebssystemen installieren 20
2.2 Die Windows PowerShell testen 24
2.3 Woher kommen die PowerShell-Befehle? 33
2.4 PowerShell Community Extensions (PSCX) herunterladen und installieren 34
2.5 Den Windows PowerShell-Editor „ISE“ verwenden 41
2.6 PowerShell 7 installieren und testen 45

3 Einzelbefehle der PowerShell **57**

3.1 Commandlets 57
3.2 Aliase 70
3.3 Ausdrücke 78
3.4 Externe Befehle (klassische Kommandozeilenbefehle) 79
3.5 Dateinamen 81

4 Hilfefunktionen ... 82
4.1 Auflisten der verfügbaren Befehle ... 82
4.2 Praxistipp: Den Standort eines Kommandozeilenbefehls suchen ... 83
4.3 Anzahl der Befehle ... 84
4.4 Volltextsuche ... 86
4.5 Erläuterungen zu den Befehlen ... 86
4.6 Hilfe zu Parametern ... 87
4.7 Hilfe mit Show-Command ... 89
4.8 Hilfefenster ... 90
4.9 Allgemeine Hilfetexte ... 92
4.10 Aktualisieren der Hilfedateien ... 92
4.11 Online-Hilfe ... 94
4.12 Fehlende Hilfetexte ... 95
4.13 Dokumentation der .NETKlassen ... 96

5 Objektorientiertes Pipelining ... 98
5.1 Befehlsübersicht ... 98
5.2 Pipeline-Operator ... 99
5.3 .NET-Objekte in der Pipeline ... 100
5.4 Pipeline Processor ... 101
5.5 Pipelining von Parametern ... 103
5.6 Pipelining von klassischen Befehlen ... 105
5.7 Zeilenumbrüche in Pipelines ... 107
5.8 Schleifen ... 108
5.9 Zugriff auf einzelne Objekte aus einer Menge ... 111
5.10 Zugriff auf einzelne Werte in einem Objekt ... 112
5.11 Methoden ausführen ... 114
5.12 Analyse des Pipeline-Inhalts ... 116
5.13 Filtern ... 131
5.14 Zusammenfassung von Pipeline-Inhalten ... 136
5.15 „Kastrierung“ von Objekten in der Pipeline ... 136
5.16 Sortieren ... 137
5.17 Duplikate entfernen ... 138
5.18 Gruppierung ... 139
5.19 Objekte verbinden mit Join-String ... 145
5.20 Berechnungen ... 146
5.21 Zwischenschritte in der Pipeline mit Variablen ... 146
5.22 Verzweigungen in der Pipeline ... 147
5.23 Vergleiche zwischen Objekten ... 149
5.24 Weitere Praxislösungen ... 150

6 PowerShell-Skripte ... 152
6.1 Skriptdateien ... 152
6.2 Start eines Skripts ... 154
6.3 Aliase für Skripte verwenden ... 155
6.4 Parameter für Skripte ... 156
6.5 Skripte dauerhaft einbinden (Dot Sourcing) ... 157
6.6 Das aktuelle Skriptverzeichnis ... 158
6.7 Sicherheitsfunktionen für PowerShell-Skripte ... 158
6.8 Skripte mit vollen Rechten (Elevation) ... 160
6.9 Blockierte PowerShell-Skripte ... 161
6.10 PowerShell-Skripte im Kontextmenü des Windows Explorers ... 162
6.11 Anforderungsdefinitionen von Skripten ... 164
6.12 Skripte anhalten ... 165
6.13 Versionierung und Versionsverwaltung von Skripten ... 165

7 PowerShell-Skriptsprache ... 168
7.1 Hilfe zur PowerShell-Skriptsprache ... 168
7.2 Befehlstrennung ... 168
7.3 Kommentare ... 169
7.4 Variablen ... 170
7.5 Variablenbedingungen ... 182
7.6 Zahlen ... 183
7.7 Zeichenketten (Strings) ... 187
7.8 Reguläre Ausdrücke ... 197
7.9 Datum und Uhrzeit ... 203
7.10 Objekte ... 204
7.11 Arrays ... 205
7.12 ArrayList ... 208
7.13 Assoziative Arrays (Hash-Tabellen) ... 209
7.14 Operatoren ... 210
7.15 Überblick über die Kontrollkonstrukte ... 214
7.16 Bedingungen ... 219
7.17 Unterroutinen (Prozedur/Funktionen) ... 222
7.18 Eingebaute Funktionen ... 228
7.19 Fehlerausgabe ... 229
7.20 Fehlerbehandlung ... 231
7.21 Laufzeitfehler erzeugen ... 243
7.22 Objektorientiertes Programmieren mit Klassen ... 243

8 Ausgaben ... 246
8.1 Ausgabe-Commandlets ... 246
8.2 Benutzerdefinierte Tabellenformatierung ... 249
8.3 Benutzerdefinierte Listenausgabe ... 251
8.4 Mehrspaltige Ausgabe ... 251
8.5 Out-GridView ... 252
8.6 Standardausgabe ... 254
8.7 Einschränkung der Ausgabe ... 256
8.8 Seitenweise Ausgabe ... 256
8.9 Ausgabe einzelner Werte ... 258
8.10 Details zum Ausgabeoperator ... 260
8.11 Ausgabe von Methodenergebnissen und Unterobjekten in Pipelines ... 263
8.12 Ausgabe von Methodenergebnissen und Unterobjekten in Zeichenketten ... 264
8.13 Unterdrückung der Ausgabe ... 264
8.14 Ausgaben an Drucker ... 265
8.15 Ausgaben in Dateien ... 266
8.16 Umleitungen (Redirection) ... 266
8.17 Fortschrittsanzeige ... 267
8.18 Sprachausgabe ... 267

9 Das PowerShell-Navigationsmodell (PowerShell Provider) ... 269
9.1 Einführungsbeispiel: Navigation in der Registrierungsdatenbank ... 269
9.2 Provider und Laufwerke ... 270
9.3 Navigationsbefehle ... 272
9.4 Pfadangaben ... 273
9.5 Beispiel ... 275
9.6 Eigene Laufwerke definieren ... 276

10 Fernausführung (Remoting) ... 277
10.1 RPC-Fernabfrage ohne WS-Management ... 278
10.2 Anforderungen an PowerShell Remoting ... 279
10.3 Rechte für PowerShell-Remoting ... 280
10.4 Einrichten von PowerShell Remoting ... 280
10.5 Überblick über die Fernausführungs-Commandlets ... 283
10.6 Interaktive Fernverbindungen im Telnet-Stil ... 283
10.7 Fernausführung von Befehlen ... 285
10.8 Parameterübergabe an die Fernausführung ... 289
10.9 Fernausführung von Skripten ... 290
10.10 Ausführung auf mehreren Computern ... 291
10.11 Sitzungen ... 292
10.12 Implizites Remoting ... 297

10.13 Zugriff auf entfernte Computer außerhalb der eigenen Domäne 298
10.14 Verwaltung des WS-Management-Dienstes 301
10.15 PowerShell Direct für Hyper-V 302
10.16 Praxislösung zu PowerShell Direct 304

11 PowerShell-Werkzeuge 307
11.1 PowerShell-Standardkonsole 307
11.2 Windows Terminal 322
11.3 Erweiterung der Konsolen 327
11.4 PowerShell Integrated Scripting Environment (ISE) 329
11.5 PowerShell Script Analyzer 340
11.6 PowerShell Analyzer 345
11.7 PowerShell Tools for Visual Studio 346
11.8 PowerShell Pro Tools for Visual Studio 348
11.9 Visual Studio Developer PowerShell 348
11.10 NuGet Package Manager Console (PMC) 351
11.11 Visual Studio Code mit PowerShell-Erweiterung 352
11.12 PowerShell-Erweiterungen für andere Editoren 354
11.13 PowerShell Web Access (PSWA) 355
11.14 Azure Cloud Shell 360
11.15 ISE Steroids 360
11.16 PowerShellPlus 361
11.17 PoshConsole 364
11.18 PowerGUI 365
11.19 PrimalScript 365
11.20 CIM Explorer for PowerShell ISE 367

12 Windows PowerShell Core 5.1 in Windows Nano Server 369
12.1 Installation 369
12.2 PowerShell-Skriptsprache 369
12.3 Werkzeuge 369
12.4 Fehlende Funktionen 370

13 PowerShell 7 für Windows, Linux und macOS 371
13.1 Motivation für den Einsatz der PowerShell 7 auf Linux und macOS 371
13.2 Basis der PowerShell 7 372
13.3 Identifizierung der PowerShell 7 373
13.4 Funktionsumfang der PowerShell 7 373
13.5 Entfallene Befehle in PowerShell 7 376
13.6 Erweiterungsmodule nutzen in PowerShell 7 382
13.7 Geänderte Funktionen in PowerShell 7 387

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

Teil B: PowerShell-Aufbauwissen 413

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

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

16 Zugriff auf die Windows Management Instrumentation (WMI) ... 447
16.1 Einführung in WMI ... 447
16.2 WMI in der PowerShell ... 474
16.3 Open Management Infrastructure (OMI) ... 476
16.4 Abruf von WMI-Objektmengen ... 476
16.5 Fernzugriffe ... 477
16.6 Filtern und Abfragen ... 478
16.7 Liste aller WMI-Klassen ... 481
16.8 Hintergrundwissen: WMI-Klassenprojektion mit dem PowerShell-WMI-Objektadapter ... 482
16.9 Beschränkung der Ausgabeliste bei WMI-Objekten ... 486
16.10 Zugriff auf einzelne Mitglieder von WMI-Klassen ... 488
16.11 Werte setzen in WMI-Objekten ... 488
16.12 Umgang mit WMI-Datumsangaben ... 490
16.13 Methodenaufrufe ... 491
16.14 Neue WMI-Instanzen erzeugen ... 492
16.15 Instanzen entfernen ... 493
16.16 Commandlet Definition XML-Datei (CDXML) ... 493

17 Dynamische Objekte ... 497
17.1 Erweitern bestehender Objekte ... 497
17.2 Komplett dynamische Objekte ... 499

18 Einbinden von C# und Visual Basic .NET ... 501

19 Win32-API-Aufrufe ... 503

20 Benutzereingaben ... 506
20.1 Read-Host ... 506
20.2 Benutzerauswahl ... 507
20.3 Grafischer Eingabedialog ... 508
20.4 Dialogfenster ... 509
20.5 Authentifizierungsdialog ... 509
20.6 Zwischenablage (Clipboard) ... 511

21 Fehlersuche ... 512
21.1 Detailinformationen ... 512
21.2 Einzelschrittmodus ... 513
21.3 Zeitmessung ... 514
21.4 Ablaufverfolgung (Tracing) ... 515
21.5 Erweiterte Protokollierung aktivieren ... 517

21.6 Script-Debugging in der ISE ... 518
21.7 Kommandozeilenbasiertes Script-Debugging ... 518

22 Transaktionen ... 520
22.1 Commandlets für Transaktionen ... 520
22.2 Start und Ende einer Transaktion ... 521
22.3 Zurücksetzen der Transaktion ... 522
22.4 Mehrere Transaktionen ... 523

23 Standardeinstellungen ändern mit Profilskripten ... 524
23.1 Profilpfade ... 524
23.2 Ausführungsreihenfolge ... 526
23.3 Beispiel für eine Profildatei ... 526
23.4 Starten der PowerShell ohne Profilskripte ... 528

24 Digitale Signaturen für PowerShell-Skripte ... 529
24.1 Zertifikat erstellen ... 529
24.2 Skripte signieren ... 531
24.3 Verwenden signierter Skripte ... 533
24.4 Mögliche Fehlerquellen ... 533

25 Hintergrundaufträge („Jobs“) ... 534
25.1 Voraussetzungen ... 534
25.2 Architektur ... 534
25.3 Starten eines Hintergrundauftrags ... 535
25.4 Hintergrundaufträge abfragen ... 536
25.5 Warten auf einen Hintergrundauftrag ... 537
25.6 Abbrechen und Löschen von Aufträgen ... 537
25.7 Analyse von Fehlermeldungen ... 537
25.8 Fernausführung von Hintergrundaufträgen ... 538
25.9 Praxislösung: Einen Job auf mehreren Computern starten ... 538

26 Geplante Aufgaben und zeitgesteuerte Jobs ... 540
26.1 Geplante Aufgaben (Scheduled Tasks) ... 540
26.2 Zeitgesteuerte Jobs ... 544

27 PowerShell-Workflows ... 550
27.1 Ein erstes Beispiel ... 550
27.2 Unterschiede zu einer Function bzw. einem Skript ... 554
27.3 Einschränkungen bei Workflows ... 555
27.4 Workflows in der Praxis ... 556
27.5 Workflows in Visual Studio erstellen ... 564

28 Ereignissystem ... 582
28.1 WMI-Ereignisse ... 582
28.2 WMI-Ereignisabfragen ... 582
28.3 WMI-Ereignisse seit PowerShell 1.0 ... 584
28.4 Registrieren von WMIEreignisquellen seit PowerShell 2.0 ... 585
28.5 Auslesen der Ereignisliste ... 586
28.6 Reagieren auf Ereignisse ... 588
28.7 WMI-Ereignisse seit PowerShell-Version 3.0 ... 590
28.8 Registrieren von .NET-Ereignissen ... 590
28.9 Erzeugen von Ereignissen ... 591

29 Datenbereiche und Datendateien ... 593
29.1 Datenbereiche ... 593
29.2 Datendateien ... 595
29.3 Mehrsprachigkeit/Lokalisierung ... 596

30 Desired State Configuration (DSC) ... 599
30.1 Grundprinzipien ... 600
30.2 DSC für PowerShell 7 ... 600
30.3 Ressourcen ... 601
30.4 Verfügbare DSC-Ressourcen ... 602
30.5 Eigenschaften einer Ressource ... 605
30.6 Aufbau eines DSC-Dokuments ... 605
30.7 Commandlets für die Arbeit mit DSC ... 606
30.8 Ein erstes DSC-Beispiel ... 606
30.9 Kompilieren und Anwendung eines DSC-Dokuments ... 607
30.10 Variablen in DSC-Dateien ... 609
30.11 Parameter für DSC-Dateien ... 610
30.12 Konfigurationsdaten ... 611
30.13 Entfernen einer DSC-Konfiguration ... 614
30.14 DSC Pull Server ... 617
30.15 DSC-Praxislösung 1: IIS installieren ... 624
30.16 DSC-Praxislösung 2: Software installieren ... 626
30.17 DSC-Praxislösung 3: Software deinstallieren ... 628
30.18 Realisierung einer DSC-Ressource ... 629
30.19 Weitere Möglichkeiten ... 629

31 PowerShell-Snap-Ins ... 630
31.1 Einbinden von Snap-Ins ... 630
31.2 Liste der Commandlets ... 634

32 PowerShell-Module ... 635
32.1 Überblick über die Commandlets ... 635
32.2 Modularchitektur ... 636
32.3 Aufbau eines Moduls ... 637
32.4 Module aus dem Netz herunterladen und installieren mit PowerShellGet ... 638
32.5 Module manuell installieren ... 644
32.6 Doppeldeutige Namen ... 644
32.7 Auflisten der verfügbaren Module ... 646
32.8 Importieren von Modulen ... 647
32.9 Entfernen von Modulen ... 650

33 Ausgewählte PowerShell-Erweiterungen ... 651
33.1 PowerShell-Module in Windows 8.0 und Windows Server 2012 ... 652
33.2 PowerShell-Module in Windows 8.1 und Windows Server 2012 R2 ... 654
33.3 PowerShell-Module in Windows 10 und Windows Server 2019 ... 656
33.4 PowerShell Community Extensions (PSCX) ... 660
33.5 PowerShellPack ... 664
33.6 www.IT-Visions.de: PowerShell Extensions ... 666
33.7 Quest Management Shell for Active Directory ... 666
33.8 Microsoft Exchange Server ... 668
33.9 System Center Virtual Machine Manager ... 669
33.10 PowerShell Management Library for Hyper-V (pshyperv) ... 669
33.11 PowerShell Configurator (PSConfig) ... 670

34 Delegierte Administration/Just Enough Administration (JEA) ... 672
34.1 JEA-Konzept ... 672
34.2 PowerShell-Sitzungskonfiguration erstellen ... 672
34.3 Sitzungskonfiguration nutzen ... 676
34.4 Delegierte Administration per Webseite ... 677

35 Tipps und Tricks zur PowerShell ... 678
35.1 Alle Anzeigen löschen ... 678
35.2 Befehlsgeschichte ... 678
35.3 System- und Hostinformationen ... 679
35.4 Anpassen der Eingabeaufforderung (Prompt) ... 680
35.5 PowerShell-Befehle aus anderen Anwendungen heraus starten ... 681
35.6 ISE erweitern ... 682
35.7 PowerShell für Gruppenrichtlinienskripte ... 683
35.8 Einblicke in die Interna der Pipeline-Verarbeitung ... 686

Teil C:PowerShell im Praxiseinsatz ... **687**

36 Dateisystem ... **689**
36.1 Laufwerke ... 690
36.2 Ordnerinhalte ... 695
36.3 Dateieigenschaften verändern ... 702
36.4 Eigenschaften ausführbarer Dateien ... 703
36.5 Kurznamen ... 705
36.6 Lange Pfade ... 705
36.7 Dateisystemoperationen ... 706
36.8 Praxislösung: Dateien umorganisieren ... 706
36.9 Praxislösung: Zufällige Dateisystemstruktur erzeugen ... 708
36.10 Praxislösung: Leere Ordner löschen ... 709
36.11 Praxislösung: Geschwindigkeitsmessung des Dateisystems (beim Kopieren von Dateien) ... 711
36.12 Einsatz von Robocopy in der PowerShell ... 712
36.13 NTFS-Komprimierung ... 715
36.14 Dateisystemkataloge ... 716
36.15 Papierkorb leeren ... 716
36.16 Dateieigenschaften lesen ... 717
36.17 Praxislösung: Fotos nach Aufnahmedatum sortieren ... 717
36.18 Datei-Hash ... 718
36.19 Finden von Duplikaten ... 719
36.20 Verknüpfungen im Dateisystem ... 721
36.21 Komprimierung ... 726
36.22 Dateisystemfreigaben ... 730
36.23 Überwachung des Dateisystems ... 741
36.24 Dateiversionsverlauf ... 742
36.25 Windows Explorer öffnen ... 743
36.26 Windows Server Backup ... 743

37 Festplattenverschlüsselung mit BitLocker ... **745**
37.1 Übersicht über das BitLocker-Modul ... 746
37.2 Verschlüsseln eines Laufwerks ... 747

38 Dokumente ... **748**
38.1 Textdateien ... 748
38.2 CSV-Dateien ... 750
38.3 Analysieren von Textdateien ... 753
38.4 INI-Dateien ... 757
38.5 XML-Dateien ... 757

38.6 HTML- und Markdown-Dateien ... 769
38.7 JSON-Dateien ... 772
38.8 Binärdateien ... 783
38.9 Praxislösung: Grafikdateien verändern ... 784
38.10 Praxislösung: Drucken vieler Dateien ... 785

39 Microsoft Office ... 786
39.1 Allgemeine Informationen zur Office-Automatisierung per PowerShell ... 786
39.2 Praxislösung: Terminserien aus Textdateien anlegen in Outlook ... 787
39.3 Praxislösung: Outlook-Termine anhand von Suchkriterien löschen ... 789
39.4 Praxislösung: Grafiken aus einem Word-Dokument (DOCX) extrahieren ... 790

40 Datenbanken ... 793
40.1 ADO.NET-Grundlagen ... 793
40.2 Beispieldatenbank ... 799
40.3 Datenzugriff mit den Bordmitteln der PowerShell ... 800
40.4 Hilfsfroutinen für den Datenbankzugriff (DBUtil.ps1) ... 813
40.5 Datenzugriff mit den PowerShell-Erweiterungen ... 816
40.6 Datenbankzugriff mit SQLPS ... 820
40.7 Datenbankzugriff mit SQLPSX ... 820

41 Microsoft-SQL-Server-Administration ... 821
41.1 PowerShell-Integration im SQL Server Management Studio ... 822
41.2 SQL-Server-Laufwerk „SQLSERVER:“ ... 823
41.3 Die SQLPS-Commandlets ... 826
41.4 Die SQL Server Management Objects (SMO) ... 828
41.5 SQLPSX ... 831
41.6 Microsoft-SQL-Server-Administration mit der PowerShell in der Praxis ... 838

42 ODBC-Datenquellen ... 844
42.1 ODBC-Treiber und -Datenquellen auflisten ... 845
42.2 Anlegen einer ODBC-Datenquelle ... 846
42.3 Zugriff auf eine ODBC-Datenquelle ... 847

43 Registrierungsdatenbank (Registry) ... 849
43.1 Schlüssel auslesen ... 849
43.2 Schlüssel anlegen und löschen ... 850
43.3 Laufwerke definieren ... 850
43.4 Werte anlegen und löschen ... 851
43.5 Werte auslesen ... 852
43.6 Praxislösung: Windows-Explorer-Einstellungen ... 853
43.7 Praxislösung: Massenanlegen von Registry-Schlüsseln ... 853

44 Computer- und Betriebssystemverwaltung ... 855
44.1 Computerinformationen ... 855
44.2 Versionsnummer des Betriebssystems ... 857
44.3 Zeitdauer seit dem letzten Start des Betriebssystems ... 857
44.4 BIOS- und Startinformationen ... 858
44.5 Windows-Produktaktivierung ... 859
44.6 Umgebungsvariablen ... 859
44.7 Schriftarten ... 863
44.8 Computername und Domäne ... 863
44.9 Herunterfahren und Neustarten ... 864
44.10 Windows Updates installieren ... 865
44.11 Wiederherstellungspunkte verwalten ... 869

45 Windows Defender ... 870

46 Hardwareverwaltung ... 871
46.1 Hardwarebausteine ... 871
46.2 Plug-and-Play-Geräte ... 873
46.3 Druckerverwaltung (ältere Betriebssysteme) ... 873
46.4 Druckerverwaltung (seit Windows 8 und Windows Server 2012) ... 875

47 Softwareverwaltung ... 877
47.1 Softwareinventarisierung ... 877
47.2 Installation von Anwendungen ... 880
47.3 Deinstallation von Anwendungen ... 881
47.4 Praxislösung: Installationstest ... 882
47.5 Praxislösung: Installierte .NET SDKs aufräumen ... 883
47.6 Windows 10 Apps verwalten ... 887
47.7 Installationen mit PowerShell Package Management („OneGet“) ... 890
47.8 Versionsnummer ermitteln ... 893
47.9 Servermanager ... 894
47.10 Windows-Features installieren auf Windows-Clientbetriebssystemen ... 905
47.11 Praxislösung: IIS-Installation ... 907
47.12 Softwareeinschränkungen mit dem PowerShell-Modul „AppLocker“ ... 909

48 Prozessverwaltung ... 915
48.1 Prozesse auflisten ... 915
48.2 Prozesse starten ... 916
48.3 Prozesse mit vollen Administratorrechten starten ... 917
48.4 Prozesse unter einem anderen Benutzerkonto starten ... 918
48.5 Prozesse beenden ... 919
48.6 Warten auf das Beenden einer Anwendung ... 920

49 Windows-Systemdienste ... 921
49.1 Dienste auflisten ... 921
49.2 Dienstzustand ändern ... 924
49.3 Diensteigenschaften ändern ... 924
49.4 Dienste hinzufügen ... 925
49.5 Dienste entfernen ... 926

