Für alle PowerShell Versionen

holger SCHWICHTENBERG

WINDOWS PowerShell und PowerShell Core

Der schnelle Einstieg

Skriptbasierte Windows-Systemadministration für Windows, Linux und macOS

HANSER

Im Internet: Codebeispiele, Feedbackmöglichkeiten und Forum

www.IT-Visions.de
Dr. Holger Schwichtenberg

Schwichtenberg

Windows PowerShell und PowerShell Core
Der schnelle Einstieg

Bleiben Sie auf dem Laufenden!

Unser **Computerbuch-Newsletter** informiert Sie monatlich über neue Bücher und Termine. Profitieren Sie auch von Gewinnspielen und exklusiven Leseproben. Gleich anmelden unter

www.hanser-fachbuch.de/newsletter

Hanser Update ist der IT-Blog des Hanser Verlags mit Beiträgen und Praxistipps von unseren Autoren rund um die Themen Online Marketing, Webentwicklung, Programmierung, Softwareentwicklung sowie IT- und Projektmanagement. Lesen Sie mit und abonnieren Sie unsere News unter

www.hanser-fachbuch.de/update

Holger Schwichtenberg

Windows PowerShell und PowerShell Core Der schnelle Einstieg

Skriptbasierte Systemadministration für Windows, Linux und macOS

HANSER

Der Autor:
Dr. Holger Schwichtenberg, Essen
www.IT-Visions.de

Alle in diesem Buch enthaltenen Informationen, Verfahren und Darstellungen wurden nach bestem Wissen zusammengestellt und mit Sorgfalt getestet. Dennoch sind Fehler nicht ganz auszuschließen. Aus diesem Grund sind die im vorliegenden Buch enthaltenen Informationen mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autor und Verlag übernehmen infolgedessen keine juristische Verantwortung und werden keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieser Informationen – oder Teilen davon – entsteht.

Ebenso übernehmen Autor und Verlag keine Gewähr dafür, dass beschriebene Verfahren usw. frei von Schutzrechten Dritter sind. Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Buch berechtigt deshalb auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Bibliografische Information der Deutschen Nationalbibliothek:

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Dieses Werk ist urheberrechtlich geschützt.
Alle Rechte, auch die der Übersetzung, des Nachdruckes und der Vervielfältigung des Buches, oder Teilen daraus, vorbehalten. Kein Teil des Werkes darf ohne schriftliche Genehmigung des Verlages in irgendeiner Form (Fotokopie, Mikrofilm oder ein anderes Verfahren) – auch nicht für Zwecke der Unterrichtsgestaltung – reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

© 2018 Carl Hanser Verlag München, www.hanser-fachbuch.de
Lektorat: Sylvia Hasselbach
Copy editing: Petra Kienle, Fürstenfeldbruck
Umschlagdesign: Marc Müller-Bremer, München, www.rebranding.de
Umschlagrealisation: Stephan Rönigk
Layout: Kösel Media GmbH, Krugzell
Druck und Bindung: Hubert & Co. GmbH & Co. KG BuchPartner, Göttingen
Printed in Germany