50 Netzwerk ... 927
50.1 Netzwerkkonfiguration ... 927
50.2 DNS-Client-Konfiguration ... 932
50.3 DNS-Namensauflösung ... 936
50.4 Erreichbarkeit prüfen (Ping) ... 937
50.5 Windows Firewall ... 938
50.6 Remote Desktop (RDP) einrichten ... 945
50.7 E-Mails senden (SMTP) ... 946
50.8 Auseinandernehmen von E-Mail-Adressen ... 947
50.9 Abruf von Daten von einem HTTP-Server ... 947
50.10 Praxislösung: Linkprüfer für eine Website ... 954
50.11 Aufrufe von SOAP-Webdiensten ... 957
50.12 Aufruf von REST-Diensten ... 960
50.13 File Transfer Protocol (FTP) ... 962
50.14 Hintergrunddatentransfer mit BITS ... 963

51 Ereignisprotokolle (Event Log) ... 967
51.1 Protokolleinträge auslesen ... 967
51.2 Ereignisprotokolle erzeugen ... 969
51.3 Protokolleinträge erzeugen ... 969
51.4 Protokollgröße festlegen ... 969
51.5 Protokolleinträge löschen ... 969

52 Leistungsdaten (Performance Counter) ... 970
52.1 Zugriff auf Leistungsindikatoren über WMI ... 970
52.2 Get-Counter ... 971

53 Sicherheitseinstellungen ... 973
53.1 Aktueller Benutzer ... 973
53.2 Grundlagen ... 974
53.3 Zugriffsrechtelisten auslesen ... 979
53.4 Einzelne Rechteeinträge auslesen ... 980
53.5 Besitzer auslesen ... 982
53.6 Benutzer und SID ... 982

53.7 Hinzufügen eines Rechteeintrags zu einer Zugriffsrechteliste ... 986
53.8 Entfernen eines Rechteeintrags aus einer Zugriffsrechteliste ... 988
53.9 Zugriffsrechteliste übertragen ... 990
53.10 Zugriffsrechteliste über SDDL setzen ... 991
53.11 Zertifikate verwalten ... 992

54 Optimierungen und Problemlösungen ... 995
54.1 PowerShell-Modul „TroubleshootingPack“ ... 995
54.2 PowerShell-Modul „Best Practices“ ... 999

55 Active Directory ... 1001
55.1 Benutzer- und Gruppenverwaltung mit WMI ... 1003
55.2 Einführung in System.DirectoryServices ... 1003
55.3 Basiseigenschaften ... 1015
55.4 Benutzer- und Gruppenverwaltung im Active Directory ... 1017
55.5 Verwaltung der Organisationseinheiten ... 1025
55.6 Suche im Active Directory ... 1026
55.7 Navigation im Active Directory mit den PowerShell Extensions ... 1033
55.8 Verwendung der Active-Directory-Erweiterungen von www.IT-Visions.de ... 1034
55.9 PowerShell-Modul „Active Directory“ (ADPowerShell) ... 1036
55.10 PowerShell-Modul „ADDSDeployment“ ... 1065
55.11 Informationen über die Active Directory-Struktur ... 1068

56 Gruppenrichtlinien ... 1071
56.1 Verwaltung der Gruppenrichtlinien ... 1071
56.2 Verknüpfung der Gruppenrichtlinien ... 1073
56.3 Gruppenrichtlinienberichte ... 1075
56.4 Gruppenrichtlinienvererbung ... 1077
56.5 Weitere Möglichkeiten ... 1078

57 Lokale Benutzer und Gruppen ... 1079
57.1 Modul „Microsoft.PowerShell.LocalAccounts“ ... 1079
57.2 Lokale Benutzerverwaltung in älteren PowerShell-Versionen ... 1080

58 Microsoft Exchange Server ... 1083
58.1 Daten abrufen ... 1083
58.2 Postfächer verwalten ... 1084
58.3 Öffentliche Ordner verwalten ... 1085

59 Internet Information Services (IIS) ... 1086
59.1 Überblick ... 1086
59.2 Navigationsprovider ... 1088

59.3 Anlegen von Websites ... 1090
59.4 Praxislösung: Massenanlegen von Websites ... 1091
59.5 Ändern von Website-Eigenschaften ... 1093
59.6 Anwendungspool anlegen ... 1094
59.7 Virtuelle Verzeichnisse und IIS-Anwendungen ... 1095
59.8 Website-Zustand ändern ... 1095
59.9 Anwendungspools starten und stoppen ... 1096
59.10 Löschen von Websites ... 1096

60 Virtuelle Systeme mit Hyper-V ... 1097
60.1 Das Hyper-V-Modul von Microsoft ... 1098
60.2 Die ersten Schritte mit dem Hyper-V-Modul ... 1100
60.3 Virtuelle Maschinen anlegen ... 1104
60.4 Umgang mit virtuellen Festplatten ... 1110
60.5 Konfiguration virtueller Maschinen ... 1113
60.6 Praxislösungen: Ressourcennutzung überwachen ... 1117
60.7 Dateien kopieren in virtuelle Systeme ... 1119
60.8 PowerShell Management Library for Hyper-V (für ältere Betriebssysteme) ... 1120

61 Windows Nano Server ... 1123
61.1 Das Konzept von Nano Server ... 1123
61.2 Einschränkungen von Nano Server ... 1125
61.3 Varianten des Nano Servers ... 1127
61.4 Installation eines Nano Servers ... 1127
61.5 Docker-Image ... 1128
61.6 Fernverwaltung mit PowerShell ... 1129
61.7 Windows Update auf einem Nano Server ... 1131
61.8 Nachträgliche Paketinstallation ... 1131
61.9 Abgespeckter IIS unter Nano Server ... 1133
61.10 Nano-Serververwaltung aus der Cloud heraus ... 1134

62 Docker-Container ... 1135
62.1 Container-Varianten für Windows ... 1135
62.2 Docker-Installation auf aktuellem Windows 10 und Windows 11 ... 1139
62.3 Docker-Installation auf älteren Windows 10-Clients ... 1147
62.4 Docker-Installation auf Windows Server ... 1149
62.5 Docker PowerShell installieren ... 1151
62.6 Docker-Basiswissen ... 1152
62.7 Container mit modernem .NET ... 1155
62.8 Container mit IIS-Webserver und klassischem ASP.NET ... 1164
62.9 Container mit Linux und PowerShell 7 ... 1173

62.10 Container mit Linux und Microsoft SQL Server ... 1175
62.11 Docker-Container mit Visual Studio ... 1177
62.12 Weitere Container-Befehle ... 1182

63 Microsoft Azure ... 1188
63.1 Azure-Konzepte ... 1188
63.2 Kommandozeilenwerkzeuge für die Azure-Verwaltung ... 1190
63.3 Benutzeranmeldung und Informationsabfrage ... 1193
63.4 Azure Ressourcen-Gruppen ... 1194
63.5 Azure Web-Apps ... 1194
63.6 Azure SQL Server ... 1196
63.7 Azure Kubernetes Services (AKS) ... 1197
63.8 Azure DevOps (ADO) ... 1221

64 Grafische Benutzeroberflächen (GUI) ... 1242
64.1 Einfache Nachfragedialoge ... 1242
64.2 Einfache Eingabe mit Inputbox ... 1243
64.3 Komplexere Eingabemasken ... 1244
64.4 Universelle Objektdarstellung ... 1246
64.5 WPF PowerShell Kit (WPK) ... 1247
64.6 Direkte Verwendung von WPF ... 1255

Teil D: Profiwissen – Erweitern der PowerShell ... 1257

65 Unit Tests mit Pester ... 1259
65.1 Einführung in das Konzept des Unit Testing ... 1259
65.2 Pester installieren ... 1260
65.3 Befehle in Pester ... 1260
65.4 Testen einer PowerShell-Funktion ... 1261
65.5 Testgenerierung ... 1262
65.6 Tests starten ... 1262
65.7 Prüf-Operationen ... 1264
65.8 Mock-Objekte ... 1264
65.9 Test von Dateisystemoperationen ... 1265

66 Entwicklung von Commandlets in der PowerShell-Skriptsprache ... 1267
66.1 Aufbau eines skriptbasierten Commandlets ... 1267
66.2 Verwendung per Dot Sourcing ... 1269
66.3 Parameterfestlegung ... 1270
66.4 Fortgeschrittene Funktion (Advanced Function) ... 1276

66.5 Mehrere Parameter und Parametersätze ... 1279
66.6 Unterstützung für Sicherheitsabfragen (-whatif und -confirm) ... 1281
66.7 Kaufmännisches Beispiel: Test-CustomerID ... 1283
66.8 Erweitern bestehender Commandlets durch Proxy-Commandlets ... 1286
66.9 Dokumentation ... 1292

67 Entwicklung eigener Commandlets mit C# ... 1296
67.1 Technische Voraussetzungen ... 1297
67.2 Grundkonzept der .NET-basierten Commandlets ... 1299
67.3 Schrittweise Erstellung eines minimalen Commandlets ... 1301
67.4 Erstellung eines Commandlets mit einem Rückgabeobjekt ... 1309
67.5 Erstellung eines Commandlets mit mehreren Rückgabeobjekten ... 1311
67.6 Erstellen eines Commandlets mit Parametern ... 1315
67.7 Verarbeiten von Pipeline-Eingaben ... 1317
67.8 Verkettung von Commandlets ... 1320
67.9 Fehlersuche in Commandlets ... 1324
67.10 Statusinformationen ... 1327
67.11 Unterstützung für Sicherheitsabfragen (-whatif und -confirm) ... 1332
67.12 Festlegung der Hilfeinformationen ... 1334
67.13 Erstellung von Commandlets für den Zugriff auf eine Geschäftsanwendung ... 1339
67.14 Konventionen für Commandlets ... 1340
67.15 Weitere Möglichkeiten ... 1342

68 PowerShell-Module erstellen ... 1343
68.1 Erstellen eines Skriptmoduls ... 1343
68.2 Praxislösung: Umwandlung einer Skriptdatei in ein Modul ... 1345
68.3 Erstellen eines Moduls mit Binärdateien ... 1345
68.4 Erstellen eines Moduls mit Manifest ... 1346
68.5 Erstellung eines Manifest-Moduls mit Visual Studio ... 1353

69 Hosting der PowerShell ... 1355
69.1 Voraussetzungen für das Hosting ... 1356
69.2 Hosting mit PSHost ... 1357
69.3 Vereinfachtes Hosting seit PowerShell 2.0 ... 1360

Anhang A: Crashkurs Objektorientierung ... 1363

Anhang B: Crashkurs .NET ... 1371
B.1 Was ist das .NET Framework? ... 1374
B.2 Was ist .NET Core/.NET? ... 1375
B.3 Eigenschaften von .NET ... 1376

B.4 .NET-Klassen ... 1377
B.5 Namensgebung von .NET-Klassen (Namensräume) ... 1377
B.6 Namensräume und Softwarekomponenten ... 1379
B.7 Bestandteile einer .NET-Klasse ... 1380
B.8 Vererbung ... 1381
B.9 Schnittstellen ... 1381

Anhang C: Weitere Informationen im Internet ... 1382

Anhang D: Abkürzungsverzeichnis ... 1383

Stichwortverzeichnis ... 1407

Vorwort

Liebe Leserin, lieber Leser,

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

Was ist das Thema dieses Buchs?

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

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

Wer bin ich?

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

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

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

Wer sind Sie?

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

Sind Sie Systemadministrator in einem Windows-Netzwerk?	○ Ja	○ Nein
Laufen die für Sie relevanten Computer mit den von PowerShell unterstützten Betriebssystemen? (Windows 7/8/8.1/10/11, Windows Server 2008/2008 R2/2012/2012 R2/2016/2019/2022, macOS, Linux)	○ 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 1400 Seiten nicht alle Details erläutern, sondern – in dem Sinn „Hilfe zur Selbsthilfe" – nur **ausgewählte Aspekte** darstellen kann, anhand deren Sie dann Ihre eigenen Lösungen für Ihre spezifischen Szenarien entwickeln?	○ 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 Auflage zur PowerShell 5.1/PowerShell 7.0 wurde das Buch um die neuen Commandlets, Funktionen und Operationen in PowerShell 7.1 und 7.2 erweitert.

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

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

Sind in diesem Buch alle Features der PowerShell beschrieben?

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

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

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

Wie aktuell ist dieses Buch?

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

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

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

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

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

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

Welche PowerShell-Versionen werden besprochen?

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

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

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

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

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

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

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

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

Welche Betriebssysteme werden besprochen?

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

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

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

Woher bekommt man die Beispiele aus diesem Buch?

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

In diesem Portal können Sie

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

Kurzreferenz ("Cheat Sheet") Windows PowerShell

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

www.IT-Visions.de
Dr. Holger Schwichtenberg

Hilfe

Alle installierten Module
Get-Module -ListAvailable | ft 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. Dateninhalt)
Set-Content (sc)	Elementinhalt festlegen
Add-Content (ac)	Elementinhalt ergänzen
New-Item (ni, mkdir)	Erstellen eines Elements (Ast oder Blatt)
Get-ItemProperty (gp)	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

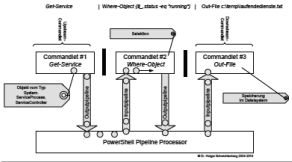
Get-Date / Set-Date	Datum und Zeit abrufen/festlegen
Get-Service	Windows-Systemdienste
Start-/Stop-/Suspend-/Resume-Service	Dienststatus ändern
Get-Process	Laufende Prozesse
Start-Process/Stop-Process	Prozess starten/beenden
Wait-Process	Warten auf Ende eines Prozesses
Get-Counter	Leistungsindikatoren 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 aber ungewöhnliche Operatoren zum Einsatz.

Vergleich unter Ignorierung der Groß-/Kleinschreibung	Vergleich unter Berücksichtigung der Groß-/Klein-schreibung	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

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 / -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

Für die logische Verknüpfung werden –and und –or sowie –not (alias !) verwendet.
Beispiel: *((1MB + 150 + $a) -gt 1000KB) -and !($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 (PSCX)
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="Nr"; 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 iexplore | 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 iexplore)) { $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/HS")
$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

Rückseite der PowerShell-Kurzreferenz

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

Wie sind die Programmcodebeispiele organisiert?

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

Buchteilname\Kapitelname\Dateiname

Die Namen sind zum Teil etwas verkürzt (z. B. „Einsatzgebiete“ statt „PowerShell im 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.

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


    Verzeichnis: x:\


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

Verzeichnisstruktur der Beispielsammlung mit vier Hauptordnern entsprechend den vier Buchteilen

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


    Verzeichnis: x:\1_Basiswissen


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

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

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

Warum gendern Sie nicht in diesem Buch?

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

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

Wie wurde die Qualität gesichert?

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

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

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

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

Wo kann man Verbesserungsvorschläge melden?

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

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

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

Wann wird die nächste Auflage erscheinen?

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

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

Wo kann man sich schulen oder beraten lassen?

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

Wem ist zu danken?

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

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

Zum Schluss dieses Vorworts . . .

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

Dr. Holger Schwichtenberg

Essen, im Sommer 2022

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 40 000` ruft aus dem aktuellen Objekt die Eigenschaft `Length` ab und vergleicht, ob diese größer (`-gt`) als 40 000 ist. `Select-Object` beschneidet alle Attribute aus `Name` und `Length`. Durch das vierte Commandlet in der Pipeline wird die Ausgabe nach dem Attribut `Length` sortiert. Das letzte Commandlet schließlich erzwingt eine Listendarstellung.

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

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

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

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

5.3 .NET-Objekte in der Pipeline

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

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

In einer Pipeline wie

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

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

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

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

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

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

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

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

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

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

5.4 Pipeline Processor

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

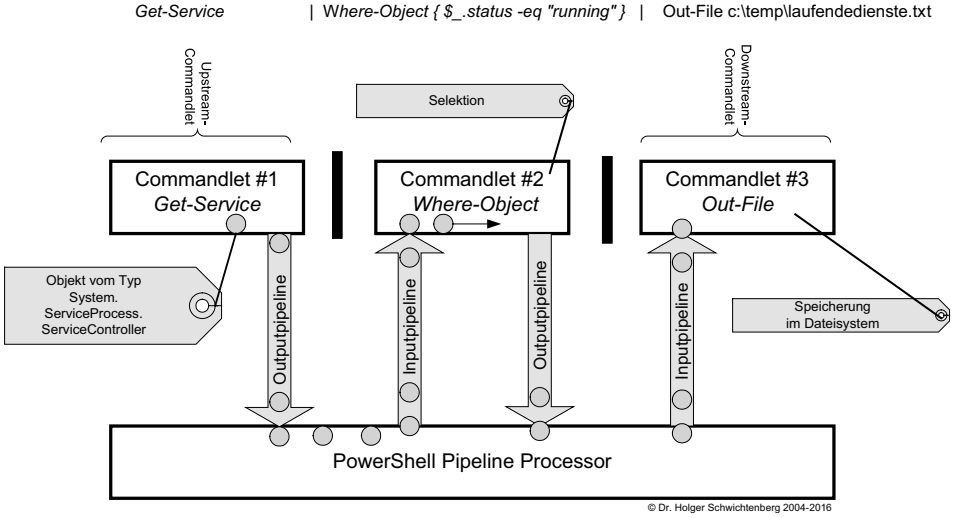


Bild 5.1 Der Pipeline Processor befördert die Objekte vom Upstream-Commandlet zum Downstream-Commandlet. Die Verarbeitung ist in der Regel asynchron.

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

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

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

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

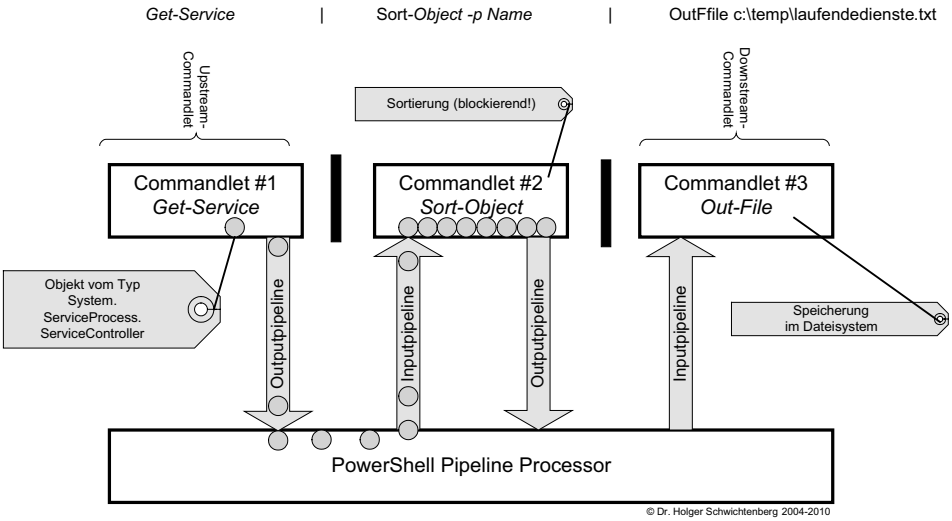


Bild 5.2 Sort-Object blockiert die direkte Weitergabe. Erst wenn alle Objekte angekommen sind, kann das Commandlet sortieren.

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

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

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

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

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

5.5 Pipelining von Parametern

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

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

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

```
-Include <string[]>
    Retrieves only the specified services. The value of this parameter 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

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

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

Im Fall von

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

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

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

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

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

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

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

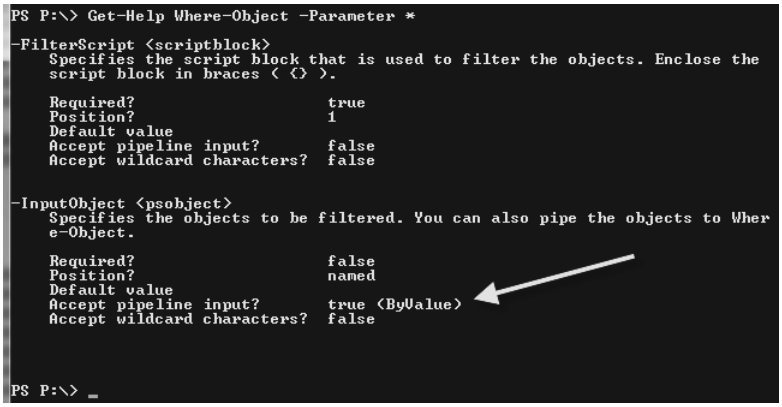


Bild 5.4 Parameter des Commandlets Where-Object

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.

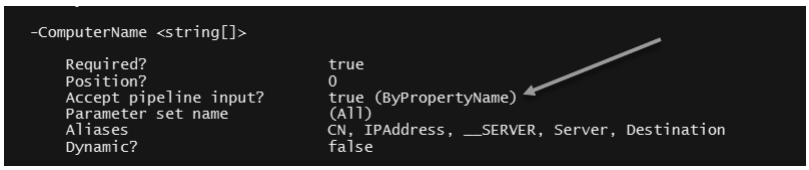


Bild 5.5 Hilfe zum Parameter ComputerName des Commandlets Test-Connection

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 im Kapitel *„PowerShell-Skriptsprache"* noch etwas näher beschrieben werden.

In dem folgenden Beispiel werden nur diejenigen Zeilen der Ausgabe von `netstat.exe` gefiltert, die einen Doppelpunkt gefolgt von den Ziffern 59 und zwei weiteren Ziffern enthalten.

Die Hervorhebung der Treffer durch Negativschrift gibt es erst seit PowerShell 7.0.

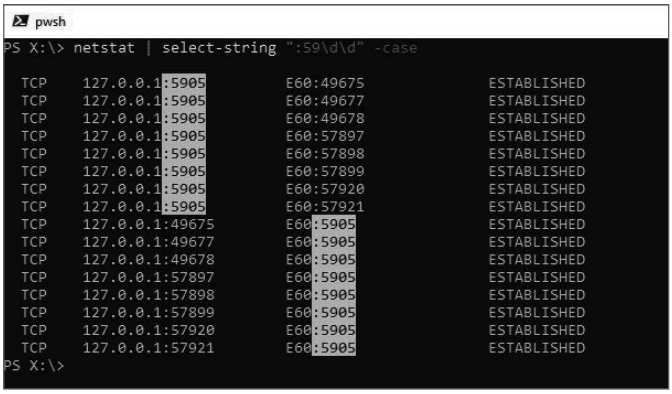


Bild 5.6
Einsatz von Select-String zur Filterung von Ausgaben klassischer Kommandozeilenwerkzeuge

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
```

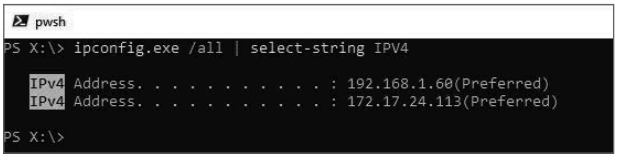
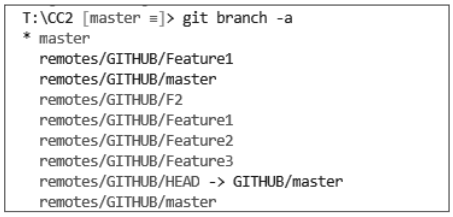


Bild 5.7
Abbildung: Ausführung des obigen Befehls

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.



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

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

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

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

oder

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

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

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

5.7 Zeilenumbrüche in Pipelines

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

- Man beendet die Zeile mit einem Pipe-Symbol [|] und drückt EINGABE. PowerShell-Standardkonsole und PowerShell-ISE-Konsole erkennen, dass der Befehl noch nicht abgeschlossen ist, und erwarten weitere Eingaben. Die Standardkonsole zeigt dies auch mit >>> an.
- Man kann am Ende einer Zeile mit einem Gravis [`], 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 Hauptthread der PowerShell, d. h., die einzelnen Durchläufe erfolgen nacheinander. Seit PowerShell 7.0 kann man mit dem Parameter `-parallel` die Ausführung auf verschiedene Threads parallelisieren (via Multithreading), sodass bei längeren Operationen in Summe das Ergebnis schneller vorliegt.

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

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

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

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

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

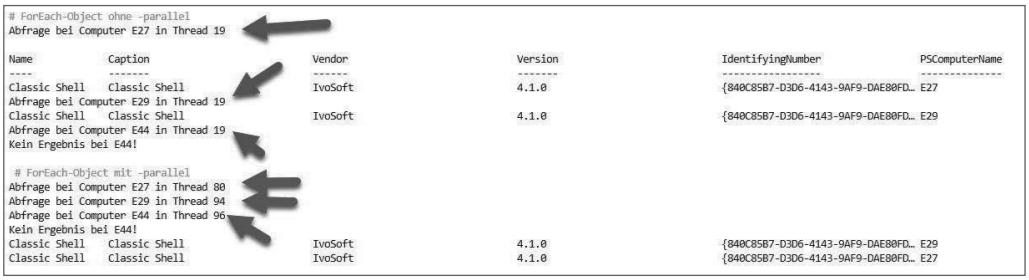


Bild 5.9 Parallelität bei Foreach-Object in PowerShell 7

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

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

5.9 Zugriff auf einzelne Objekte aus einer Menge

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

Beispiele:

Der erste Prozess:

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

Der dreizehnte Prozess:

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

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

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

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

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

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

```
PS C:\Windows\System32> (get-process)[0]

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
     20       2     1968       2664    17     0,03   2784 cmd

PS C:\Windows\System32> (get-process)[12]

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
     69       9     1484       4196    41     0,03   2100 dlpwdnt

PS C:\Windows\System32> (get-process | where-object { $_.WorkingSet64 -gt 20mb } )[12]

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    685      29    53924      59544   291    34,39   4984 powershell

PS C:\Windows\System32>
```

Bild 5.10 Zugriff auf einzelne Prozessobjekte

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
```

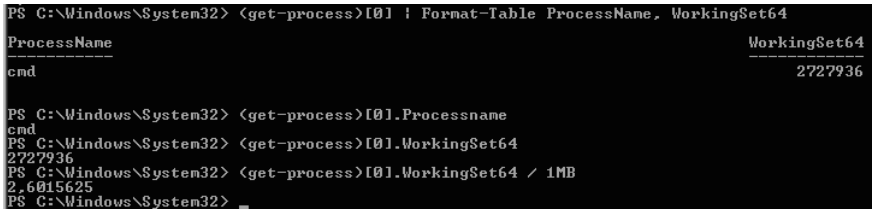


Bild 5.11 Ausgabe zu den obigen Beispielen

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

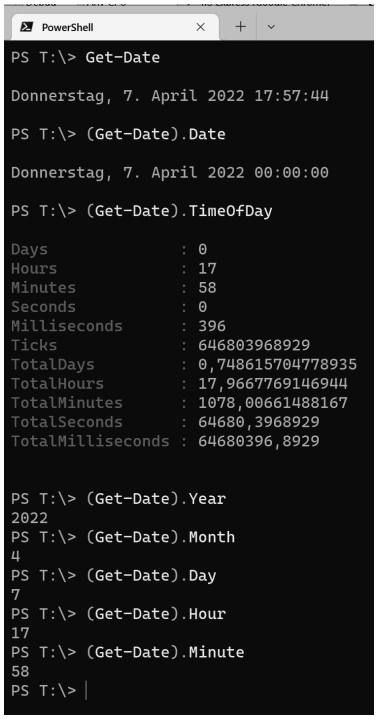


Bild 5.12
Zugriff auf einzelne Werte aus dem aktuellen Datum/der aktuellen Zeit

5.10.2 Null-Werte

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

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

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

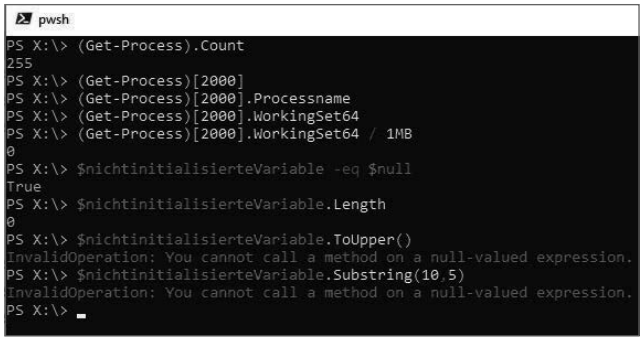


Bild 5.13
Null-Werte in der PowerShell

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.

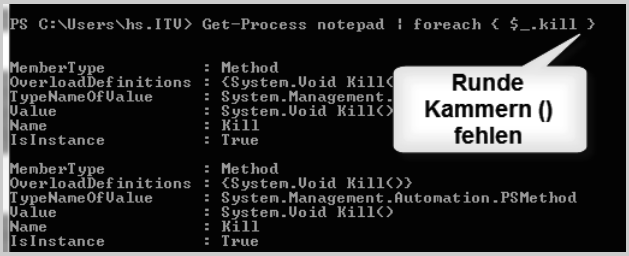


Bild 5.14
Folgen des vergessenen Klammernpaars

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 folgende Abbildung.

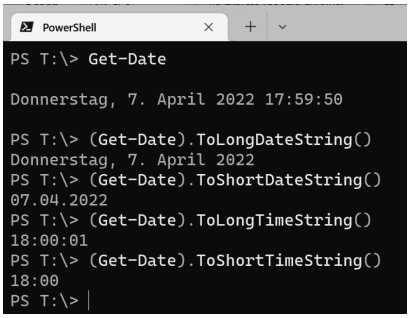


Bild 5.15
Ausgaben der Methoden der Klasse DateTime

5.12 Analyse des Pipeline-Inhalts

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

- Wie viele Objekte sind in der Pipeline? (Das wurde schon zuvor in diesem Kapitel erörtert.)
- Welchen Typ haben die Objekte, die ein Commandlet in die Pipeline legt?
- Welche Attribute und Methoden haben diese Objekte?

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

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

Bei anderen Commandlets aber heißt es nur wenig hilfreich:

```
OUTPUTS
   Object
```

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

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

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

5.12.1 Anzahl der Objekte in der Pipeline mit Count und Length

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

Es kann aber auch sein, dass ein Commandlet, das normalerweise eine Menge von Objekten liefert, im konkreten Fall (z. B. bei Einsatz eines filternden Parameters) nur ein einzelnes Objekt liefert (z. B. `Get-Process idle`). In diesem Fall liefert die PowerShell dem Benutzer nicht eine Liste mit einem Objekt, sondern direkt das ausgepackte Objekt.

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

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

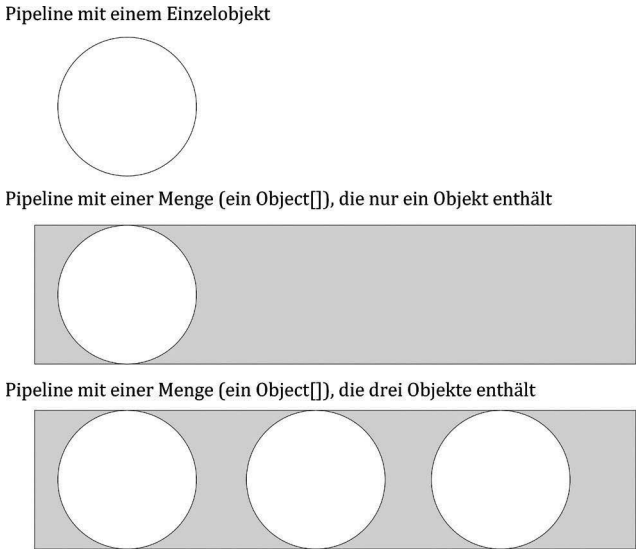


Bild 5.16
Einzelobjekt versus Menge

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 100 MB Hauptspeicher (RAM) verbrauchen?

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

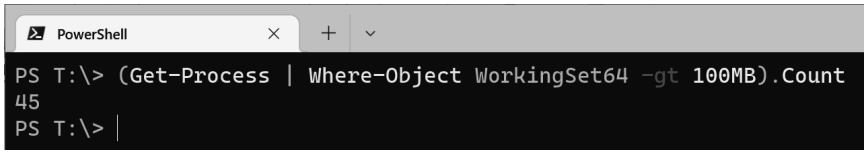


Bild 5.17 Aufruf von Count für eine Pipeline

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

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

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

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

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

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

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

```
# Workaround für Anomalie: Das vorangestellte Komma macht aus dem Einzelobjekt eine
Menge mit einem Objekt.
$firmen = , $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 Abbildung:

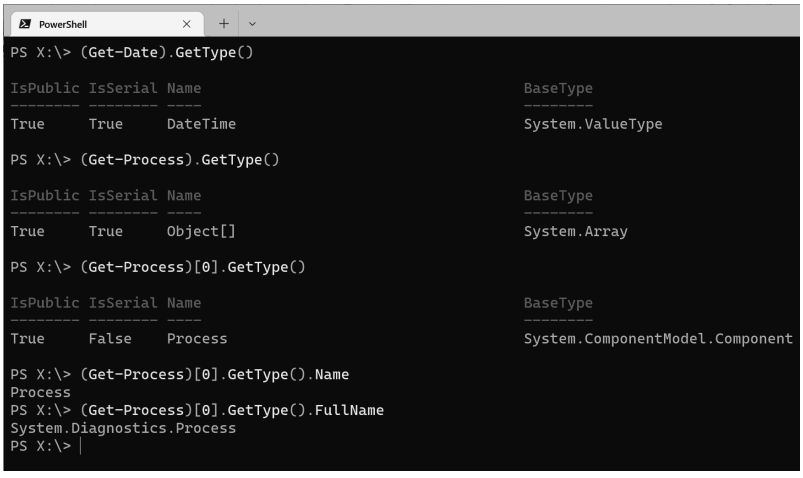


Bild 5.18 Einsatz von GetType()

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 5.3 [Basiswissen\Pipelining\ToString.ps1]

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

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

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

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

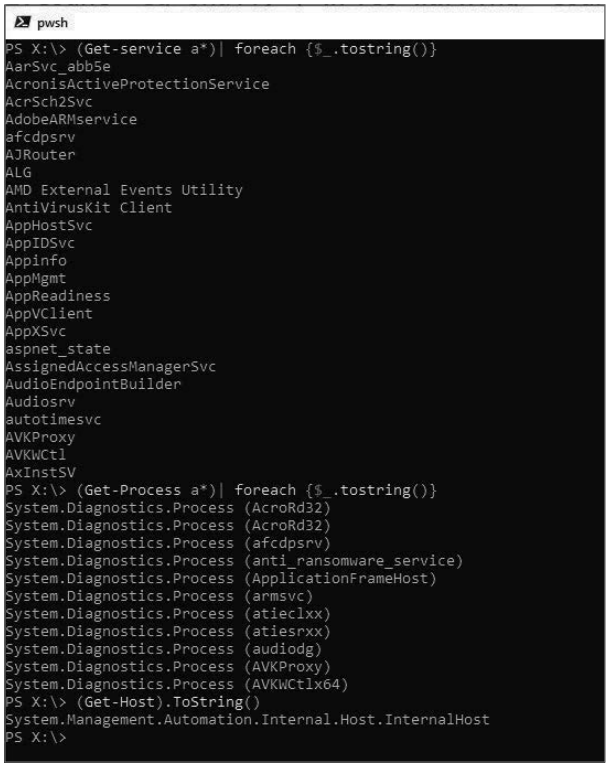


Bild 5.19
Anwendung von ToString() auf Instanzen verschiedener Klassen

5.12.4 Get-PipelineInfo

Das Commandlet `Get-PipelineInfo` aus den PowerShell Extensions von *www.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

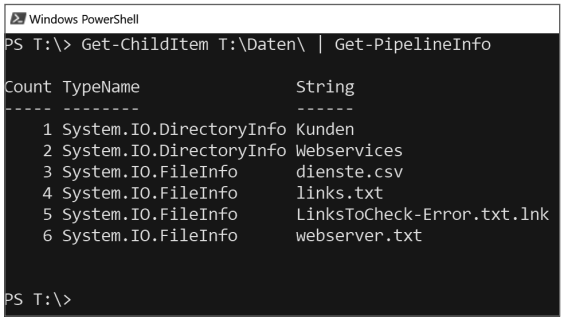


Bild 5.20
Get-PipelineInfo liefert Informationen, dass sich in dem Dateisystemordner elf Objekte befinden. Davon sind sieben Unterordner (Klasse DirectoryInfo) und vier Dateien (Klasse FileInfo).

Das Stichwort Zeichenkettenrepräsentation (Spalte „String“ in der Abbildung) ist erklärungsbedürftig: Dies ist die Zeichenkettenrepräsentation mit `ToString()`

5.12.5 Get-Member

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

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

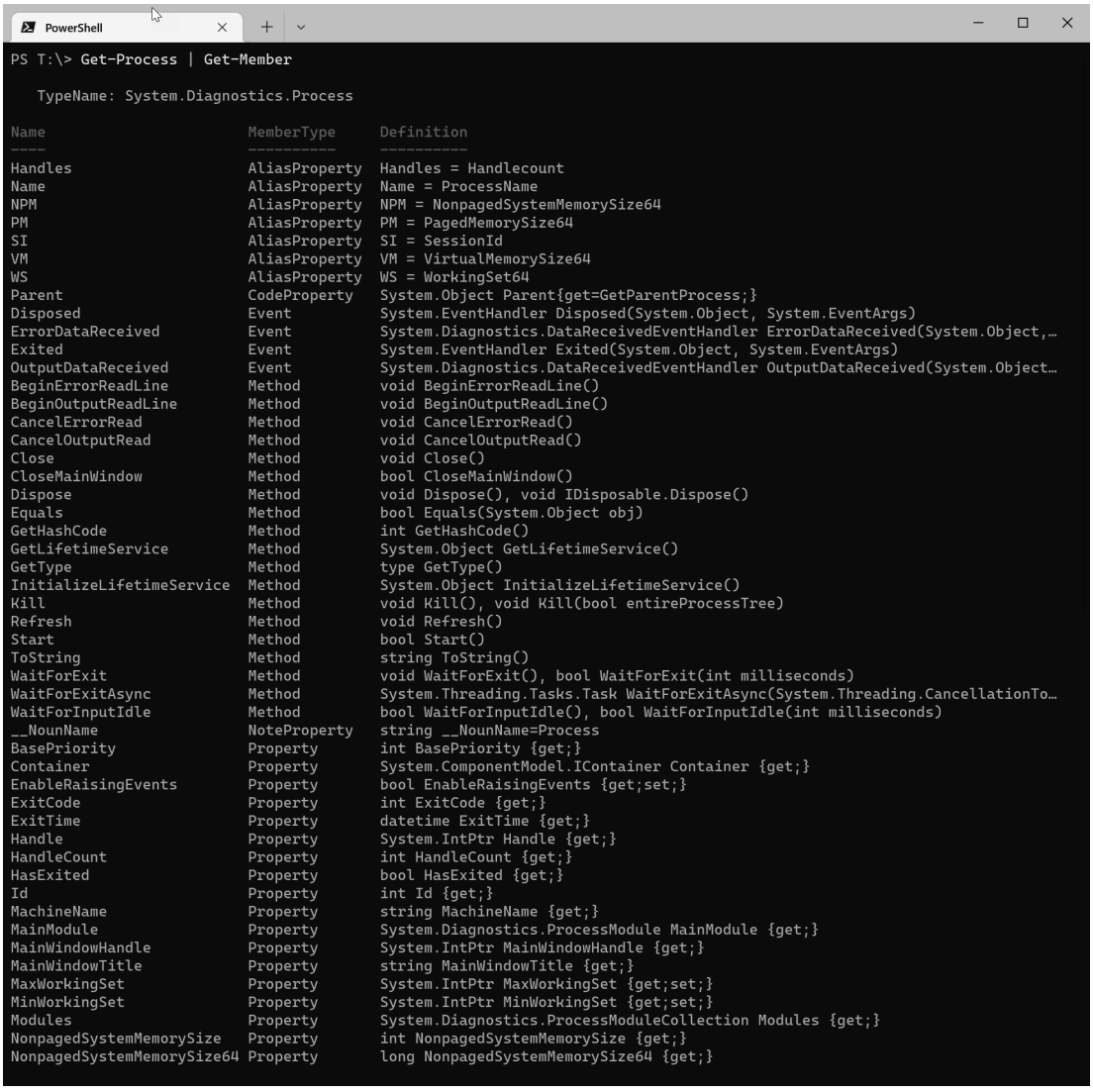


Bild 5.21 Teil 1 der Ausgabe von Get-Process | Get-Member

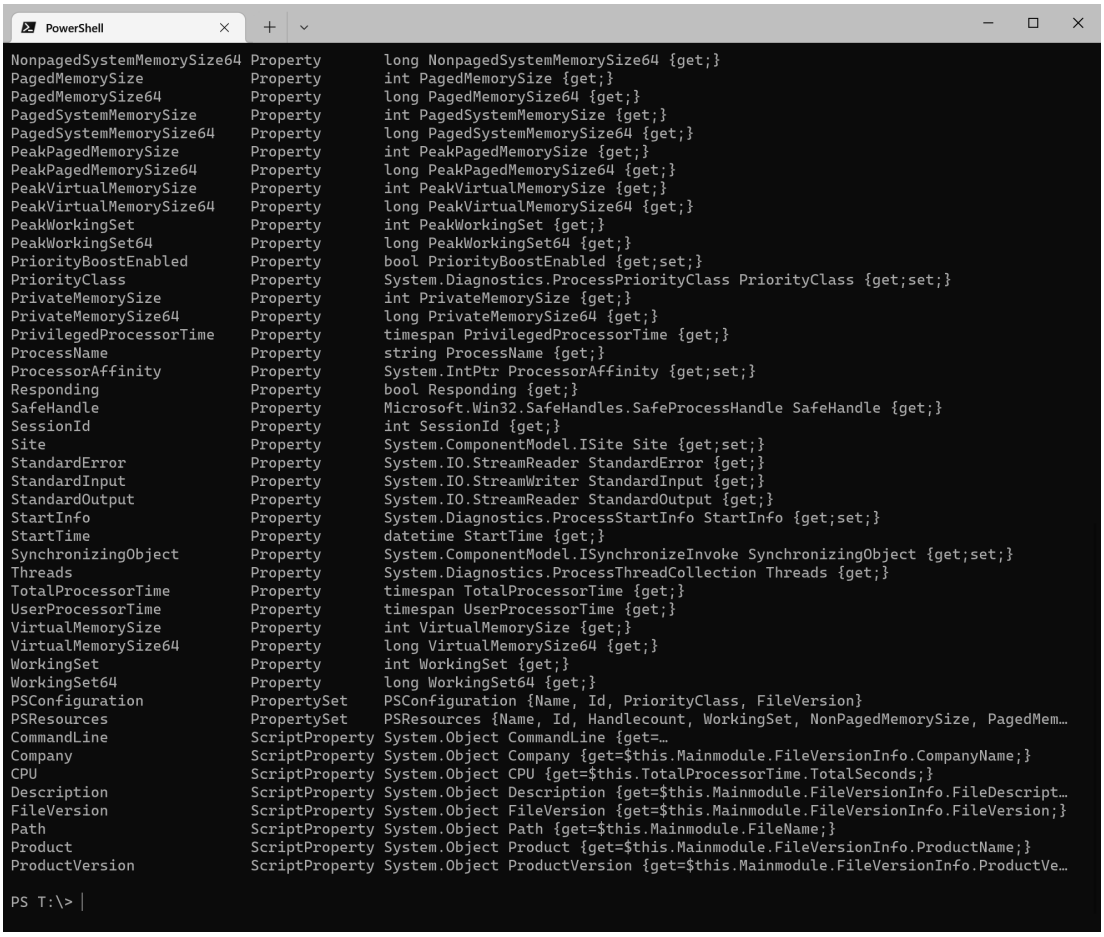


Bild 5.22 Teil 2 der Ausgabe von Get-Process | Get-Member

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

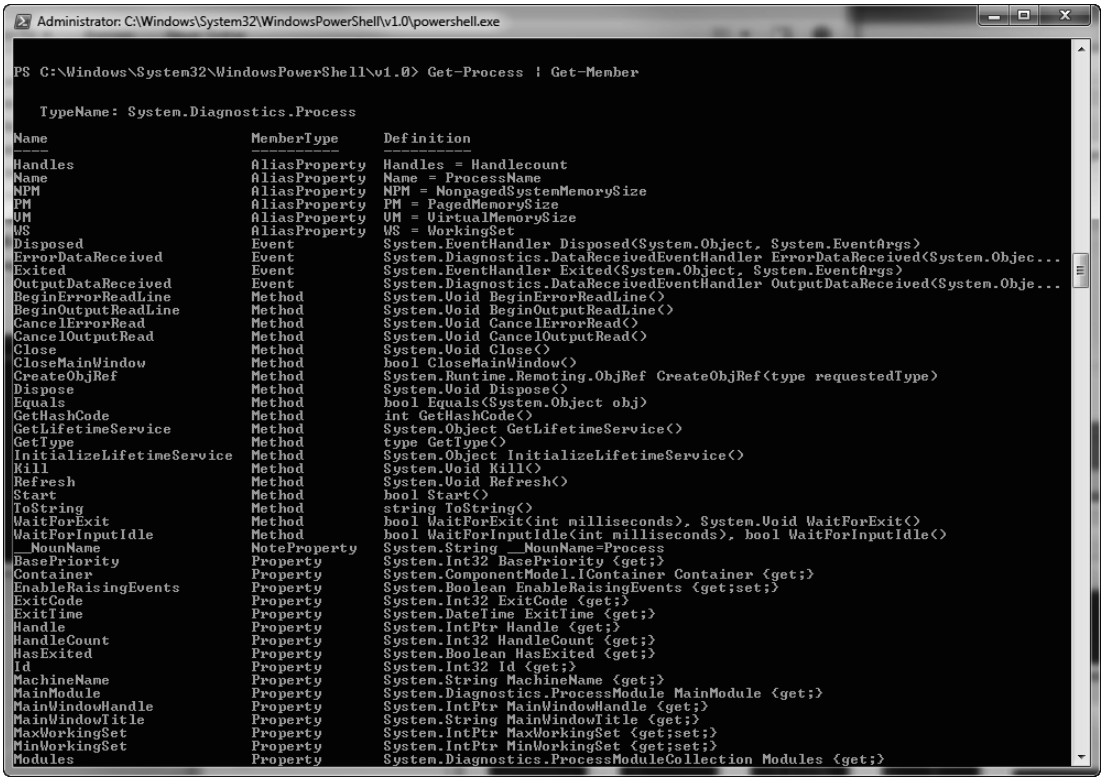


Bild 5.23 Anzeige der Getter und Setter in PowerShell 1.0

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"
```

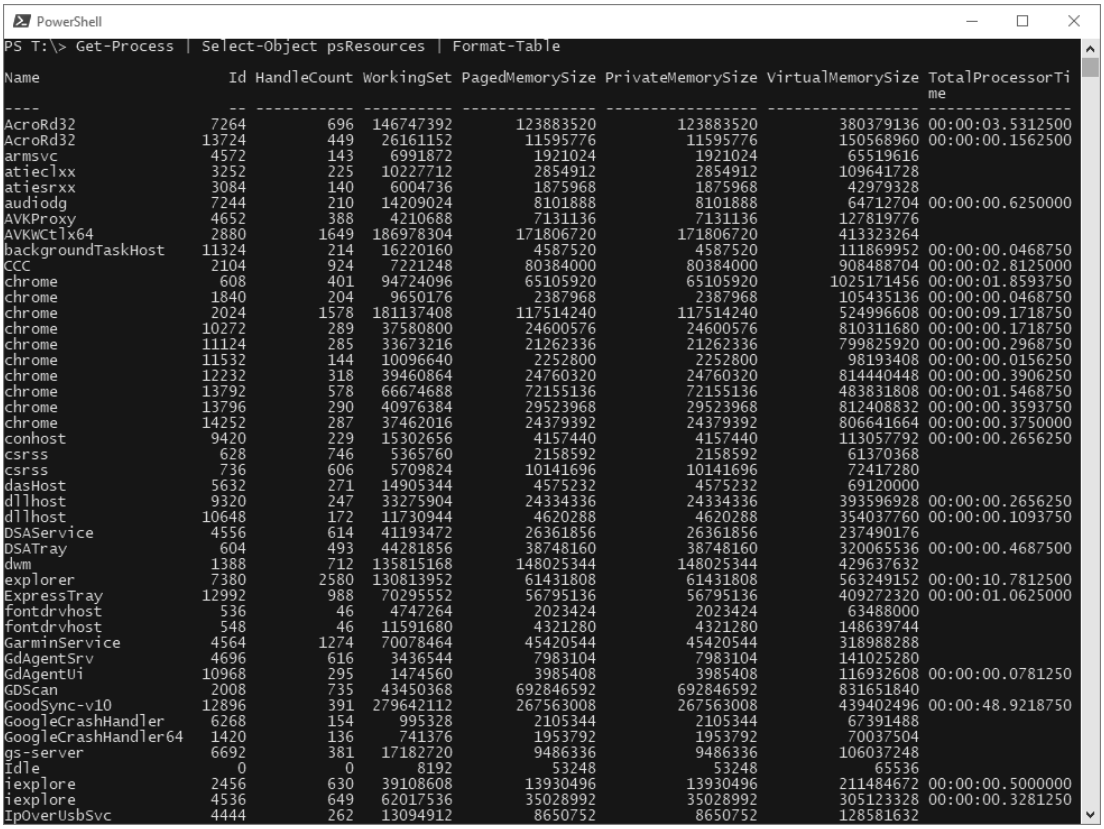
Hier geht nur die o. g. Schreibweise mit `Foreach-Object`.

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.