Print-ISBN: 978-3-446-45214-5
E-Book-ISBN: 978-3-446-45281-7

Inhalt

Vorwort . . . . **XV**

Über den Autor Dr. Holger Schwichtenberg . . . . **XXI**

1 Erste Schritte mit der PowerShell **1**

1.1 Was ist die PowerShell? 1

1.2 Windows PowerShell versus PowerShell Core 2

1.3 Windows PowerShell herunterladen und auf anderen Windows-Betriebssystemen installieren 2

1.4 Die Windows PowerShell testen 7

1.4.1 PowerShell im interaktiven Modus 7

1.4.2 Installierte Version ermitteln 10

1.4.3 PowerShell im Skriptmodus 11

1.4.4 Skript eingeben 11

1.4.5 Skript starten 12

1.4.6 Skriptausführungsrichtlinie ändern 13

1.4.7 Farben ändern 16

1.5 Woher kommen die Commandlets? 17

1.6 PowerShell Community Extensions (PSCX) herunterladen und installieren . 18

1.7 Den Windows PowerShell-Editor „ISE“ verwenden 20

2 Fakten zur PowerShell **23**

2.1 Geschichte der PowerShell 23

2.2 Warum PowerShell einsetzen? 24

2.3 Einflussfaktoren auf die Entwicklung der PowerShell 28

2.4 Betriebssysteme mit vorinstallierter PowerShell 29

2.5 Anbindung an Klassenbibliotheken 31

2.6 PowerShell versus WSH 31

3 Einzelbefehle der PowerShell 35
3.1 Commandlets 35
3.1.1 Aufbau eines Commandlets 35
3.1.2 Aufruf von Commandlets 36
3.1.3 Commandlet-Parameter 36
3.1.4 Platzhalter bei den Parameterwerten 39
3.1.5 Abkürzungen für Parameter 40
3.1.6 Allgemeine Parameter (Common Parameters) 41
3.1.7 Dynamische Parameter 45
3.1.8 Zeilenumbrüche 45
3.1.9 PowerShell-Module 46
3.1.10 Prozessmodell 47
3.1.11 Aufruf von Commandlets aus anderen Prozessen heraus 47
3.1.12 Namenskonventionen 48
3.2 Aliase 48
3.2.1 Aliase auflisten 49
3.2.2 Neue Aliase anlegen 53
3.2.3 Aliase für Eigenschaften 54
3.3 Ausdrücke 56
3.4 Externe Befehle 57
3.5 Dateinamen 59
3.6 Aufgaben zu diesem Kapitel 59

4 Hilfefunktionen 61
4.1 Auflisten der verfügbaren Befehle 61
4.2 Volltextsuche 63
4.3 Erläuterungen zu den Befehlen 64
4.4 Hilfe zu Parametern 65
4.5 Hilfe mit Show-Command 67
4.6 Hilfefenster 69
4.7 Allgemeine Hilfetexte 70
4.8 Aktualisieren der Hilfedateien 70
4.9 Online-Hilfe 72
4.10 Fehlende Hilfetexte 73
4.11 Dokumentation der .NETKlassen 75
4.12 Aufgaben zu diesem Kapitel 78

5 Objektorientiertes Pipelining 79
5.1 Pipeline-Operator 79
5.2 .NET-Objekte in der Pipeline 80
5.3 Pipeline Processor 82
5.4 Pipelining von Parametern 83

5.5 Pipelining von klassischen Befehlen ... 86
5.6 Anzahl der Objekte in der Pipeline ... 87
5.7 Zeilenumbrüche in Pipelines ... 88
5.8 Zugriff auf einzelne Objekte aus einer Menge ... 88
5.9 Zugriff auf einzelne Werte in einem Objekt ... 90
5.10 Methoden ausführen ... 92
5.11 Analyse des Pipeline-Inhalts ... 94
5.12 Filtern ... 105
5.13 Zusammenfassung von Pipeline-Inhalten ... 108
5.14 „Kastrierung“ von Objekten in der Pipeline ... 109
5.15 Sortieren ... 110
5.16 Duplikate entfernen ... 111
5.17 Gruppierung ... 112
5.18 Berechnungen ... 114
5.19 Zwischenschritte in der Pipeline mit Variablen ... 114
5.20 Verzweigungen in der Pipeline ... 115
5.21 Vergleiche zwischen Objekten ... 117
5.22 Zusammenfassung ... 118
5.23 Aufgaben zu diesem Kapitel ... 119

6 PowerShell-Skripte ... 121
6.1 Skriptdateien ... 121
6.2 Start eines Skripts ... 123
6.3 Aliase für Skripte verwenden ... 125
6.4 Parameter für Skripte ... 125
6.5 Skripte dauerhaft einbinden (Dot Sourcing) ... 127
6.6 Das aktuelle Skriptverzeichnis ... 127
6.7 Sicherheitsfunktionen für PowerShell-Skripte ... 128
6.8 Anforderungsdefinitionen von Skripten ... 130
6.9 Skripte anhalten ... 131
6.10 Versionierung und Versionsverwaltung von Skripten ... 131
6.10.1 Versionierung ... 131
6.10.2 Versionsverwaltung (Versionskontrolle) ... 132
6.11 Aufgaben zu diesem Kapitel ... 133

7 Die PowerShell-Skriptsprache ... 135
7.1 Hilfe zur PowerShell-Skriptsprache ... 135
7.2 Befehlstrennung ... 136
7.3 Kommentare ... 136
7.4 Variablen ... 137
7.4.1 Variablen in der PowerShell ... 137