```
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 das Kapitel „*Dynamische Objekte*“.

5.12.10 Skripteigenschaften (ScriptProperty)

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

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

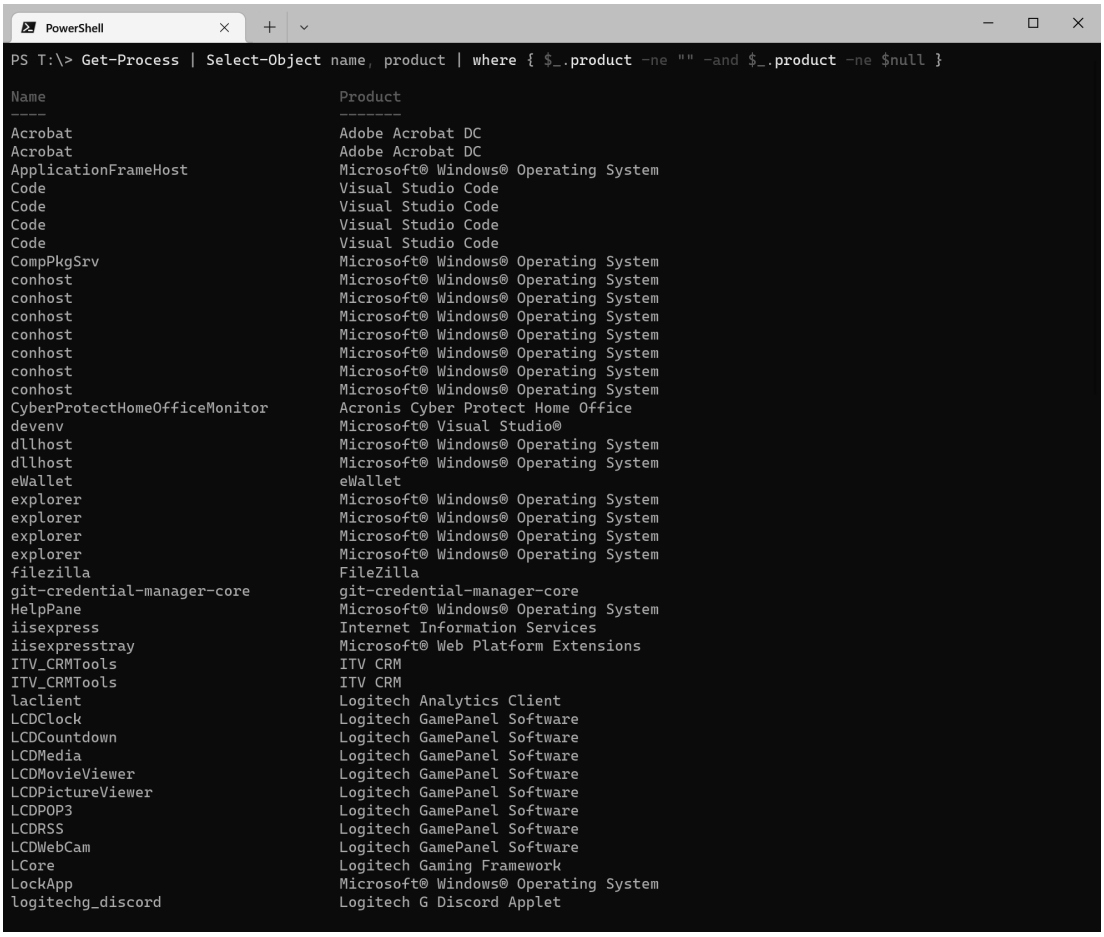
listet alle Prozesse mit den Produkten auf, zu denen der Prozess gehört (siehe folgende Abbildung). Dies ist gut zu wissen, wenn man auf seinem System einen Prozess sieht, den man nicht kennt und von dem man befürchtet, dass es sich um einen Schädling handeln könnte.

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

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

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

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



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

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

Bild 5.26 Auflistung der berechneten Eigenschaft „Product“

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

Bild 5.27 Definition einer Skripteigenschaft in der types.ps1xml

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

5.12.11 Codeeigenschaften (Code Property)

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

5.12.12 Aliaseigenschaft (AliasProperty)

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

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

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

Als Adapted Type System (ATS) bezeichnet Microsoft die grundsätzliche Anpassung von .NET-Klassen aus der .NET-Klassenbibliothek auf die Bedürfnisse von PowerShell-Benutzern. Wie bereits dargestellt, zeigt die PowerShell für viele .NET-Objekte mehr Mitglieder an, als eigentlich in der .NET-Klasse definiert sind. In einigen Fällen werden aber auch Mitglieder ausgeblendet.

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

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

In der Dokumentation nimmt das PowerShell-Entwicklungsteam dazu wie folgt Stellung:

> - Some .NET objects are "meta" objects (for example: WMI Objects, ADO objects, and XML objects) whose members describe the data they contain. However, in a scripting environment it is the contained data that is most interesting, not the description of the contained data. ETS resolves this issue by introducing the notion of Adapters that adapt the underlying .NET object to have the expected default semantics.
> - Some .NET Object members are inconsistently named, provide an insufficient set of public members, or provide insufficient capability. ETS resolves this issue by introducing the ability to extend the .NET object with additional members.

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

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

Das ATS verpackt grundsätzlich jedes Objekt, das von einem Commandlet in die Pipeline gelegt wird, in ein PowerShell-Objekt des Typs `PSObject`. Die Implementierung der Klasse `PSObject` entscheidet dann, was für die folgenden Commandlets und Befehle sichtbar ist.

Diese Entscheidung wird beeinflusst durch verschiedene Instrumente:

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

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

Tabelle 5.2 .NET-Klassen mit ATS

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

5.13 Filtern

Nicht immer will man alle Objekte weiterverarbeiten, die ein Commandlet liefert. Einschränkungskriterien sind Bedingungen (z. B. nur Prozesse, bei denen der Speicherbedarf größer ist als 10 000 000 Byte) oder die Position (z. B. nur die fünf Prozesse mit dem größten Speicherbedarf). Zur wertabhängigen Einschränkung verwendet man das Commandlet `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 { $_.DisplayName -match "^\w+ \w+$" }
```

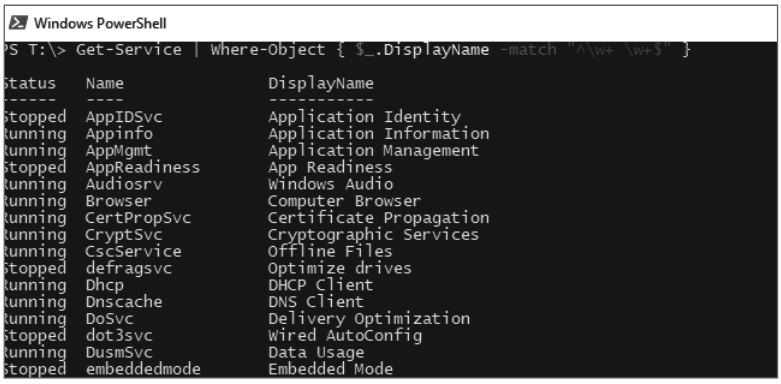


Bild 5.29 Ausgabe zu obigem Beispiel

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

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

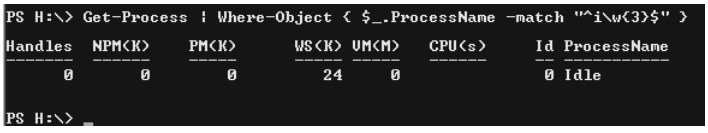


Bild 5.30 Ausgabe zu obigem Beispiel

Tabelle 5.3 Vergleichsoperatoren der PowerShell

Vergleich unter Ignorierung der Groß-/Kleinschreibung	Vergleich unter Berücksichtigung der Groß-/Kleinschreibung	Bedeutung
`-lt / -ilt`	`-clt`	Kleiner
`-le / -ile`	`-cle`	Kleiner oder gleich
`-gt / -igt`	`-cgt`	Größer
`-ge / -ige`	`-cge`	Größer oder gleich
`-eq / -ieq`	`-ceq`	Gleich
`-ne / -ine`	`-cne`	Nicht gleich
`-like / -ilike`	`-clike`	Ähnlichkeit zwischen Zeichenketten, Einsatz von Platzhaltern (* und ?) möglich
`-notlike / -inotlike`	`-cnotlike`	Keine Ähnlichkeit zwischen Zeichenketten, Einsatz von Platzhaltern (* und ?) möglich
`-match / -imatch`	`-cmatch`	Vergleich mit regulärem Ausdruck
`-notmatch / -inotmatch`	`-cnotmatch`	Stimmt nicht mit regulärem Ausdruck überein
`-is`		Typvergleich, z. B. (Get-Date) -is [DateTime]
`-in -contains`		Ist enthalten in Menge
`-notin -notcontains`		Ist nicht enthalten in Menge

Tabelle 5.4 Logische Operatoren in der PowerShell-Sprache

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

5.13.2 Vereinfachte Schreibweise von Bedingungen seit PowerShell 3.0

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

Die Bedingung

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

kann der Nutzer seitdem vereinfacht schreiben als

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

Dass auch

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

und

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

zum gleichen Ergebnis führen, wirkt 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 5.31
Gespaltene Ausgabe der Zufallszahlen

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

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

5.14 Zusammenfassung von Pipeline-Inhalten

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

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

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

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

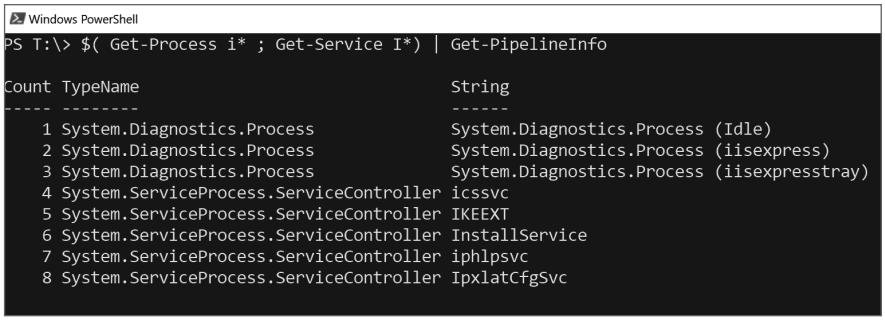


Bild 5.32 Anwendung von Get-PipelineInfo auf eine heterogene Pipeline

5.15 „Kastrierung“ von Objekten in der Pipeline

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

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

Beispiel:

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

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

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

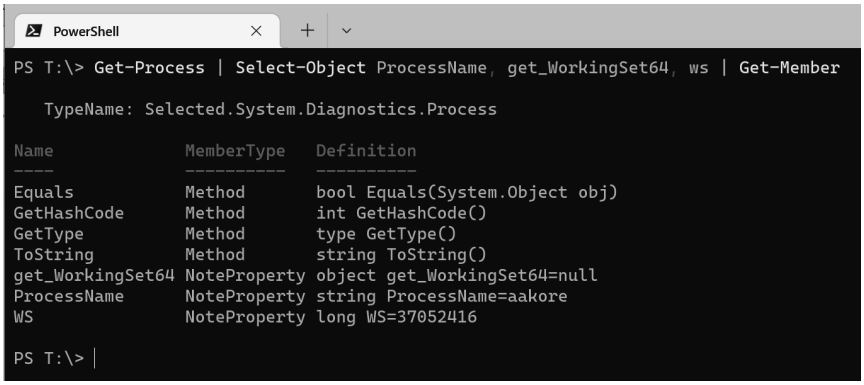


Bild 5.33 Wirkung der Anwendung von Select-Object

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

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

5.16 Sortieren

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

Der folgende Befehl sortiert die Prozesse absteigend nach ihrem Speicherverbrauch:

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

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

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

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

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

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

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

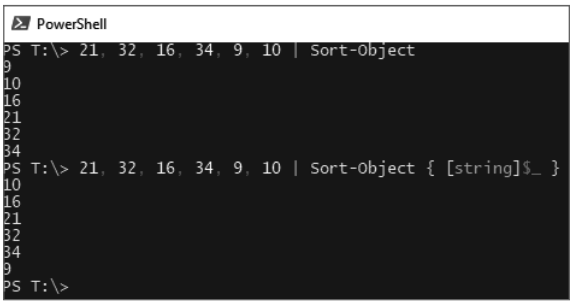


Bild 5.34
Numerische versus alphabetische Sortierung von sechs Zahlen

5.17 Duplikate entfernen

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

Beispiel

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

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

ACHTUNG: Bei Get-Unique muss die Liste vorher sortiert sein!

Richtig ist daher:

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

Falsch wäre:

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

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

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

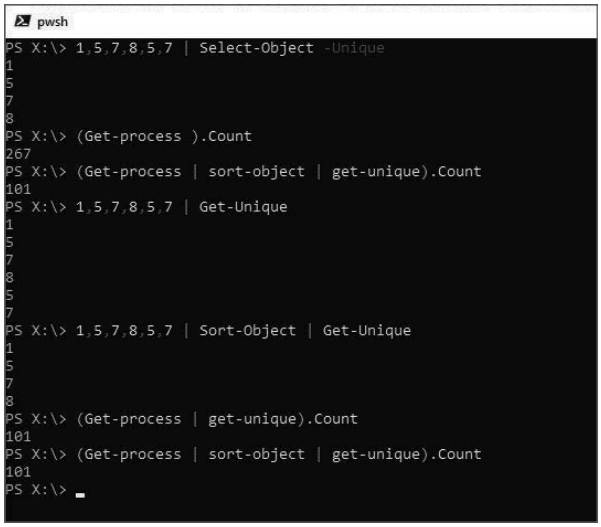


Bild 5.35
Einsatz von Get-Unique und Select-Object -unique

Praxislösung: Microsoft-Office-Wörterbücher zusammenfassen

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

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

5.18 Gruppierung

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

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

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

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

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

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

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

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

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

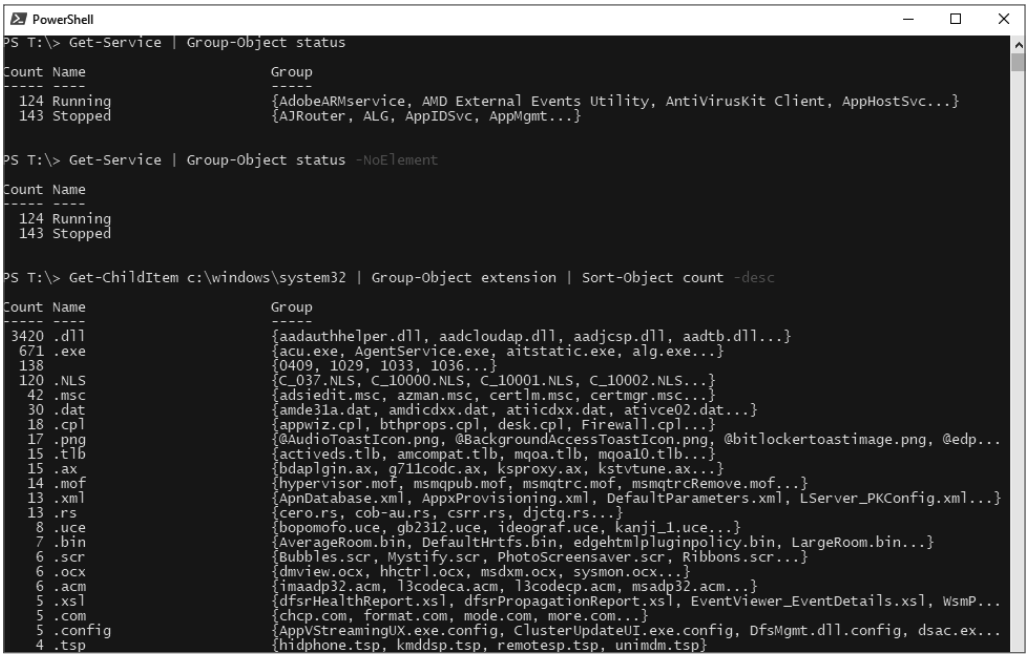


Bild 5.36 Einsatz von Group-Object

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

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

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

BEISPIEL:

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

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

```
PS C:\Users\hs.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 5.37 Ergebnis des obigen Befehls (Zahlen können in Abhängigkeit vom Betriebssystem abweichen)

Praxislösung 1

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

Listing 5.4 [Finde doppelte Zeichenketten.ps1]

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

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

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

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

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

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

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

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

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

    $Duplikate  | Ft Name, Count
    }
```

Praxislösung 2

Wenn man sich die Elemente der einzelnen Gruppen 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 angebeben 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.

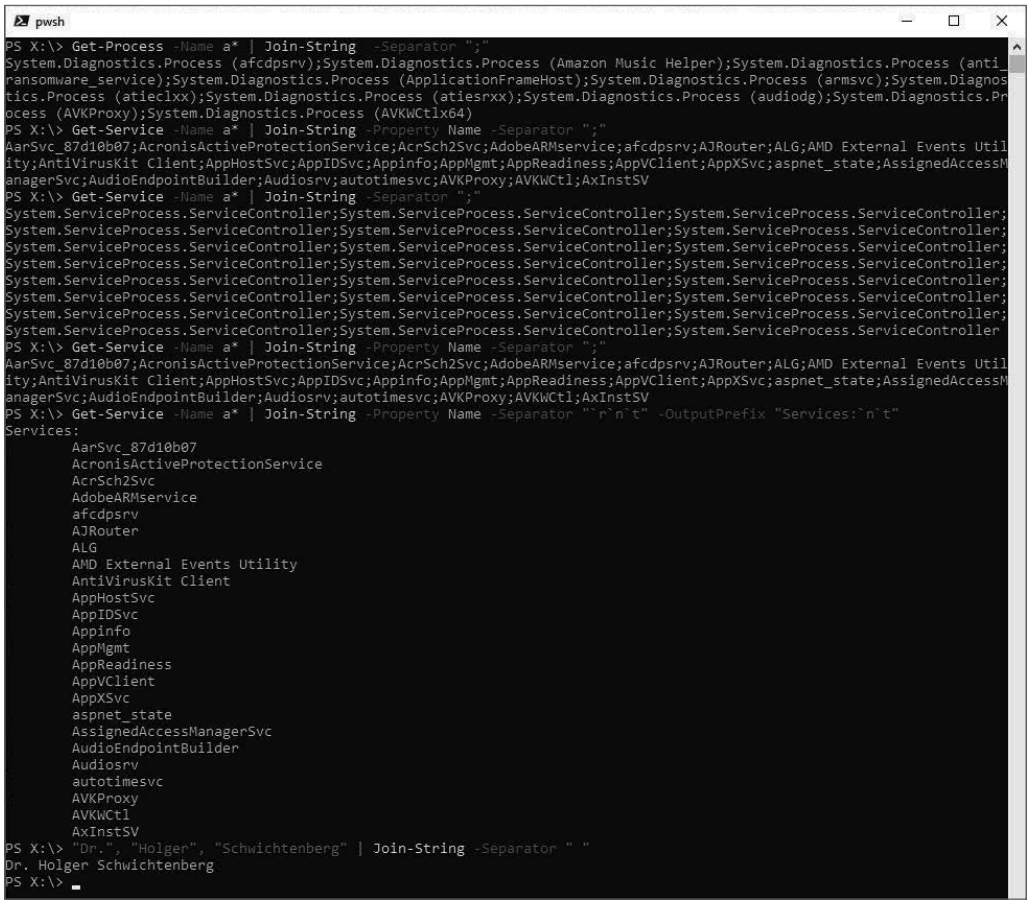


Bild 5.38 Einsatzbeispiele für Join-String

5.20 Berechnungen

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

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

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

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

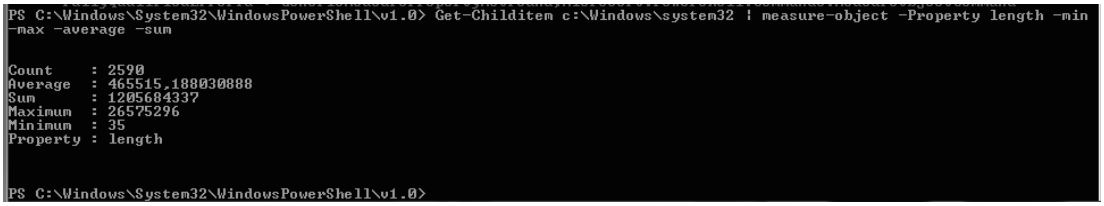


Bild 5.39 Beispiel für den Einsatz von Measure-Object

5.21 Zwischenschritte in der Pipeline mit Variablen

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

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

kann man die folgenden Befehle nacheinander in getrennte Zeilen eingeben:

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

Das Ergebnis ist in beiden Fällen gleich.

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

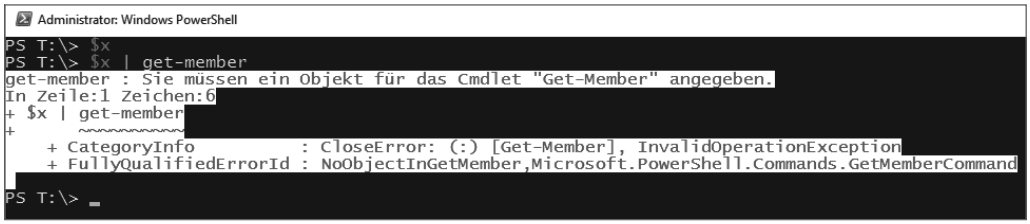


Bild 5.40 Zugriff auf Variablen ohne Inhalt

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

Beispiel:

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

Die Ausgabe für `$null` ist nichts.

5.22 Verzweigungen in der Pipeline

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

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

Tee-Object

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

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

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

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

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

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

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

Parameter -OutVariable

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

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

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

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

Parameter -PipelineVariable

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

Beispiel 1

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

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

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

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

Beispiel 2

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

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

5.23 Vergleiche zwischen Objekten

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

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

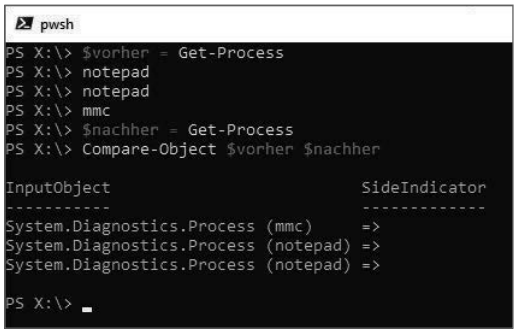


Bild 5.41
Vergleich von zwei Pipelines

5.24 Weitere Praxislösungen

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

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

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

  oder synonym und kürzer:

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

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

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

- Gib die Summe der Speichernutzung aller Prozesse aus.

```
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

?? 176
?. 176 ff.
& 80
&& 231
% 108
> 266
>> 266
|| 231
$_ 99 f., 108, 114, 181, 500
$? 231 ff.
$ConfirmPreference 1283
$ErrorView 229
$null 113, 128, 156, 218, 805
$PSItem 108
$PSStyle 316, 392
$psUnsupportedConsoleApplications 335
$psversiontable 393
$PSVersionTable 27
-and 134
-as 178
-band 213, 440
-Bit 21
-bnot 213, 440
-bor 213, 440
-bxor 213
.cat 716
-cmatch 197
-cnotmatch 197
.dll 61, 166, 431, 703, 1347 f.
.exe 166, 348
-expression 686
–force 66
-imatch 197
-inotmatch 197
-ItemsSource 1255
-Join 196
-match 197
-notmatch 197
-or 134
-Parameter 87
-Split 195 f.
-Verbose 66, 608
.NET 3, 17, 46, 100, 181, 204, 422, 509, 1242, 1296, 1324, 1339, 1371
– Bibliothek 415
– Klasse 415, 1377
– Runtime Host 15
.NET 6.0 45 ff., 1371
.NET 7.0 45, 1371
.NET API Portability Analyzer 1126
.NET CLI 51
.NET Core 17, 376, 1152, 1180, 1371, 1375
.NET Core 3.1 379
.NET Core SDK 51
.NET Data Provider 793 f.
.NET Framework 4, 15, 279, 550, 630, 894, 1135, 1371, 1374, 1378
– 4.0 20
.NET Standard 418, 1376
.nupkg 435
.pfx 532
.pkg 54
.ps1 28, 76, 152, 1348
.psd1 596, 637, 1270, 1348, 1353
.psm1 1270, 1343 ff., 1348, 1353
.psproj 346
-wait 608
.yml 1181
32-Bit 21, 330, 799, 856
64-Bit 799, 856
[Type] 427

A

Ablaufverfolgung 3, 515 f.
About 168
Absent 614
AbsoluteTimerInstruction 464
abstract 1368
Accelerator 172
Accent Grave
– Gravis 68
AccessControl 973, 978
Access Control Entry 974
Access Control List 974, 986
Access Control Type 974
Access Mask 974
AccessMask 730
AccountDisabled 1021
AceFlags 974
ACL 977, 991
ACR 1152
Active Directory 3, 271, 450, 604, 651, 666, 1010, 1017, 1021, 1036, 1066 ff., 1271
– PowerShell 1036
– Struktur 1068
– Suche 1026
Active Directory Application Mode 1068
Active Directory Domain Services 1065
Active Directory Service Interface siehe ADSI 1080
ActiveScriptEventConsumer 465
Active-Scripting 158
ActiveX Data Objects 793, 804, 1007
ADAccount 1042
Adapted Type System siehe ATS 483
ADComputer 1042
Add() 430
Add-ADGroupMember 1046, 1064
AddCommand() 1361
Add-Computer 863
Add-Content 749, 783
Add-DirectoryEntry 666
Add-DistributionGroupMember 1084
Add-Feature 745
Add-JobTrigger 545
Add-LDAPObject 1034, 1272 f.
Add-LocalGroupMember 1079
Add-Member 131, 497, 500, 1281
Add-Migration 351
Add-ODBCDSN 844
Add-PSSnapin 376, 1300, 1305
AddScript() 1358, 1361
ADDSDeployment 1065 ff.

Add-Type 432 f., 501, 508, 717, 829, 962
Add-VirtualHardDisk 669
Add-VMDisk 1122
Add-VMDrive 1122
Add-VMHardDiskDrive 1099, 1111
Add-VMNIC 1122
Add-VMSwitch 1099
Add-WBSystemState 744
Add-WindowsCapability 905
Add-WindowsFeature 743, 894, 898 ff., 1065 f.
Administration
– delegiert 672
– webbasiert 355, 677
Administrator 160, 303, 973
Administratorrechte 280, 300, 303, 319 ff., 489, 641, 737, 917, 973, 1149, 1155
ADODB.Connection 1008
ADO.NET 793, 801, 1007, 1027
ADPowerShell 1036, 1042
ADSI 1004, 1007 ff., 1014, 1080
– Bindung 1009 f.
– COM 1007, 1014
– Container 1015
– .NET 1001 ff., 1007
– Pfad 1009
AdsPath 1026
ADUser 1042
Advanced Function 1267, 1276
ADWS 1038
AgentPC 1003
AKS 1197
Akte X 1002
Aktivierung 671
Aktivität 557, 561
Alias 57, 70, 269, 689
Aliaseigenschaft 123, 130
AliasInfo 70
AllNodes 611
AllowClobber 36, 641
AllowEmptyCollection 1276
AllowEmptyString 1276
AllowNull 1276
AllowPrerelease 39
AllSigned 159, 529
Alpine 13
Alvin Kersh 1002
Amazon Web Service siehe AWS
Änderungshistorie 798
Animation 1255
Ankerelement 199
ANSI-Terminal 316
Anwendungspool 1094 ff.
Anzeigesprache 596
Apache 1152
AppDomain 433, 829
AppendChild(). 762
Apple Software Package 54
AppLockerPolicy 910 f.
appSettings 357
AppX 888
Args 156, 181, 289
Array 205, 208 ff., 1380
ArrayList 208
Artifact 1222
AsJob 535, 554
ASP.NET 450, 1135, 1178
ASP.NET Core 371
Assembly 417, 420, 431, 630, 635, 703, 1305, 1335
– verbreiten 1379
AssocClass 923
ASSOCIATORS OF 466
Assoziation 458
– WMI 455, 458
Asynchronous 964
ATS 130, 483, 1008
Attribut 1363, 1368, 1380
– indiziert 1380
Audio 426
Aufgabe
– geplant 540
Aufzählung 439
Aufzählungstyp 309
Ausdruck 78
– Regulär 197
Ausdruckauflösung 189
Ausdrucksmodus 78
Ausführungsrichtlinie 158
Ausgabe
– mehrspaltig 251
– unterdrücken 264
Ausgabeobjekt 1309
Auslagerungsdatei 671
Authentifizierung 509, 1021, 1065
AuthorizationRuleCollection 979 f.
AutoUpdate 865
AWS 386
Az 1190
az aks 1203
az.cmd 1192, 1202
az extension 1203
Az.Tools.Predictor 314
Azure 79, 371, 386
– Kontext 1193
– Kubernetes Services 1197
– Resource Group 1188, 1194
– SQL Server 1196
– Subscription 1188, 1193
– Web-App 1194
Azure CLI 1192, 1202
Azure Cloud Shell 360
Azure Container Registry 1152
Azure Container Registry siehe ACR 1152
Azure DevOps 167, 1221 f., 1236
Azure DevOps CLI 1222
Azure Kubernetes Services siehe AKS
AzureRM 1190
Azure SQL 1196
Azure Subscription 1188
Azure-Webportal 1188

B

BackgroundColor 182, 336, 359
Background Intelligent Transfer Service 963
Background Intelligent Transfer Service siehe BITS 963
Backspace 191
Backup 743 f., 828
Backup-GPO 1073
Backup-SqlDatabase 826 ff.
Base 1026, 1048
bash 312, 360, 371, 398, 1141
BasicHtmlWebResponseObject 950
Basisauthentifizierung 277
Basisimage 1180 f.
Basisklasse 797
Batterie 873
Bedingung 219
Beep 191
Beep() 429
Befehl
– Extern 57, 79
Befehls-Add-On 89
Befehlseingabefenster 41
Befehlsgeschichte 678
Befehlsmodus 78
Befehlsobjekt 802
begin 1267
Begin 719
BeginProcessing() 1299, 1303
Benutzer 450, 1031, 1364, 1367
– Active Directory 1017
– anlegen 1020
– lokal 1079
– löschen 1022
– umbenennen 1022
– verschieben 1023
Benutzerabmeldung 683
Benutzeranmeldung 683
Benutzerdaten lesen 1051
Benutzer-DSN 846
Benutzereingabe 506
Benutzergruppe 1064
Benutzerkennwort 1021
Benutzerkontensteuerung siehe UAC
Benutzerkonto 1051

Benutzername 509
Benutzerschnittstelle 448
Berechnung 146
Best Practice 999
Beta 40
Betriebssystembasis-Image 1123
Bezeichner 1378
Beziehung 1367
Bibliothek 1375
Big Endian 1026
Bild 665
Bildschirmschoner 503, 1003
Binärdatei 783
Binäre Operation 213
Binärmodul 1343
Bindung
– ADSI 1009
– serverlos 1010
– WMI 473
Bing 1250
BIOS 449
Bitflag 213, 439
BitLocker 745
– Überblick 745
Bitmap 717f.
BITS 59, 963, 966
Bitweise Operation 213
Bitweises NOT 213
Bitweises ODER 213
Bitweises UND 213
Blatt 1007
Blockierung 162
BMC 448
Board 1222
Boolean 231
Boot-Konfiguration 449
Bootstrap 436
break 214ff., 226, 234
Breaking Change 601
Build 1235
bxor 702, 1081
Bypass 159
ByPropertyName 104f.
Byte 184
ByValue 104f.
BZIP2 728

C

C# 4, 168, 501ff., 1222, 1267, 1296, 1303, 1371
C++ 1376f.
C++/CLI 1296
CAB 93
Canvas 1252
Carriage Return 191
cat 398ff.
CategoryView 229
CATID 467
CD 431
Certificate
– Zertifikat 992
ChangeAccess 737
Checkpoint-Computer 869
Checkpoint-VM 1098, 1113
Children 1007
ChildSession 294
Chkdsk() 491
CHKDSK 450
chmod 399
Chocolatey 892
Chocolatey.org 891
chown 403
Chrome 603, 891f.
CIL 15, 1374
CIM 8, 448, 451
– Repository 459
CimClass 472ff., 484
CimClassProperties 484
CIM Explorer 367
CimInstance 472ff., 484
CimInstanceProperties 484
CimProperty 484
CIM Query Language siehe CQL
Cisco 448
City 1043f.
class 243
ClassCreationEvent 468, 583
ClassDeletionEvent 468, 583
ClassModificationEvent 468, 583
clean 1268
Clear-BitLockerAutoUnlock-Funktion 746
ClearCase 167
Clear-Content 749
Clear-DnsClientCache 936
Clear-EventLog 278, 969
Clear-History 679
Clear-Host 228, 678
Clear-Item 689
Clear-RecycleBin 716
Clear-Variable 180
Click 1253
Clipboard siehe Zwischenablage 511
CliXml 766
Close() 1252
Cloud 1188
CLR 15, 27, 1347, 1374
cmd 323
cmd.exe 100
Cmdlet 1332
CmdletBinding 1276, 1281
Cmdlet Help Editor 1335
Cmdlets 3
cn 1017
Codeausschnitt 331, 353
Codeeigenschaft 123, 130
Color 875
COM 17, 441f., 716, 1377
– Kategorie 467
– Klasse 444
– Komponente 450
– Moniker 1009
– Sicherheit 463
Commandlet 3, 57, 69, 79, 82, 100, 275, 373
– binär 1296
– erstellen 1267, 1296
– Klasse 1299
– Konvention 1322, 1340
– Provider 270
– Proxy 1286
– Verkettung 1320
Command Line Event Consumer 466
Command Mode 78
Comma-Separated Values 753
CommitChanges() 425, 1005, 1014, 1021
Common Information Model siehe CIM
Common Intermediate Language siehe CIL 15
Common Language Runtime siehe CLR 27
Common Language Specification 1374
Common Management Information Protocol 448
Common Parameter 63
Common Type System 1374
compare 98
Compare-Object 98, 149, 701
Compare-VM 1098, 1115f.
CompatiblePSEditions 383
Complete-BITSTransfer 964
Complete-Transaction 376, 520ff.
Component Object Model 4
Component Object Model siehe COM
Compress-Archive 726
Computer 457, 1031, 1367
Computergruppe 357
ComputerInfo 855
Computername 156, 389, 863, 875
Computerrichtlinie 517
Computerverwaltung 855, 917
ConciseView 229
configuration 605
ConfigurationData 612
ConfigurationID 622
ConfigurationNamingContext 1040
confirm 65f.

Confirm 64 ff., 876, 931, 1048, 1051, 1281 ff., 1332
ConfirmPreference 66, 1283
conhost 323
Connect-AzAccount 1193
Connection 800
Connect-VMNetworkAdapter 1099
ConsolePaneBackgroundColor 336
Console.WriteLine() 1327, 1341
Container 1015, 1073, 1127, 1135
Container-Klasse 1007
continue 65, 226
Continue 214 ff., 234, 238
ConvertFrom-JSON 772
ConvertFrom-Markdown 769 f.
ConvertFrom-String 754
ConvertFrom-StringData 594
Convert-Html 769
Convert-String 753
ConvertTo-ContainerImage 1185
ConvertTo-CSV 753
ConvertTo-DataTemplate 1255
ConvertTo-JSON 772, 782
ConvertTo-SecureString 511, 1065
ConvertTo-WebApplication 1095
ConvertTo-XML 767
Convert-VHD 1099, 1110, 1113
Convert-Xml 768
copy 706
Copy-ContainerFile 1169 f.
Copy-GPO 1072
Copy-Item 234, 294, 689, 706, 712 f., 850, 993
Copy-NetFirewallRule 939
Copy/Paste 309
Copy-ToZip 728
Copy-VMFile 1119
CORBA 1377
Core 393
Count 117 f., 207, 226
Country 1044
CPU 150
CQL 466, 481
Create() 492, 1287
CreateCommand() 802
CreateElement() 762
CreateInstance() 874
CreateObject() 445
CreationTime 717, 1368
Credential 1065
Credentials 1046
CSV 76, 506, 750 ff., 853, 1060, 1091
CSV-Datei 150
CultureInfo 1309
CurrentThread 109
Cursor 797
CustomerID 1284
CVS 167

D

Dana Scully 1002
DataReader 796 f., 802 ff.
DataRow 130 f.
DataSet 796 f., 804, 807 ff.
Data Source Name siehe DSN 844
DataTable 807
Date 112
Datei 246, 450, 1364
– Eigenschaft 702, 717
– kopieren 706
– löschen 59
– Rechte 450
– verschieben 706
Dateiname 57
Dateinamenerweiterung 150 ff.
Dateisystem 3, 709, 977, 1367, 1379
Dateisystemfreigabe 603, 730
Dateisystemkatalog 716
Dateisystemoperation 696, 1265
Dateisystemstruktur 708
Dateiversionsverlauf 742
Datenabfrage 467
Datenbank 459, 666, 793
Datenbankmanagementsystem 800
Datenbankverbindung 800
Datenbankzeile 130
Datenbankzugriff 793
Datenbereich 593
Datenbindung 1255
Datendatei 593
Datenmenge 269
Datenquelle 844 f.
Datenquellensteuerelement 796
Datentyp 171, 180 f., 209, 1011, 1320, 1380
– .NET 101
– PowerShell 171
– WMI 454
Datenzugriff 801
DateTime 112, 116, 120, 427
Datum 203
Day 112
DB2 794, 844
dBase 844
DbCommand 802
DbDataReader 804
DBG 518
DbNull 805
DbProviderFactories 795
DCOM 445, 459, 475, 866, 1315, 1328
– Konfiguration 450
dcpromo 1065
DDL 466
debug 1327
Debug 65 f.
Debugger 44
Debugging 3, 43, 353, 518, 564
Debug-Modus 518
DebugPreference 66, 1327
Decimal 184
Deep Throat 1002
Default Domain Policy 1077
DefaultNamingContext 1040
Deinstallation 881
Delete() 1368
Deleting 875
Delimiter 750
Deployment 1379
Deployment Image Servicing and Management siehe DISM
Description 1018, 1044, 1292
DESCRIPTION 1285
Deserialisierung 286
Desired State Configuration siehe DSC
Desktop 393, 450
Desktop-Anwendungen 1375
Desktop Management Task Force siehe DMTF 277
Destruktor 1364, 1380
Deutsche Telekom 529
Developer PowerShell 348 ff.
Dezimalzahl 183
DHCP 927 ff.
Diagnose 515
Dialogfenster 509, 1242
diff 98
Digest 277
dir 402
Dir 392
Directory 977
DirectoryEntry 130 f., 173, 422 f., 1005 ff., 1020, 1023
DirectoryInfo 121, 136, 497, 702, 977, 1368 f.
Directory Management Objects 1068
DirectorySearcher 173, 1027
DirectorySecurity 979
DirectoryString 1018
Disable-ComputerRestore 869
Disable-ExperimentalFeature 391
Disable-JobTrigger 545
Disable-Mailbox 1084
Disable-NetFirewallRule 939
Disable-PnpDevice 873
Disable-PSRemoting 283
Disable-PSSessionConfiguration 293 ff.
Disable-VMIntegrationService 1102
Disable-WindowsOptionalFeature 905 ff.
Disk Quotas 450

DISM 905
Dismount-VHD 1110
DisplayName 1044
Distinguished Name 1010, 1015, 1018, 1040, 1044
Distributed COM 277
Distributed Component Object Model siehe DCOM 445
Distributed COM siehe DCOM 445
Distributed File System 450
Distributed Managements Objects siehe DMO
DML 466
DMO 828
DMTF 277, 448
DNS 304, 936, 1065, 1126 f.
DnsClient 929, 936
DNS-Client 932
DNSClient 931 f.
DNS-Konfigurationseinstellungen
– Per WMI abfragen 933
DNS-Server 450, 931, 935
do 214
Docker 51, 79, 372, 1123, 1126 ff., 1135, 1139, 1147 ff.
Docker CLI 1135, 1151
docker cp 1169 f.
Docker EE 1150
Docker Enterprise 1150
docker events 1182
docker exe 1164
Dockerfile 1154, 1181
Docker Hub 1152
Docker-Image 1155
docker info 1182
docker inspect 1167, 1173
docker logs 1182
docker pull 1158
docker run 1158
docker system 1182
DockPanel 1252
DOCX 790
Dokument 748
Dokumentation
– Active Directory 1020
– .NET 96
Dollarzeichen 146, 189
Domain 939
Domain Controller 1066
Domain Specific Language siehe DSL
Domäne 863, 1010, 1068, 1364, 1367
– Beitritt 304, 863
– hinzufügen 863
DOS 4
dotnet.exe 51, 233
DotNetTypes.Format.ps1xml 246, 254 f.
Dot Sourcing 75, 157 f., 528, 630, 1269 f., 1273
Double 184
DownloadString() 424, 947
DriveInfo 427
Driver 847
DriveType 437
Druckauftrag 874
– löschen 874
Drucker 246, 265, 449, 875
– verwalten 873 ff.
Druckerport 874
Druckerverwaltung 873 ff., 927
DSC 599
DSC Pull Server 617, 622
DSL 1260
DSN 844 ff.
DuplexingMode 875
Duration 165
DVD 431, 1116, 1121

E

echo 78 f.
Echo 78
Edit-NanoServerImage 1127
Eigenschaft 123 f.
Eigenschaftenzwischenspeicher 1005
Eigenschaftssatz 123, 126
Eingabe 506
Eingabeaufforderung 680 f.
Eingabedialog 508
Eingabemaske 1244
Eingabeobjekt 1317
Eingabesteuerelement 1255
Eingabeunterstützung 330, 353
Einzelobjekt 118 f.
Einzelschrittmodus 513
elevated 917
Elevated 160, 164, 917, 973, 1155
Elevation 160
Else 220
Emacs 153, 312 f., 354
E-Mail 947, 958
– Adresse 947
– EmailEvent 469
– senden 946
EmailAddress 1044
EmailEvent 469
Enable-BitLocker 747
Enable-ComputerRestore 869
Enable-ExperimentalFeature 391
Enable-JobTrigger 545
Enable-NetFirewallRule 939, 945
Enable-ODBCPerfCounter 844
Enable-PnpDevice 873
Enable-PSRemoting 281, 330, 875, 945
Enable-PSSessionConfiguration 293 ff.
Enable-PSSsessionConfiguration 295
Enable-VMIntegrationService 1102
Enable-WindowsOptionalFeature 905 ff.
Encoding 750
end 1267 f.
End 719
EndProcessing() 1299, 1303
endregion 339
Enter-PSSession 283, 293, 298, 406, 518, 1129
Enum 438
EnumerateCollection 1311
Enumeration 309
Enumerationsklasse 437
env 398, 861
Environment 602
Ereignis 1363, 1380
– PowerShell 582, 591
– WMI 464, 582
Ereignisabfrage 468
Ereigniskonsument 464 ff.
– permanent 464
– temporär 464
Ereignisprotokoll 150, 376, 449 f., 460, 466, 517, 603, 967
– Überwachung 469, 584
Ereignisprovider 464
Ereignissystem 582
Error 181, 231, 242, 875
ErrorAction 65 f., 231, 238 ff., 710
ErrorActionPreference 66, 181, 231, 240, 1053
ErrorBackgroundColor 315
ErrorRecord 234, 237, 241 f.
ErrorVariable 65, 241
ErrorView 230
Ethernet 929
ETS 100, 123, 130, 1342
Event 452
EventConsumer 452
Event siehe Ereignis
EventViewerConsumer 465
Example 1292
EXAMPLE 1285
Exception 234, 237, 243, 1323, 1329
Exchange Management Shell 668, 1083
Exchange Server 96, 450, 460, 1083
ExecuteNonQuery() 802
ExecuteReader() 802 ff.
ExecuteRow() 802

ExecuteScalar() 802
ExecutionPolicy 29 ff.
EXIF 717
Exists() 1013
exit 214
Exit-PSSession 285, 293
Expand-Archive 726
explorer.exe 743
Export-Alias 75 f.
Export-CliXml 506, 766
Export-Console 632, 1306
Export-Counter 972
Export-CSV 136, 389, 506, 753
Export-ModuleMember 635
Export-PfxCertificate 994
Export-VM 1098, 1115
Export-VMSnapshot 1114
Express 822
Expression 249
Expression Mode 78
Extended Reflection 100
Extensible Application Markup Language siehe XAML
Extrinsic Event 464

F

facsimileTelephoneNumber 1044
false 60, 66, 172, 181
Farbe 315 f., 392
Fax 1044
FBI 1002, 1023
Feature 894
FeatureOperationResult 900
Fehler 65
Fehlerausgabe 229
Fehlerbehandlung 231, 1376
Fehlerklasse 214, 237
Fehlermeldung 60
Fehlerstatus 231
Fehlersuche 512, 1324
Fehlertext 214
Fernaufruf 278
Fernausführung 156, 277
– Hintergrundauftrag 538
Fernverwaltung 277
Fernzugriff 277
Festplatte
– virtuell 1110
Festplattenverschlüsselung 745
Fibre-Channel 1098
Field 124
File 977, 1366
File History 742
FileInfo 121, 136, 497, 702, 977, 1368 f.
FileInformation 913
FileSecurity 977 ff.
FileSystem 664, 689
FileSystemAccessRule 980
FileSystemInfo 1368
FileSystemObject 1366
FileSystemRights 439
FileSystemWatcher 590
FileVersionInfo 893
filter 698
Find() 1006
Find-Module 642 f.
Find-Package 351, 891 f.
Firefox 603
Firewall 401, 671
Firewall-Regel 942
First 111
fish 371
For 215 ff.
Force 63, 700, 1048
Foreach 114, 145, 150 f., 214, 217, 562, 1006, 1313
Foreach-Object 98, 108 ff., 114 f., 126, 144 ff., 207, 218, 225, 561, 1091, 1311
Foregroundcolor 88
Forest 1068
Format 249
Formatkennzeichner 258
Format-List 99, 247, 979
Format-Table 112, 126, 150, 247 ff., 256, 979
Format-Wide 246 f., 251 f.
Format-Xml 759
Form Feed 191
Fortschrittsanzeige 267
Fox Mulder 1002
FoxPro 844
Framework Class Library 415
Freigabe 739
FTP 389
FullAccess 737
FullName 119, 1368
function 57, 214, 223, 228, 269
Funktion 57, 222 f., 269, 373
– eingebaut 228
– fortgeschritten 1276

G

GAC 431 f.
Ganzzahl 183
Gast 1097
Gateway 930
GeneralizedTime 1018
Geplante Aufgabe 540
Gesamtstruktur 1068 f.
Geschäftsanwendung 1339
GetAccessRules() 979 ff.
Get-Acl 973, 977 ff., 990
Get-ADComputer 1046
Get-ADDomain 1069
Get-ADDomainController 1069
Get-ADForest 1069
Get-ADGroup 1046, 1064
Get-ADGroupMember 1046, 1064
Get-ADObject 61, 1001, 1033, 1046 ff.
Get-ADOptionalFeature 1069
Get-ADOrganizationalUnit 1046, 1050
Get-ADPrincipalGroupMembership 1064
Get-ADRootDSE 1069
Get-ADUser 1046, 1051 f.
Get-Alias 70 f.
Get-AppLockerFileInformation 910
Get-AppLockerPolicy 910 f.
GetAssemblies() 433
Get-AuthenticodeSignature 533
Get-AzAks 1202, 1205
Get-AzAppServicePlan 1194
Get-AzLocation 1193
Get-AzResource 1194
Get-AzResourceGroup 1193
Get-AzSqlServer 1194
Get-AzSubscription 1193
Get-AzWebApp 1194 ff.
Get-BitLockerVolume 746 f.
Get-BITStransfer 965
Get-BITSTransfer 964
Get-BPAModel 999
Get-BPAResult 999
Get-CDRomDrive 666, 871
Get-ChildItem 58 ff., 99, 136, 140, 145, 150 f., 269, 392, 695, 698, 726, 849, 879
– BitLocker 747
Get-CimAssociatedInstance 474
Get-CimClass 447, 474, 481
Get-CimInstance 376, 447, 470, 474 ff., 492, 872, 875
Get-Clipboard 511, 645, 661
Get-Command 69, 82 f., 373, 648
Get-Computerinfo 863
Get-ComputerInfo 855, 1309
Get-Computername 1302
Get-ComputerRestorePoint 869
Get-Container 1184 f.
Get-ContainerImage 1160, 1184, 1187
Get-Content 689, 710, 748, 783, 853, 878, 1091, 1256
Get-Counter 278, 970 f.
Get-Credential 89, 301, 509, 673, 918, 947
Get-Culture 194
Get-DataRow 666, 817
Get-DataTable 666, 816

Get-Date 112, 116, 203 f.
Get-DHCPServer 34, 928
Get-DirectoryChildren 666
Get-DirectoryEntry 189, 666, 1322
Get-DirSize 1267
Get-Disk 690 ff., 871, 1311, 1315 ff., 1328
Get-DisplaySetting 873
Get-DnsClient 933
Get-DnsClientCache 936
Get-DnsClient-Funktion – Beispiel 933
Get-DnsClientServerAddress 933
Get-DomainController 34, 40, 1010
GetDrives() 426
Get-DSCConfiguration 616
Get-DVDDrive 669
Get-Error 231, 242, 317
Get-Event 586
Get-EventLog 26, 150, 278, 376, 967 f.
Get-ExCommand 1083
Get-ExecutionPolicy 29
Get-ExperimentalFeature 391
Get-ExportedType 703
GetFactoryClasses() 795
Get-Filecatalog 96
Get-FileHash 718 f.
Get-FileVersionInfo 703, 893
Get-FloppyDrive 669
Get-Flug 1339
Get-Flugziele 1339
Get-Font 863
Get-FreeDiskSpace 692 ff.
Get-GPInheritance 1077
Get-GPO 1071 ff.
Get-GPOReport 1075
Get-GPPermissions 1078
Get-GPPrefRegistryValue 1078
Get-GPRegistryValue 1078
Get-GPResultantSetOfPolicy 1075
Get-GPStarterGPO 1072
Get-Help 86 f., 90 ff., 95, 116, 168, 389, 393, 1267, 1285, 1294, 1334
Get-History 393, 678
Get-Host 393, 679
Get-HotFix 278
Get-Item 717, 849, 1090, 1093
Get-ItemProperty 717, 849
Get-Job 534 ff.
Get-JobTrigger 545
Get-Keyboard 872
Get-LDAPChildren 1034, 1272
Get-LDAPObject 1034, 1272
Get-LocalGroupMember 1079
Get-LocalUser 1079
Get-Location 70, 689
Get-LogicalDiskInventory 690
GetLongDateString() 116
GetLongTimeString() 116
Get-LoremIpsum 708
Get-Mailbox 1083 f.
Get-Mailboxdatabase 1083
Get-MarkdownOption 770
Get-Member 98, 117, 122 ff., 136 f., 427, 436, 442, 484, 1008, 1311
Get-Members 1023
Get-MemoryDevice 666, 871
Get-Methode 124
Get-Module 36, 250, 635, 646, 1343
Get-MountPoint 34
Get-MPComputerStatus 870
Get-MultiTouchMaximum 873
GetNames() 438
Get-NanoServerPackage 1127
Get-NetAdapter 929
Get-NetAdapterBinding 929
Get-NetFirewallAddressFilter 939
Get-NetFirewallAddressFilter-Funktion 941
Get-NetFirewallApplicationFilter 939
Get-NetFirewallInterfaceFilter 939
Get-NetFirewallInterfaceTypeFilter 939
Get-NetFirewallPortFilter 939
Get-NetFirewallProfile 939 f.
Get-NetFirewallRule 939 ff.
Get-NetFirewallRule-Funktion 941
Get-NetIPInterface 930
Get-NetworkAdapter 666, 871
GetObject() 445
Get-ODBCDriver 844
Get-ODBCDSN 844
Get-OSVersion 857
Get-Package 892
Get-PackageProvider 890 f.
Get-PackageSource 891
Get-Passagier 1339
Get-PipelineInfo 117, 121, 136, 1321
Get-PnpDevice 873
Get-PnpDeviceProperty 873
Get-PointingDevice 872
Get-PowerShellDataSource 1255
Get-Printer 875 f.
Get-PrintJob 875 f.
Get-Process 42, 58 ff., 69 f., 76, 99, 104, 114 f., 120 ff., 136, 146, 150, 231, 256, 265, 278, 389, 398, 915, 1297, 1320 ff.
Get-Processor 666, 871 f.
Get-PSBreakpoint 519
Get-PscxUptime 858
Get-PSDrive 690
Get-PSProvider 272
Get-PSReadLineKeyHandler 314
Get-PSRepository 643
Get-PSSession 293
Get-PSSessionConfiguration 293 f., 675
Get-PSSnapIn 633
Get-PswaAuthorizationRule 357
Get-Random 135, 185
GetRelated() 923
Get-ReparsePoint 725
Get-Service 88, 104, 116 f., 120, 136, 156, 265, 278, 389 f., 409, 921 f., 1286 f.
Get-SHA1 719
GetShortDateString() 116
Get-ShortPath 705
GetShortTimeString() 116
Get-SmbShare 731, 737
Get-SmbShareAccess 738
Get-SoundDevice 871
Get-SqlData 833
Get-Storagegroup 1083
Get-Tapedrive 871
Get-TargetResource 629
GetTempName() 444
Getter 124 f., 1380
Get-TerminalSession 34
GetTimestamp() 410
Get-TraceSource 515
Get-Transaction 376, 520, 523
GetType() 119, 181, 263, 383
Get-Unique 85, 138
Get-Uptime 410, 857 f.
Get-USB 872
Get-USBController 666, 872
Get-Variable 170, 174, 182
Get-VHD 1110
Get-VideoController 666, 871
Get-VirtualHardDisk 669
Get-VM 1098 ff., 1109, 1117
Get-VMBIOS-VM 1101
Get-VMBuildScript 1120 ff.
Get-VMHost 669, 1099
Get-VMMemory 1101
Get-VMProcessor 1101, 1118
Get-VMSnapshot 1114
Get-VMSummary 1122
Get-VMThumbnail 1121 f.
Get-WBBackupSet 744
Get-WBPolicy 744
Get-WBSummary 744
Get-WebApplication 1090
Get-WebitemState 1095 f.
Get-Website 1090, 1093
Get-WebvirtualDirectory 1090
Get-WindowsCapability 906
Get-WindowsEdition 857
Get-WindowsFeature 894 ff.

Get-WindowsOptionalFeature 905 f.
Get-WinEvent 278
Get-WmiObject 16, 278, 376, 431, 447, 474 ff., 694, 871, 877, 922 ff., 927, 970, 1003
Get-xDscOperation 624
Gigabyte 184
Git 79, 106, 167
git.exe 233
Github 36, 410
GitHub 47, 167
Gitternetz 1252
GivenName 1043 f.
Gleichheitszeichen 211
global 179, 1252
Global 1252
Global Assembly Cache siehe GAC
GlobalSign 529
Global Unique Identifier 1015, 1377
gm 98
Go 1222
Google 1250
GPMC 1071
Grafikkarte 450, 477
Grant-SmbShareAccess 738
Gravis 68, 107, 169, 191
grep 371
Grid 1252 f.
GridView 252
Group 139, 602, 1008
GROUP BY 466, 583
Group-Object 98, 139, 150, 968
Gruppe 1031, 1046, 1060 f.
– Active Directory 1017
– anlegen 1024
– auflisten 1023
– lokal 1079
– Mitglied aufnehmen 1024
Gruppenmitglieder 1046
Gruppenmitgliedschaft 1025
Gruppenrichtlinie 517, 683, 1071 ff., 1077
– Vererbung 1077
Gruppierung 139
GUID 694
Gültigkeitsbereich 171, 179
GZIP 728

H

Haltepunkt 43, 333, 518
Hardlink 723
Hardware 450, 666
Hardwareverwaltung 871
Hash-Tabelle 209, 423, 492
Hashtable 209 f., 423, 611 f., 629
Hashwert 716
HAVING 466, 583
Heimatordner 181
Help-Info 93
HelpMessage 1276
Herausgeber 530, 533
Here-String 187
Herunterfahren 864
Hexadezimalzahl 183
hidden 245
Hilfe 82
Hilfetext 92
Hintergrundauftrag 534
Hintergrunddatentransfer 963
Hintergrundübertragungsdienst 59
History 678
Hit Refresh 371
HKCU 269
HKEY_CURRENT_USER 851
HKEY_LOCAL_MACHINE 851
HKLM 269
Home 181
HomeDirectory 1044
HomeDrive 1044
Host 182, 336, 359, 679, 1097
Hosting 1355
hostname.exe 863
Hotfix 450
Hour 112
HTML 769 f., 1092, 1244
HtmlWebResponseObject 949
HTTP 277, 389, 617
HTTPS 277, 389
Hyper-V 33, 303, 651, 669, 1097, 1119, 1123, 1126 f., 1150, 1165
– Überblick 1097
Hyper-V-Container 1137
Hyper-V-Integrationsdienste 1102
Hypervisor 1097
Hyper-V-Modul
– Überblick 1098

I

i 212
IADs 1005 ff.
IADsComputer 1007
IADsContainer 1005 ff.
IADsGroup 1008
IADsUser 1007, 1021
idempotent 599 f.
Identität 1094
Identity 974
IdentityReference 982
IDL 461
IEnumerable 1006
if 214, 220
IIS 33, 271, 355, 1010, 1126 f., 1154
– Internet Information Services 1086
– Nano 1133
IISAdmin 923
IISAdministration 1086, 1133
IIS-Anwendung 1095
IIS Management Service 1133
ILSpy 436
Impersonifizierung 509, 1012
– WMI 463
Implizites Remoting 297
Import-Alias 76
Import-AzAksCredential 1202, 1205
Import-CliXml 506, 679, 766
Import-Counter 972
Import-CSV 150, 506, 701, 752, 1061, 1091
Import-GPO 1073
Import-INIFile 757
Import-Module 564, 636, 648, 1349 f.
Import-PSSession 297
Import-VM 1099, 1115 f.
IncludeUserName 916
Index 205
Indexer 1380
Informix 794, 844
Ingres 794
Inheritance Flags 974
INI-Datei 757
inlinescript 558 f.
Innertext 763
Input 181
Inputbox 1243
InputBox 433, 508, 1244
InputBox() 508, 1243
InputObject 104, 436, 1322 ff.
inquire 65, 239
Install-ADDSDomainController 1067
Install-ADDSForest 1067
Installation 880
Installationsordner 21, 128, 181
Installationstechnologie 880
Install-Module 36, 361, 640, 643, 1260
Install-Package 351, 435
Install-PswaWebApplication 356
installutil.exe 1300, 1305
Install-WindowsFeature 356, 618, 894
InstanceCreationEvent 468, 583 ff.
InstanceDeletionEvent 452, 464, 468, 583
InstanceModificationEvent 464, 468, 583
Instanz 423, 1364
Instanziierung 1365

Instanzmitglied 1381
int 172
Int32 172
Int64 184
INTEGER 1018
Integrated Scripting Environment 41
Integrated Scripting Environment siehe ISE 686
IntelliJ 354
IntelliSense 42, 58, 314, 346
Interface 1366
Interface Definition Language 461
InternalHost 679
InternalHostUserInterface 507
International .NET Association XXXIII
Internet Control Message Protocol 937
Internet Information Server 450
Internet Information Services 451, 617, 624, 907, 923, 1010, 1086
Interpretermodus 317
IntervallTimerInstruction 464
Intrinsic Event 464
InvalidOperationException 1006
Invoke-BPAModel 999
Invoke-CimMethod 376, 474, 492
Invoke-Command 283 ff., 300, 303, 518, 535, 1129
Invoke-ContainerImage 1160
Invoke-DbCommand 816
Invoke-DBMaint 831
Invoke-DBScalarCommand 816
Invoke-Expression 213
Invoke-History 678
InvokeMethod() 473
Invoke-Query 833
Invoke-SqlBackup 840 f.
Invoke-SqlCmd 820 f., 826 ff., 838, 841
Invoke-SqlCommand 666
Invoke-Webrequest 950, 963
Invoke-WebRequest 389, 949, 954, 960
Invoke-WmiMethod 376, 474, 491
IP
– Adresse 149, 173, 304, 467, 603, 927, 930, 1196
– Konfiguration 467
IPAddress 173, 930
ipconfig 79, 106
IPHostEntry 936
IP Routing 450
IRQ 450
Is64BitOperatingSystem 856
Is64BitProcess 856
ISA 466
IsCoreCLR 389
ISE 153, 267, 334, 394, 1129, 1355
– Debugging 333, 518
– Integrated Scripting Environment 329
IsePack 664
ISE Steroids 360
IsInRole() 974
IsLinux 389
IsmacOS 389
ISO 1097, 1104, 1107, 1116
Isolation 1165
IsWindows 389
Item() 1006
Iteration 108
IT-Visions XXXII

J

J 185
Java 1222, 1376 f.
JavaScript 436
JEA 672
Jeffrey Snover 154
Job
– zeitgesteuert 544
Job-Trigger 544
– Zeitgesteuerte Jobs 544
John Doggett 1002
Join 193
Join() 196
Join-String 145, 196
JPEG 717
JScript .NET 501
JSON 772, 960, 1235
Junction 724
Junction Point 723 f.
Just-In-Time-Compiler 1374

K

Kennwort 509 f., 1004, 1021
Kerberos 277, 463
Kill() 114 f.
Kilobyte 184
Klammer 61
– rund 199
Klammeraffe 78
Klasse 171, 426, 453, 457, 497, 1364 ff., 1380
– CIM 451
– COM 420, 442
– Commandlet 1299, 1319, 1341
– .NET 243, 415, 1299, 1377 ff.
– PowerShell 243 f.
– statisch 428
– WMI 449 ff.
Klassendiagramm 1368 f.
Klassenhierarchie 1368
Klassenmitglied 426, 1381
Klassenname 422
Known Host 407
Kommandomodus 317
Kommandozeilenbefehl 79
Kommentar 169
Komponentenorientierung 1374 f.
Komponententest siehe Unit Test
Komposition 1255
Komprimierung 715, 726 ff.
Konstante 786
Konstruktor 1364, 1380
Konstruktorfunktion 421 ff., 442
Kontakt 1031
Konvention 1340
Kopieren/Einfügen 309
Kosinus 427
Kreuzzuweisung 212
ksh 371
Kubernetes Command Line Client 1202

L

Label 249
Language Server Protocol siehe LSP
LastAccessTime 1368
LastExitCode 181, 231 ff.
Laufwerk 269 f., 276, 690, 850
– virtuell 1110
LCOW 1135 f.
LDAP 1001, 1008 ff., 1026
– Suchanfrage 1005, 1026
– Suche 1032
LDAP-Query 1027
Leaf 1015
Least Privilege 672
Leistung 970
Leistungsindikator 970 f.
Lenght 118
Length 117, 497
Limit-EventLog 278, 967 ff.
LinearGradientBrush 1253
Linie 1368
LINK 1285
Linux 6 f., 13, 25, 45, 53, 56, 272, 371, 379, 398, 406, 441, 639, 1136, 1145, 1377
– Container 1180 f.
– Dateisystem 402
Linux-Container 1135
Linux Foundation 371
LinuxKit 1136
Literal 258
Little Endian 1026

Lizensierung 671
Load-ContainerImage 1186
LoadFrom() 432
LoadWithPartialName() 432
Logarithmus 427
Log File Event Consumer 466
Logoff 683
Logon 683
Lokalisierung 456, 596
Loopback 280
ls 398
LSP 353

M

Machine.config 795
MachineName 279, 1303
macOS 7, 13, 25, 45, 55, 272, 371, 379, 398, 441, 639, 1377
– Dateisystem 402
MailAddress 947
MailMessage 946
makecert.exe 529
MAML 92, 1334
man 398
Manage-Bde 745
Managed Code 1004
Managed Object 448
Managed Object Format 461
Managed Provider 793, 1027
ManagedThreadId 109
ManagementBaseObject 472
ManagementClass 130 f., 173, 472 ff., 478, 483, 1311
ManagementEventWatcher 585
Management Infrastructure API 472 f.
ManagementObject 131, 173, 472 ff., 478, 482 ff., 491, 923, 1311
ManagementObjectCollection 482, 1311
ManagementObjectSearcher 173, 478, 1311
ManagementScope 584
Mandatory 1276, 1315
Manifest 637
Manifestmodul 1343
Markdown 769
MarkdownInfo 770
Maschinencode 1374
Match 132
MAXIMAGO XXXII
MaximumDriveCount 276
MaximumErrorCount 181
maxSessionsAllowedPerUser 357
MCR 1152
md 710, 718, 853
measure 146
Measure-Command 514 f., 695
Measure-Object 98, 146, 150
Measure-VM 1099
Megabyte 184
Mehrsprachigkeit 596
Menge 118 f.
Mercurial 167
Merge-VHD 1110
Message 234
MessageBox 509, 1242
MessageBoxButtons 1242 f.
MessageBoxDefaultButton 1243
MessageBoxIcon 1243
Metamodell 458
Metaobjekt 1015
Methode 123 f., 1363, 1368, 1380
– Getter 1380
– Setter 1380
Method siehe Methode
Microsoft Access 809
– Treiber 799
Microsoft-Access 21
Microsoft.ACE.OLEDB 799
Microsoft Azure 1188
Microsoft Certified Solution Developer XXXII
Microsoft Command Shell 4
Microsoft Container Registry siehe MCR 1152
Microsoft Developer Network siehe MSDN 211
Microsoft Excel 1061
Microsoft Exchange 1066
Microsoft Exchange Server 603, 668, 1083
Microsoft.GroupPolicy 1072
Microsoft.Jet.OLEDB 799
Microsoft.Management.Infrastructure.CimClass 484
Microsoft.Management.Infrastructure.CimClassProperties 484
Microsoft.Management.Infrastructure.CimInstance 484
Microsoft.Management.Infrastructure.CimInstanceProperties 484
Microsoft.Management.Infrastructure.CimProperty 484
Microsoft Office 451, 786
Microsoft Outlook 920
Microsoft.PowerShell 50
Microsoft Print Ticket XML 875
Microsoft Shell 4
Microsoft SQL Server 821, 1196
Microsoft System Center 470
Microsoft.Update 865
Microsoft.Vhd.PowerShell 1110
Microsoft.VisualBasic 508, 1243 f.
Microsoft.VisualBasic.Interaction 445, 508, 1244
Microsoft.Win32 879
Microsoft.Win32.RegistryKey 849
Microsoft Word 445
Minute 112
Mirantis 1150
Mitglied 1363
– .NET 1380
– statisch 1381
– WMI 488
MMC 463
Mock-Objekt 1264
Modul 635, 644
Module Browser 641
Modulo 210
MOF 461, 610 ff.
Monad 4
Monica Reyes 1002
Moniker 1009
Mono 1377
Month 112
more 257
Most Valuable Professional XXXII
Mount-SpecialFolder 691
Mount-VHD 1110
move 706
Move-ADObject 1046 f.
Move-Item 689, 706, 718
Move-Mailbox 1084
Move-VM 1099
MSCL 15
mscorlib.dll 1379
MSDN 211, 1249
MSDN Library 419
MSFT_Printer 875
MSFT_PrintJob 875
MSFT_SmbShare 737
MSFT_SmbShareAccessControlEntry 738
MSFT_WUOperationsSession 865, 868, 1131
MSFT_WUSettings 865
MSFT_WUUpdate 865
MSH siehe Microsoft Shell
MSI 626, 877 ff., 888, 1300
MTA 1247
Multithreading 109, 1376
MySQL 794, 806, 820, 831, 1152
MySqlConnection 806
MySqlLib 820

N

Nachkommastelle 183, 427
Name 104, 1044
Namensauflösung 936
Namensraum 453 ff., 977, 1068, 1377
– ADSI 1010

– .NET 1377ff.
– WMI 455, 458
Namensraumhierarchie 1378
NamespaceCreationEvent 468, 583
NamespaceDeletionEvent 468, 583
Namespace-ID 1009
NamespaceModificationEvent 468, 583
Nano Server 12, 369, 1123, 1126, 1162
– IIS 1133
– Installation 1127
– Paketinstallation 1127, 1131
NanoWbem 447
NativeObject 1006ff.
Navigation 269
Navigation Provider 270
Navigationsbefehl 272
Navigationsmodell 689
Navigationsparadigma 269
Navigationsprovider 1001
Negation 213, 440
NetAdapter 929f.
NetSecurity 939
NetSecurity-Modul
– Überblick 938
Netsh 932, 935, 942f.
Netsh
– Per PowerShell aufrufen 934
netstat 79
NetTCPIP 929f.
NetworkInterface 1309
Network Load Balancing 450
Netzlaufwerk 449
Netzlaufwerksverbindung 450
Netzwerkadapter 1097
Netzwerkcenter 282
Netzwerkkarte 450, 467, 927, 1364
– Geschwindigkeit 932
Netzwerkkartenprofil 945
Netzwerkkonfiguration 671, 927
Netzwerkmanagement 447
Netzwerkprofil
– Per PowerShell setzen 945
Netzwerkverbindung 450, 603, 929f.
Neustart 304, 900
Neustarten 864
new() 421ff., 442
New-ADGroup 1046, 1064
New-ADObject 1047
New-ADOrganizationalUnit 1046, 1050
New-ADUser 1046, 1051, 1054
New-AppLockerPolicy 910, 913
New-AzAks 1202ff.
New-AzSqlDatabase 1196
New-AzSqlServer 1196
New-AzSqlServerFirewallRule 1196
New-AzWebApp 1195
New-Buchung 1339
New-Button 87, 1251ff.
New-CheckBox 1255
New-CimInstance 474, 492
New-CimSession 477
New-CimSessionOption 477
New-ComboBox 1255
New-Container 1183
New-DSCChecksum 623
New-Elippse 1255
New-Event 591
New-EventLog 278, 967ff.
New-FileCatalog 716
New-Fixture 1262
New-GPLink 1073
New-GPO 1072
New-GPStarterGPO 1072
New-Grid 1253
New-Guid 421
New-Hardlink 724
New-HardwareProfile 669
New-IISSite 1086, 1133
New-Image 1255
New-Int64Animation 1255
New-Item 269, 689, 721, 749, 849f., 853
New-Itemproperty 851ff.
New-JobTrigger 545ff.
New-Junction 725f.
New-Label 1248, 1253
New Line 191
New-Line 1255
New-ListBox 1255
New-LocalUser 28
New-Mailbox 1084
New-Mailboxdatabase 1084
New-MediaElement 1255
New-Menu 1255
New-Module 635
New-NanoServerImage 1127f.
New-NetFirewallRule 939, 942
New-NetIPAddress 930f.
New-Object 420ff., 429ff., 442, 786, 962, 1281, 1381
New-PasswordBox 1253
New-ProgressBar 1255
New-PSDrive 276, 850, 879, 1041
New-PSSession 281ff., 293, 296f., 300, 303, 676
New-PSSessionConfigurationFile 672
New-RadioButton 1255
New-Rectangle 1255
New-RichTextBox 1255
New-ScheduledJobOption 548
New-ScrollBar 1255
New-SelfSignedCertificate 530
New-Service 921, 925
New-Shortcut 722
New-Slider 1255
New-SmbShare 65, 736f.
New-StatusBar 1255
New-Storagegroup 1084
New-Storyboard 1255
New-TextBlock 1255
New-TextBox 1253
New-TimeSpan 204
New-TreeView 1255
New-UrlShortcut 723
New-VHD 1106, 1110f.
New-ViewBox 1255
New-VirtualDVDDrive 669
New-VirtualNetworkAdapter 669
New-VM 669, 1099, 1104, 1122
New-VMSwitch 1099
New-WebApplication 1095
New-WebAppPool 1094
New-WebServiceProxy 957ff.
New-Website 1086, 1090, 1133
New-WebVirtualDirectory 1095
New-Window 1253
New-Zip 728
NoAccess 737
Node 762
node.js 1152
nodeJS 1222
NoElement 140
Non-Terminating Error 234, 1328, 1331
NoProfile 528
NormalView 229
Northwind 822, 842
Notation 1368
– umgekehrt polnische 1026
Notepad 153
NoteProperty 497, 757
Notes 1292
NOTES 1285
Notizeigenschaft 123, 128, 137
Novell 1010
Now 427
Nslookup 932, 936
NtAccount 982
NTAccount 978, 981f.
NT Event Log Event Consumer 466
NTFS 460, 715
NTLM 277
NTSecurityDescriptor 1017
Nuget 418, 434
NuGet 36, 51, 351, 1222
NuGet Package Manager 351
null 147, 156, 176, 219, 223, 265
Null Coalescing Assignment Operator 176

Null Coalescing Operator 176, 390
Null Conditional Operator 175 f.
Null-Wert 113

O

Object 1369
Object[] 118, 226
ObjectCategory 1017, 1030, 1049
ObjectClass 1017, 1030, 1043 f., 1049
ObjectGUID 1018, 1044
ObjectiveC 1222
ObjectSecurity 977
ObjectSecurityDescriptor 1017
ObjectSid 1017
ObjectVersion 1019
Objekt 204, 754, 1341, 1363, 1366 f.
– Dynamisch 497, 1267
– .NET 1246
– WMI 451
Objektadapter 131, 482 f., 796
Objektassoziation
– WMI 458
Objektbaum 1367
Objektidentifikation
– ADSI 1009 f.
Objektmenge 700
Objektorientierte Programmierung siehe OOP 1363
Objektorientierung 100
Objektorientierung siehe OO 1374
Objekt-Pipeline 701
Objekttyp 1364
OCI 386
OData 617
ODBC 793 f., 844, 847
Office 1044
OFS 181
ogv 246
Oh-my-Posh 327
OK 1242
OKCancel 1242
OLEDB 793 f., 1007
– Provider 809, 1007, 1027
OleDbCommand 802, 809
OleDbConnection 800, 804, 809
OleDbDataAdapter 809
OMI 447, 476
OMI siehe Open Management Infrastructure
On_Click 1253
OneGet 890
OneLevel 1048
ONELEVEL 1026
OO 1363, 1374 f.
OOP 1363
OOP siehe COP 98
Open() 800
Open Database Connectivity
– Einstellung 450
Open Management Infrastructure 404
– OMI 476
Open Source 5
OpenSSH 404
OpenView 470
Operator 133, 196, 206, 210
Optimize-VHD 1110
Oracle 386, 794, 806
Oracle Cloud Infrastructure siehe OCI 386
OracleCommand 802
OracleConnection 800
Ordner 59, 81, 273, 695, 705 f., 709
– Dateisystem 450
Ordnerstatistik 699
Organisationseinheit 1031, 1060 f.
– anlegen 1025
OSS
– Open Source 5
OutBuffer 65, 102
Out-Default 246, 254, 1358 ff.
Out-File 246, 266
Out-GridView 89, 246 f., 252 f.
Out-Host 246, 257
Outlook 139, 787
Outlook.Application 786
Out-Null 246 f., 264, 432
Out-Printer 246, 265, 874
OUTPUTS 1285
Out-Speech 246, 267
Out-SqlScript 831
OutVariable 65, 148
ov siehe OutVariable

P

Packaged Script 348
PackageManagement 638
Page File 671
Paketinstallation
– Nano Server 1131
Paketmanager 36
Panel 1252
PaperSize 875
Papierkorb 716
parallel 561
Parallelisierung 109
Parallelität 110
Parameter 58 f., 105, 227, 1276, 1292, 1315, 1324
– Abkürzung 62 f.
– Skript 156
PARAMETER 1285
Parameterliste 156
ParentSession 294
ParsedHtml 950
parsen 753
Partition 1046
PascalCasing 1378
PASH 4
PassThru 148, 1046
passwd 400
Pause 875
PE 703
PercentComplete 267
Perforce 167
Performance Counter Provider 970
Performance Monitor 449, 460
PERL 168
Persistenz 562
Pester 1259 f., 1353
Petabyte 184
Pfad 457
– ADSI 1009
– Verzeichnisdienst 1009
– WMI 453, 456 f.
Pfadangabe 273
Pfeilspitze 1368
Pflichtparameter 26
PHP 168, 1222
PhysicalDeliveryOfficeName 1019, 1044
PIN 746
Ping 79, 450, 937 f., 1328
Ping-Host 34, 40 f., 937
Ping-VM 1122
Pipe 100
Pipeline 3, 15, 99 f., 120, 146, 264, 372, 487, 1222, 1235 f.
– Ausgabe 1309
– Eingabe 1317
Pipeline Processor 101, 1300
PipelineVariable 65, 149, 256
Pipelining 98, 210, 269
Plattform Invoke 503
Plattformunabhängigkeit 1374
Platzhalter 61, 258
Plug-and-Play 873
Polymorphismus 1370
Port 401
Portable-Executable-Format siehe PE 703
PoshConsole 364
posh-git 327
Position 1276, 1315
Postfach 1084
Postfix-Notation 1026
PowerGUI 153, 329
Power Management 460
PowerShell 3, 70, 98
– Extension 666
– Hosting 3

– Konsole 307
– Laufwerk 269, 276, 850
– Skriptsprache 168
– Version 1.0 4
– Version 2.0 4
– Version 3.0 4
– Version 4.0 4
– Version 5.0 4
– Version 5.1 5
– Web Admin 677
PowerShell Analyzer 345
PowerShell Community Extensions 34, 660, 1001
PowerShell Community Extensions siehe PSCX 660
PowerShell Core 7, 17, 45
– Funktionsumfang 373
– installieren 45
– Konsole 392
– Version 5.1 12, 369, 1126
– Version 6.x 6, 56, 371
– WMI 447
PowerShell Direct 302 ff., 1129
PowerShell Editor Services 353 f.
powershell.exe 69, 649, 681, 1247
powerShellExePath 396 f.
PowerShell Gallery 33, 36 ff., 602, 638
PowerShellGet 38 f., 638
PowerShell Management Library for Hyper-V 1098, 1120
PowerShellPlus 153, 329, 361, 1322
PowerShell Remoting
– Port 302
PowerShell-Remoting 355
PowerShell Remoting Protocol 277 ff., 404
PowerShell Remoting siehe Remoting 277
PowerShell Script Analyzer 340, 353
PowerShell Web Access siehe PSWA
PowerStudio 347
Predictive IntelliSense 314
PrimalScript 365
Principal 978
Printer 246
Printing 875
PrintManagement 875
Print Ticket XML 875
Private 939
Privileg 463
process 1267
Process 104, 120, 136, 149, 602, 719, 1322
ProcessRecord() 1299, 1303, 1315
Professional Developer Conference 4
profile 524
ProfilePath 1044
profile.ps1 432, 1269, 1324
Profilskript 320, 524, 528
Programmcodeanalyse 340
Programmgruppe 450
Programmiersprache 219
Programmiersprachenunabhängigkeit 1374
Prompt 680
PromptForChoice() 507
Propagation Flags 975
Property 124, 256, 1363, 1380
Property Cache 1014
PropertyCollection 1005
PropertyDataCollection 473, 483 f.
PropertyGrid 1246
PropertyNames 1006
PropertyValueCollection 1005, 1013
ProtectedFromAccidentalDeletion 1044, 1048
Protokolldatei 466
Protokollierung 517
Provider 272
– ADO.NET 793
– Dateisystem 689
– PowerShell 270
– Verzeichnisdienst 1001
– WMI 451
Proxy 958
ProxyCommand 1287
Proxy-Commandlet 297, 1286
Prozedur 223
Prozess 58 f., 150, 450, 459
– auflisten 265, 915
– beenden 919
ps 398
PSAnsiRenderingFileInfo 392
PSAzureProfile 1193
PSBase 1008, 1014 ff.
PSCmdlet 1299
PSCodeGen 665
PSComputerName 291
PSConfig 651
PSCredential 509, 918, 1065
PSCustomobject 1281
PSCustomObject 118, 137, 499, 751, 757, 1278, 1281
PSCX 39 f., 246, 511, 640, 651, 660, 721, 728 f.
PSCX siehe PowerShell Community Extensions
psd1 595
PSDriveInfo 690
PSDSCRunAsCredential 605
psedit 331, 338
PSEdition 369, 393 f.
PSHome 181
PSHost 181, 1357
PSHostRawUserInterface 1357 ff.
PSHostUserInterface 1357 ff.
PSImageTools 665
psISE 336
PSItem 99, 500
PSModuleAutoLoadingPreference 181, 647
PSModulePath 388, 398, 636, 1344
PSObject 131, 1360 f.
PSReadline 311 ff., 326 f., 392 f.
PSRemotingJob 535
PSRP siehe PowerShell Remoting Protocol
PSRSS 665
PSScheduledJob 540
PSScriptRoot 158
PSSession 292
PSSnapIn 1301
PSStyle 316
PSSystemTools 665, 690, 872 f.
psUnsupportedConsoleApplications 335
PSUserTools 665
PSVariable 170
PSVersion 394
psversiontable 52, 369
PSWA 355 f.
Public 939, 1315
Public Network 282
Pull-ContainerImage 1155
Pull Request 410
Punktnotation 112, 424, 488
Push-ContainerImage 1187
Put() 488 f.
pv siehe PipelineVariable 148
pwsh 47, 392
pwsh.exe 319
Python 168, 1222

Q

Quantifizierer 199
Quantor 199
QueryDialect 481
Quest 651, 666

R

Range 193
Raspberry Pi OS 13
Raspbian 13
RawUI 336
RDP 304, 945
ReadAccess 737
Read-Host 506, 511, 555 ff.
Receive-Job 534 ff., 554

Rechenleistung 150
Recovery Console 1123 f.
recurse 60, 698, 1046
recursive 1046, 1051
Red Hat Enterprise Linux 13
Redirection siehe Umleitung
Redo 314
Redstone 5
REFERENCES OF 466
Referenzkopie 211 f.
Refresh() 1368
RefreshCache() 1014
RefreshFrequencyMins 622
Regel 909, 942
region 339
Register-ArgumentCompleter 310
Register-CimIndicationEvent 474, 590
Register-DnsClient 933
Register-Event 588
Register-ObjectEvent 741
Register-Packagesource 642, 891
Register-PSSessionConfiguration 293, 296, 672
Register-ScheduledJob 546 f.
Register-WmiEvent 585, 590
Registrierungsdatenbank 3, 269, 275 f., 450, 614, 849, 879
– Schlüssel 849
Registry 458 ff., 602, 849
RegistryKey 879, 977
RegistrySecurity 979
RegistryValueChangeEvent 464, 468, 583
Regulärer Ausdruck 173, 197, 715
Relative Distinguished Name 1015 f.
Remote Desktop Protocol siehe RDP
Remote Procedure Call siehe RPC
Remote Server Administration Tools 1038
Remoting 156, 277, 304, 406, 875, 1129
– Implizit 297, 1286
Remove-ADGroup 1064
Remove-ADGroupMember 1046, 1064
Remove-ADObject 1046 ff.
Remove-ADOrganizationalUnit 1050
Remove-ADUser 66, 1051
Remove-Alias 76
Remove-AzAks 1202, 1221
Remove-AzSqlDatabase 1197
Remove-AzSqlServer 1197
Remove-AzWebApp 1196
Remove-Buchung 1339
Remove-CimInstance 474, 493
Remove-Computer 863
Remove-Container 1184
Remove-ContainerImage 1184 ff.
Remove_DirectoryEntry 1332
Remove-DirectoryEntry 666, 1322
Remove-Event 588
Remove-EventLog 278, 967
Remove-GPLink 1073
Remove-GPO 1072
Remove-GPPrefRegistryValue 1078
Remove-GPRegistryValue 1078
Remove-Item 59, 63, 66, 76, 689, 706, 850, 853
Remove-ItemProperty 852
Remove-Job 534, 537
Remove-JobTrigger 545
Remove-LDAPObject 1034, 1272
Remove-LocalUser 1079
Remove-Module 636, 650
Remove-NetFirewallRule 939, 943
Remove-NetFirewallRule-Funktion 943
Remove-NetIPAddress 930
Remove-NetRoute 930
Remove-ODBCDsn 844
Remove-PrintJob 875 f.
Remove-PSBreakpoint 519
Remove-PSSession 293 f.
Remove-PSSnapin 376
Remove-PswaAuthorizationRule 357
Remove-Service 926
Remove-SmbShare 63, 66, 737
Remove-Variable 180
Remove-VM 1099, 1109
Remove-VMSnapshot 1114
Remove-WebApplication 1096
Remove-WebAppPool 1096
Remove-Website 1096
Remove-WebVirtualDirectory 1096
Remove-WindowsCapability 905
Remove-WindowsFeature 894, 900
Remove-WmiObject 474, 493
Rename-ADObject 1046 f.
Rename-Computer 863
Rename-Drive 694
Rename-GPO 1072
Rename-Item 706
Rename-NetAdapter 932
Rename-NetFirewallRule 939
Rename-VM 1099
Rename-VMSnapshot 1114
Repair-VM 1099
Replace 195
Replikation 449
Repo 1222
Repository 459
requires 164
Resize-VHD 1099, 1110
Resolve-Assembly 433
Resolve-DnsName 936
Resolve-DNSName-Funktion
– Beispiel 936
Resolve-DsnName 933
Resolve-Host 936
Resolve-Path 273
ResponseHeaders 424
Ressource 601
Ressource Group 1188
REST 960
Restart-Computer 278, 864, 900
Restart-PrintJob 875
Restart-Service 66, 299, 302, 921, 924
Restart-VM 1099
Restore-ADObject 1047
Restore-Computer 869
Restore-DscConfiguration 614
Restore-GPO 1073
Restore-VMSnapshot 1114
Restricted 158
Restricted Runspace 672
Resume-PrintJob 875
Resume-Service 921, 924
Resume-VM 1099
return 214, 275, 1300
Revoke-SmbShareAccess 738
RHEL 13
Richtlinienergebnisbericht 1075
Robocopy 712 ff.
Rolle 894, 1126
Rollendienst 894
rootcimv2 457
RPC 278
RSAT 33, 1098
RSS 665, 948
Ruby on Rails 1152
Rückgabeobjekt 1309
RuleCollection 911
Run-ContainerImage 1160, 1179, 1183
RunNow 546
Runspace 345, 672
RuntimeException 243

S

s 376
sa 830
SAM 1010
SAMAccountName 1018, 1021, 1026, 1044
Sapien 365, 671
SAPI.SPVoice 267, 442 f.
Satya Nadella 371
Save-ContainerImage 1185
Save-Help 93

Save-Module 640
Save-VM 1099
SCA 46 f., 372
Schablone 1364 f.
Schalter 60, 1342
Schattenkopie 742
ScheduledJob 546
Scheduled Task 540
Schema 1015, 1042
– Active Directory 1020
– WMI 458
Schemaabfrage 467
SchemaNameCollection 1006
SchemaNamingContext 1040
Schleife 108, 217
Schlüssel 269
Schlüsselattribut
– WMI 453
Schnittstelle 243, 797, 1366, 1370
– .NET 1381
Schriftart 863
Schtasks.exe 540
Schwichtenberg, Holger XXXII
Scope siehe Gültigkeitsbereich 171
script 179
Script 616
Script Analyzer 340
Scripting.FileSystemObject 444
ScriptMethod 497
Script-Migration 351
ScriptPaneBackgroundColor 337
SDDL 675, 731, 991
sealed 1369
Searcher 865
SearchScope 1048
Secure Socket Layer 356 f., 950, 959
Secure String 745 f.
Security Descriptor 974
Security Descriptor Definition Language 295
Security Identifier 974, 978 ff.
Security Service Provider 463
Select
– PowerShell 136
SELECT 466, 582
– WQL 466 ff.
SelectNodes() 760 f.
Select-Object 26, 98 f., 111, 126, 131 f., 136 ff., 150, 256, 486, 761, 1324
SelectSingleNode() 760 f.
Select-String 80, 105, 714, 749
Select-Xml 760 f.
Self-Contained App siehe SCA 46
Semantic Versioning 173, 601
Semaphore 977
Semikolon 150, 917
Send-MailMessage 946 f.
Send-SmtpMail 946
sequence 558
Serialisierung 120, 286
Seriennummer 859
Server 1046
ServerCertificateValidationCallback 959
Server Management Objects siehe SMO
Server Manager 1065
ServerRemoteHost 359
ServerURL 622
Service 602
ServiceController 120, 286, 921
Serviceorientierung 1375
Serviceorientierung siehe SOA 1374
ServicePointManager 950
Session 445, 865
sessionState 357
Set-Acl 973, 987, 990
Set-ADAccountPassword 1046
Set-ADGroup 1064
Set-ADObject 1047 f.
Set-ADOrganizationalUnit 1050
Set-ADUser 1051 ff.
Set-Alias 75
Set-AppLockerPolicy 910
Set-AuthenticodeSignature 531
Set-AzAks 1202
Set-AZContext 1193
Set-AzWebApp 1195
Set-BPAResult 999
Set-CimInstance 474, 490
Set-Clipboard 511
Set-Content 689, 749, 783, 947
Set-DataRow 817
Set-DataTable 666, 817
Set-Date 204
Set-DistributionGroup 1084
Set-DnsClientServerAddress 931 ff.
Set-ExecutionPolicy 29 ff., 152, 158, 529, 533
Set-FileTime 702 f.
Set-GPInheritance 1077
Set-GPLink 1073
Set-GPPermissions 1078
Set-GPPrefRegistryValue 1078
Set-GPRegistryValue 1078
SetInfo() 1005, 1014
Set-Item 299, 689
Set-ItemProperty 702, 852, 945
Set-JobTrigger 545
Set-Location 70, 269, 689, 849
Set-Mailbox 1084
Set-MarkdownOption 770
Set-Methode 124
Set-NetFirewallPortFilter 939
Set-NetFirewallProfile 939 f.
Set-NetFirewallRule 939, 943
Set-NetIPInterface 930
Set-ODBCDriver 844
Set-ODBCDsn 844
Set-PrintConfiguration 875
Set-PSBreakpoint 518 f.
Set-PSDebug 174, 512 f.
Set-PSReadLineKeyHandler 314
Set-PSReadlineOption 313, 393
Set-PSReadLineOption 314 f.
Set-PSSessionConfiguration 293
Set-ScheduledJob 546
Set-Service 390, 921, 924 f.
Set-StrictMode 174
Set-TargetResource 629
Setter 124 f., 1380
Set-TraceSource 516
Set-Variable 170, 182, 1251 f.
Set-VHD 1110
Set-VM 1099 ff.
Set-VMMemory 1122
Set-VMProcessor 1138, 1142
Set-VolumeLabel 694
Set-VolumneLabel 34
Set-WmiInstance 474, 489
Set-WSManQuickConfig 282
SHA256 716
Shell 3, 99
Shell.Application 716, 729
Shielded VM 1127
ShouldProcess() 1332 f.
Show() 1242
Show-Command 89, 332
ShowDialog() 1256
Show-EventLog 278, 968
Show-HyperVMenu 1121
Show-Markdown 770
Show-NetFirewallRule 939
Show-Service 278
Show-VMMenu 1121
ShowWindow 90
Shutdown 683
Sicherheit
– COM 463
– Dateisystem 450, 460
– PowerShell 158
– WMI 463
Sicherheitsabfrage 1281, 1332
Sicherheitsbeschreibung 974
Sicherheitseinstellung 973
Sicherheitsmodell 3
Sicherheitsrichtlinie 159
SID 974
Side-by-Side Executing 1376
Signatur
– digital 529

Signieren 529
SilentlyContinue 65, 239, 710, 1053
Simple Network Management 449, 460
Simple Object Access Protocol
siehe SOAP 277, 957
Sitzung 292 ff.
Sitzungskonfiguration 295, 672, 676
Skip 111
SkipEditionCheck 383
SkipNetworkProfileCheck 283, 945
Skript 152 ff.
– PowerShell 152
Skriptausführungsrechte 29
Skriptausführungsrichtlinie 30
Skriptblock 179, 285, 1251
Skriptdatei 152
Skripteigenschaft 123, 128
Skriptfenster 41
Skriptmodul 1343
Skriptsprache 1267
SMB 1126
SMO 822, 828 f., 838 ff.
Smoking Man 1002
SMTP 946 f.
SmtpClient 946
Snap-in 1300 ff., 1324, 1335
Snap-In 376, 630, 635, 649
Snapshot
– Hyper-V 1113
SNA Server 460
Snippet 331
SOA 1374
SOAP 277, 459, 957
Software 449 f., 670
– deinstallieren 881
– installieren 626 ff., 880
– inventarisieren 877
– verwalten 877
Softwareentwickler 419
Softwareentwicklungsplattform 1375
Softwarekomponente 417, 431, 434
Softwarepaket 891
Softwarequelle 891
Software Restriction Policy 909
Sortieren 137
Sort-Object 98 ff., 131, 137, 140, 145, 150 f., 226, 1300
Speak() 443
Speech 246
SpeechSynthesizer 267
Speicher 118
Speicherbereinigung 1376
Speicherverbrauch 798
Speicherverwaltung 1376
Spitzname 1366
Spoolerdienst 875
Spooling 875
Sprachausgabe 246, 267, 443
Sprache 596
Sprachkürzel 596
SQL 466, 848
SQLASCOMMANDLETS 821
Sqlcmd.exe 827
SqlCommand 802, 826
SqlConnection 423, 800, 804 ff.
SqlDataSourceEnumerator 796
SQLPS 271, 820 ff., 826
SQLPSX 820 ff., 831 f., 838
SQL Server 371, 794, 820 f.
– Agent 838
– Laufwerk 823
SqlServerCe 794
SqlServerCmdletSnapin100 821
SQL Server Management Studio 822, 838
SSH 405 f.
sshs 405
SSL siehe Secure Socket Layer
STA 1247
StackPanel 1252
StackTrace 181
Stammzertifizierungsstelle 530, 533
Standarddrucker 265
Standardkonsole 307
Start-BITSTransfer 964
Start-Container 1183
Start-ContainerProcess 1164
Start-DscConfiguration 607
Start-Job 534 ff.
Startmenü 450
Start-Process 540, 785, 915 ff.
Start-PSSession 538
Start-Service 287, 921, 924
Start-Sleep 165
Start-Transaction 520 ff.
Start-Transcript 555
Startup 683
Start-VM 1099, 1107, 1122
Start-WBBackup 744
Start-WBFileRecovery 744
Start-WBHyperVRecovery 744
Start-WBSystemStateRecovery 744
Start-WBVolumeRecovery 744
Start-Webitem 1096
Start-Website 1095
static 243
Status 267, 875
Stop 65, 238
Stop-Computer 278, 864
Stop-Container 1184
Stop-Job 534, 537
Stop-Process 114, 555, 915, 919, 1322
StopProcessing() 1299
Stop-Service 59, 921, 924
Stop-VM 1099
Stopwatch 410, 515
Stop-WBJob 744
Stop-Webitem 1096
Stop-Website 1095
Stored Procedure 808
Streaming 102
StreetAddress 1044
String 187, 195
Subnetzmaske 930
Substring() 193
SubTree 1048
SUBTREE 1026
Subversion 167
Suche
– Active Directory 1026
– Assembly 703
– LDAP 1007
– Verzeichniseintrag 1016
– XML 760
SupportsShouldProcess 1332
Surname 1044
Suse 13
Suspend 64
Suspend-PrintJob 875
Suspend-Service 921, 924
Suspend-VM 1099
Switch 60, 214, 220, 1097
SwitchParameter 1342
Sybase 794
Symbolic Link 723 ff.
SymLink 725 f.
Synopsis 1292
SYNOPSIS 1285
Syntaxfarbhervorhebung 353
System 1378 f.
System32 144, 150 f.
System ACL 978
System.ApplicationException 243
Systemattribut
– WMI 453
System.Boolean 273
System Center Virtual Machine Manager 669
System.Collections.Hashtable 209
System.Console 429, 678
System.Data 421
System.Data.ODBC 794, 847
System.Data.OleDb 794, 801
System.Data.OLEDB 794
System.Data.OracleClient 794, 801
System.Data.SqlClient 794, 801, 826
System.Data.SqlClient.SqlConnection 423
System.Data.SqlServerCe 794

System.DateTime 203, 423 f., 427
System.DbNull 805
System.Diagnostics 515
System.Diagnostics.EventLog 967
System.Diagnostics.Process 100, 114, 254, 915, 1320
Systemdienst 103, 450, 921
– auflisten 467
– überwachen 469
System.DirectoryServices 421, 1001 ff., 1011, 1017, 1020, 1027, 1068, 1080
System.DirectoryServices.ActiveDirectory 1068
System.Directoryservices.DirectoryEntry 423 ff.
System.dll 1379
System-DSN 846
Systemende 683
System.Enum 438
System.Environment 284, 856 ff., 973, 1303 f., 1309
SystemEvent 468, 583
System.Globalization.CultureInfo 680
System.Int32 172, 184
System.IO.Compression 727 ff.
System.IO.Directory 977
System.IO.DirectoryInfo 497, 695, 699
System.IO.DriveInfo 426, 429 ff., 437, 690 ff.
System.IO.DriveType 437
System.IO.File 977
System.IO.FileInfo 497, 695, 699, 717
Systemklassen
– WMI 452
Systemmanagement 447
System.Management 421, 1311
System.Management.Automation 96, 1299 ff., 1311, 1360
System.Management.Automation.Cmdlet 1299
System.Management.Automation.PathInfo 273 f.
System.Management.Automation.PSCustomObject 751
System.Management.Automation.PSDriveInfo 690
System.Management.ManagementObject 204
System Management Server 460
System.Math 427
System.Media.SoundPlayer 426
Systemmodul 38
System.Net.Mail 946 f.
System.Net.WebClient 424 f., 947 ff., 960
System.Object 120, 499, 1090, 1320, 1324
SystemParametersInfo 503
System.Random 186, 423
System.Reflection 431
System.Security 978
System.Security.AccessControl 977
System.ServiceProcess.ServiceController 921 ff., 1320
Systemstart 683
System.String 187, 192, 400, 1303
System.TimeSpan 165, 204
System.Type 119, 181, 263
System.ValueType 211
Systemwiederherstellung 869
System.Windows 1247
System.Windows.FontStyle 1250
System.Windows.Forms 717, 1242
System.Xml.Node 762
Sysvol 683

T

Tab Completion 309
Tabellenformatierung 249
TabPanel 1252
Tabulator 191
Tabulatorvervollständigung 309
Tag 1152
Tag-ContainerImage 1186
TAR 728
TaskScheduler 664
TCP/IP 930
tcsh 371
Team Foundation Server siehe TFS
Tee-Object 147 f.
Teilmenge 135
Telnet 283
Terminal Services 450
Terminating Error 234, 1328
Ternary Operator 221
Terrabyte 184
Test-32Bit 872
Test-64Bit 872
Test-AppLockerPolicy 910 ff.
Test-Assembly 703
Test-Connection 41, 105, 937 f.
Test-CustomerID 1283
Test-DbConnection 816
Test-DscConfiguration 608
Test-FileCatalog 95, 716
Test-IsAdmin 973
Test-JSON 772, 781
Test-ModuleManifest 637
Test-Path 273
Test Plan 1222
Test-PswaAuthorizationRule 357
Test-ServiceHealth 1083
Test-SqlScript 831
Test-TargetResource 629
Test-UserGroupMembership 1025
Test-VHD 1099, 1110
Test-Xml 759
Textanzeige 1255
Textdatei 150, 748
Texteingabefeld 508, 1243
TextInfo 194
Textpad 153
TFS 167, 1222
Thawte 529
this 500, 1252
Thread 109, 561
Thread-Modell 1247
ThrottleLimit 110, 291
throw 214, 243
ThrowTerminatingError() 1328
Thumbprint 992
TIFF 717
TimeOfDay 112
TimeSpan 514
Tivoli 470
TLS siehe Transport Layer Security
ToLower() 194
Ton 429
ToString() 120 f., 383, 430, 1320, 1369
TotalProcessorTime 263
ToTitleCase() 194
ToUpper() 194
TPM 745
Trace-Command 686
Tracing 515
Transaktion 376, 520
Transformation 1255
Translate() 982
Transport Layer Security 950, 959
trap 214
Trap 231, 234 ff.
Treiber 670
– ODBC 845
Trigger 544
Troubleshooting Pack 995
true 172, 181, 872
Trusted Host 300
Trusted Platform Module siehe TPM
Trustee 974
Try...Catch 1053
Try-Catch-Finally 231, 237
T-SQL 826 f.
Tuva 1123
Typ 172, 415, 1364, 1379
– Namensgebung 1379
Typadapter 172, 206
Typbezeichner 172
Type Cast siehe Typkonvertierung 178

types.ps1xml 76 f., 128, 131
Typisierung 171
Typkennzeichner siehe Typbezeichner 172
Typkonvertierung 138, 178
Typname siehe Typbezeichner 172

U

UAC 30, 155, 160, 319, 917
Überladung 228
Ubuntu 13, 53, 1135, 1139 f., 1173
ufw 401
Umgebungsvariable 449
– Linux 398
Umlaut 750
Umleitung 266
Unblock-File 162
Undefined 159
Undo 314
Undo-Transaction 376, 520 ff.
UniformGrid 1252
Uninstall-Package 351, 892
Uninstall-WindowsFeature 894
Unit Test 1259
Universal Coordinated Time 454, 490
Unix 3 f., 99 f., 168, 689, 723
Unlock-BitLocker 747
Unregister-PSSessionConfiguration 293, 296, 675
Unrestricted 159
Unterbrechungsfreie Stromversorgung siehe USV
Unternamensraum 1377
Unterordner 121, 698
Unterroutine 222
Unterschlüssel 269
until 214
Update 450, 670
– Einstellungen 868
– installieren 867
– suchen 866
UpdateColl 865
Update-Database 351
Update-Help 93
Update-Module 38, 1260
UseBasicParsing 950
UsePropertyCache 1014
User 602, 1007, 1017, 1031
user32.dll 503
User Account Control siehe Benutzerkontensteuerung
UserDomainName 973
UserName 973
User Settings 396
UseTestCertificate 356
UseTransaction 521
using 558
USV 873
UTF8 750
UWP 887

V

ValidateCount 1276
Validate-CustomerID 1283
ValidateLength 182, 1276, 1342
ValidateNotNull 1276, 1342
ValidatePattern 182, 1276, 1342
ValidateRange 183, 1276
ValidateScript 182, 1276
ValidateSet 183
ValueFromPipeline 1276, 1317 ff., 1324
ValueFromPipelineByPropertyName 1276, 1319
ValuesCollection 1005
ValueType 211
Variable 44, 119, 148, 170, 181, 258, 269
– Auflösung 188
– eingebaut 181, 389
– vordefiniert 181, 389
– Workflow 559
Variablenauflösung 188 f., 258
Variablenkennzeichner 148, 170
Variablentypisierung 171
VB 501 ff.
VBA 786
Verb 785
Verbindungszeichenfolge 423, 800, 808
Verbose 65 f., 1327
VerbosePreference 66, 181, 1327
VerbsCommon 1340
VerbsCommunications 1341
VerbsData 1341
VerbsDiagnostic 1341
VerbsLifeCycle 1341
VerbsSecurity 1341
Vererbung 458, 1368, 1381
Vererbungsdiagramm 1368
Vererbungshierarchie 471, 1042, 1368
– WMI 458
Vergleich 149
Vergleichsoperator 132
Verifikation 1376
VeriSign 529
Verknüpfung 722
Verzeichnisattribut 1013
Verzeichnisdienst 130, 460, 666, 1015, 1027
Verzeichnisdienstklasse 1015
Verzeichnisobjekt 1011, 1016
Verzweigung 147
VHD 1109 f., 1121, 1127
VHDX 1106, 1110, 1121
Video 1255
View 255
Vim 354
VirtualHardDisk 1110
Virtualisierung 1097, 1150
VirtualizingStackPanel 1252
Virtual Machine Platform 1141
Virtuelle Maschine siehe VM
Virtuelles System 1097
Virus 159
Visual Basic 501, 1371
Visual Basic 6.0 1376
Visual Basic for Applications siehe VBA
Visual Basic .NET 4, 1296
Visual Studio 346 ff., 351, 564, 1135, 1180, 1257, 1296 f., 1324
– Container 1177
– Erweiterung siehe 1297
Visual Studio Code 56, 153, 352, 394, 1155
Visual Studio Team Services siehe VSTS
Visual Web Developer Express 1297
VMBus 303
VMGUID 303
VMName 303
void 265, 445
VolumeLabel 425
Volume Shadow Copy Service siehe VSS 742
VSCode
– Visual Studio Code 352
VSCode-PowerShell 352, 394
VSI 1297
VSIX 1297
VSS 742
VSTS 167, 1222
VT100 771

W

WaitForAll 629
WaitForAny 629
WaitForSome 629
Wait-Job 534, 537
Wait-Process 920
Walter Skinner 1002
WarningAction 65, 238
WarningVariable 65
Warnung 65
WAS 923
WBEM 8, 447
WCOW 1135
WDAC 844

WebAdministration 1086 ff., 1133
Webanwendung 1375
Web Based Enterprise Management 447 f.
WebClient 963
Webdienst 957
Weblog 948, 1382
Webserver 271, 1091
Webservice 957 ff.
Web Service Description Language 958
Web Services Description Language siehe WSDL 957
Website 954, 1091, 1096
Well-Known GUID 1011
Well-Known Object 1011
Well-Known Security Principal 983
WellKnownSidType 984
Werkzeug 307
Wertemenge 205
Wertkopie 211 f.
WhatIf 63 ff., 181, 706, 876, 1281, 1332
WhatIfPreference 66
Where() 134 f.
WHERE 466
Where-Object 70 f., 98 ff., 115, 131 ff., 146, 150, 265, 922, 1300, 1311, 1324
while 214
Whistler 456
whoami.exe 321
Width 249
Wiederherstellungspunkt 869
Win32 451
Win32_Account 1003
Win32_ACE 734
Win32-API 503
Win32_Battery 873
Win32_Bios 858
Win32_BootConfiguration 858
Win32_CDRomdrive 871
Win32_CDRomDrive 486
Win32_CodecFile 879
Win32_ComponentCategory 467
Win32_ComputerShutdownEvent 464, 468, 583
Win32_Computersystem 856
Win32_Currenttime 204
Win32_Desktop 1003
Win32_Diskdrive 871
Win32_Group 1003
Win32_Keyboard 872
Win32_LocalTime 204
Win32_LogicalDisk 457 f., 468, 491, 690 ff., 1311
Win32_MappedLogicalDisk 694
Win32_MemoryDevice 871
Win32_NetworkAdapter 871
Win32_NetworkAdapterConfiguration 467, 927, 933 f.
Win32_NTLogEvent 467 ff., 584
Win32 OpenSSH 404
Win32_OperatingSystem 856 f.
Win32_OSRecoveryConfiguration 859
Win32_PerfRawData 970
Win32_PerfRawData_PerfOS_Processor 970
Win32_PerfRawData_PerfProc_Process 970
Win32_PingStatus 937 f.
Win32_PointingDevice 872
Win32_PowerManagementEvent 468, 583
Win32_Printer 873 ff., 927
Win32_Printjob 873 f.
Win32_Process 584
Win32_Processor 871 f., 1117
Win32_ProcessStartTrace 468, 583
Win32_Product 109, 877 ff.
Win32_Quickfixengineering 879
Win32_SecurityDescriptor 734
Win32_Service 467 ff., 584, 921
Win32_Share 731 f.
Win32_SoundDevice 871
Win32_SystemConfigurationChangeEvent 464, 468, 583
Win32_Tapedrive 871
Win32_TCPIPPrinterPort 873 f., 927
Win32_Trustee 734
Win32_USBController 872
Win32_UserAccount 149, 457, 1003
Win32_VideoController 477, 486, 871 ff.
Win32_Volume 694
Win32_WindowsProductActivation 859
window 1252
Windows
– Rolle 894, 1126
Windows 8 481
Windows 8.1 654
Windows 9x 459
Windows 10 5, 12, 20, 303, 308, 656
– Anniversary Update 5
Windows 2000 456
Windows Activation Service siehe WAS
WindowsApps 50
Windows-Authentifizierung 830
Windows Communication Foundation 450
Windows-Container 1135
Windows Container siehe Docker 1136
Windows Data Access Components siehe WDAC
Windows Defender 870, 1127
Windows Driver Model 460
Windows Explorer 743, 853
Windows Firewall 671, 938, 944
– Im Netzwerk abfragen 944
– Per PowerShell konfigurieren 938
Windows Forms 348, 1242 ff.
WindowsIdentity 973
Windows Installer 460, 909
Windows Management Framework 20, 448, 875
Windows Management Instrumentation 15
Windows ME 459
Windows Nano Server 12, 1123, 1152
Windows PowerShell XXIV, 3, 6
Windows PowerShell Community Extensions 660
Windows Pre Installation Environment siehe WinPE
Windows Presentation Foundation siehe WPF
Windows Remote Management 277
Windows Remote Management siehe WinRM 277
Windows Script Host 4, 17, 158
Windows Server 1709 12
Windows Server 2003 4, 456, 459, 1026
Windows Server 2012 481, 652, 1065
Windows Server 2012 R2 360, 654
Windows Server 2016 5, 12, 20, 303, 308, 656
Windows Server 2019 12
Windows Server 2022 12, 34, 322
Windows Server Container 1136
Windows Server Core 329, 1152
Windows Subsystem for Linux siehe WSL 1136
Windows Troubleshooting Platform 995
Windows Universal Platform siehe UWP
Windows Update 865, 868
– Agent API 1131
– Nano Server 1131
Windows Vista 1372
Windows Workflow Foundation 376
Windows XP 15, 456, 466
WinMgmt.exe 459 f.
WinPE 745
WinPSCompatSession 383
WinRM 277 ff., 459, 534, 864
WinSCP 962
WITHIN 466, 583

WKGUID 1011
WMI 3, 8, 15, 204, 277, 376, 447, 451, 458, 478, 1311, 1315, 1328
– Class Explorer 471
– Command Shell 15
– Data Query 467
– Ereignis 464
– Event Query 466 ff.
– Klasse 471
– Namespace 455
– Object Browser 470 f.
– Query Language 466, 480
– Repository 459, 464, 482
– Schema 458, 467
– Schema-Query 467
– Steuerung 459
WMI API 472
WMIClass 447, 478
WMI Object Browser 471
WMI Query Language siehe WQL
WMISEARCHER 447, 478 ff.
Word 139, 445, 790
Workflow 550, 555 ff., 564
– Designer 565
– Einschränkungen 555
– Persistenz 562
– Verschachtelt 560
WorkflowInfo 564
WorkingSet 130
WorkingSet64 101
Work Item 1228
Workspace Settings 396
World Wide Wings 1339
Wörterbuch 139
WPF 87 ff., 252, 329, 348, 364, 389, 433, 554, 664, 1242, 1247, 1255, 1376
WPF PowerShell Kit 664, 1247
WPK 664, 1247
WQL 466, 481, 582, 1311
WrapPanel 1252
Write-BZip2 728
Write-Clipboard 511
WriteDebug() 1328
Write-Error 258
WriteError() 1328, 1331
Write-EventLog 278, 376, 967 ff.
Write-GZip 728
Write-Host 88, 246, 258, 359, 557
WriteObject 1304, 1313
WriteObject() 1300, 1311
Write-Output 78
Write-Progress 267, 495
Write-Tar 34, 728
WriteVerbose() 1328, 1331
Write-Warn 258
WriteWarning() 1328, 1331
Write-Zip 728
WScript.Shell 722
WSDL 957
WSH 1008
WSL 371, 1136, 1141
WSMan 271, 301
WS-Management 277 ff., 301, 459, 472, 475 ff.
WT_SESSION 326
Wurzelnamensraum 1377
www.IT-Visions.de XXXIII, 651, 666, 677, 690, 816

X

x64 1097
x86 1097
XAML 554, 564, 1255
XamlReader 1256
XCopy-Deployment 1376, 1379
xDscDiagnostics 623
xDscWebService 618
X-Files 1002
XFilesServer 1002
XML 92, 506, 632, 739, 757 f., 766 f., 960, 1334
XML Application Markup Language siehe XAML
XmlAttribute 761
XmlDocument 767
XmlElement 761
XML-Schema 759
XML-Webservice 1377
XPathDocumentNavigator 761
XslCompiledTransform 768

Y

YAML 1178, 1235
Year 112
YesNo 1242
YesNoCancel 1242

Z

Zahl 183
Zahlenliteral 184
Zeichenkette 187 f., 196, 258, 1321, 1341
– ersetzen 195
– leer 128
– Operation 193
– trennen 195
– verbinden 196
Zeichensatz 750
Zeilenumbruch 68, 107
– Pipeline 107
Zeitmessung 514
Zeitplandienst 450
Zertifikat 356, 529, 992
– selbst signiert 530
Zertifikatsdatei 532
Zertifikatsspeicher 3, 269, 992
Zertifikatsverwaltung 529 f., 533
ZIP 726 ff., 790
ZipFile 727 ff.
zsh 371
Zufallszahl 185 f.
Zugriffsrechteliste 973, 979
Zugriff verweigert 490
Zuweisungsoperator 211
Zwischenablage 511
Zwischencode 1374
Zwischenschritt 146
Zwischenspeicher 798