7.4.2 Typisierung ... 138
7.4.3 Datentypen/Typbezeichner in PowerShell ... 139
7.4.4 Typisierungszwang ... 141
7.4.5 Typkonvertierung (Typumwandlung) ... 142
7.4.6 Gültigkeitsbereiche (Scope) ... 143
7.4.7 Variablen leeren oder löschen ... 144
7.4.8 Variablentyp ermitteln ... 145
7.4.9 Vordefinierte Variablen ... 145
7.5 Variablenbedingungen ... 147
7.6 Zahlen ... 148
7.7 Zeichenketten (Strings) ... 150
7.7.1 Zeichenkettenliterale ... 150
7.7.2 Zeichenketten zusammensetzen ... 151
7.7.3 Variablenauflösung in Zeichenketten ... 151
7.7.4 Wiederholte Zeichenketten ... 153
7.7.5 Leere Zeichenketten ... 153
7.7.6 Sonderzeichen in Zeichenketten ... 154
7.7.7 Bearbeitungsmöglichkeiten für Zeichenketten ... 155
7.7.8 Zeichenketten ersetzen ... 157
7.7.9 Zeichenketten trennen und verbinden ... 158
7.8 Reguläre Ausdrücke ... 159
7.8.1 Mustervergleichsoperatoren ... 159
7.8.2 Allgemeiner Aufbau von regulären Ausdrücken ... 161
7.9 Datum und Uhrzeit ... 165
7.10 Arrays ... 167
7.10.1 Deklaration ... 167
7.10.2 Arrayoperationen ... 167
7.10.3 Array auflisten ... 169
7.10.4 Arrays verbinden ... 169
7.10.5 Mehrdimensionale Arrays ... 170
7.11 ArrayList ... 170
7.12 Assoziative Arrays (Hash-Tabellen) ... 171
7.13 Operatoren ... 172
7.13.1 Vergleichsoperatoren ... 172
7.13.2 Arithmetische Operatoren ... 172
7.13.3 Zuweisungsoperator ... 173
7.13.4 Bit-Operatoren ... 175
7.13.5 Aufrufoperator ... 175
7.14 Überblick über die Kontrollkonstrukte ... 176
7.15 Schleifen ... 177
7.16 Bedingungen ... 182
7.17 Unterroutinen (Prozedur/Funktionen) ... 184
7.17.1 Prozedur versus Funktion ... 184

7.17.2 Prozeduren ... 185
7.17.3 Funktionen (mit Rückgabewerten) ... 185
7.17.4 Art der Rückgabewerte ... 188
7.17.5 Parameterübergabe ... 188
7.18 Eingebaute Funktionen ... 190
7.19 Fehlerbehandlung ... 191
7.20 Objektorientiertes Programmieren mit Klassen ... 200
7.21 Aufgaben zu diesem Kapitel ... 202

8 Ausgaben ... 203
8.1 Ausgabe-Commandlets ... 203
8.2 Benutzerdefinierte Tabellenformatierung ... 206
8.3 Benutzerdefinierte Listenausgabe ... 209
8.4 Mehrspaltige Ausgabe ... 209
8.5 Out-GridView ... 210
8.6 Standardausgabe ... 212
8.7 Einschränkung der Ausgabe ... 215
8.8 Seitenweise Ausgabe ... 216
8.9 Ausgabe einzelner Werte ... 217
8.10 Details zum Ausgabeoperator ... 220
8.11 Ausgabe von Methodenergebnissen und Unterobjekten in Pipelines ... 223
8.12 Ausgabe von Methodenergebnissen und Unterobjekten in Zeichenketten ... 224
8.13 Unterdrückung der Ausgabe ... 224
8.14 Ausgaben an Drucker ... 225
8.15 Ausgaben in Dateien ... 225
8.16 Umleitungen (Redirection) ... 226
8.17 Fortschrittsanzeige ... 227
8.18 Sprachausgabe ... 227
8.19 Aufgaben zu diesem Kapitel ... 228

9 Benutzereingaben ... 229
9.1 Read-Host ... 229
9.2 Benutzerauswahl ... 230
9.3 Grafischer Eingabedialog ... 231
9.4 Dialogfenster ... 232
9.5 Authentifizierungsdialog ... 232
9.6 Zwischenablage (Clipboard) ... 234
9.7 Aufgaben zu diesem Kapitel ... 235

10 Das PowerShell-Navigationsmodell (PowerShell Provider) 237
10.1 Einführungsbeispiel: Navigation in der Registrierungsdatenbank 237
10.2 Provider und Laufwerke 239
10.3 Navigationsbefehle 241
10.4 Pfadangaben. 241
10.5 Beispiel 243
10.6 Eigene Laufwerke definieren 244
10.7 Aufgaben zu diesem Kapitel 245

11 Fernausführung (Remoting) 249
11.1 RPC-Fernabfrage ohne WS-Management 250
11.2 Anforderungen an PowerShell Remoting 251
11.3 Rechte für PowerShell-Remoting 252
11.4 Einrichten von PowerShell Remoting 253
11.5 Überblick über die Fernausführungs-Commandlets 255
11.6 Interaktive Fernverbindungen im Telnet-Stil 256
11.7 Fernausführung von Befehlen 257
11.8 Parameterübergabe an die Fernausführung 261
11.9 Fernausführung von Skripten 262
11.10 Ausführung auf mehreren Computern 263
11.11 Sitzungen 264
11.11.1 Commandlets zur Sitzungsverwaltung 265
11.11.2 Sitzungen erstellen 266
11.11.3 Kopieren von Dateien in Sitzungen 266
11.11.4 Schließen von Sitzungen 267
11.11.5 Sitzungskonfigurationen 267
11.11.6 Zugriffsrechte für Fernaufrufe 267
11.12 Implizites Remoting 269
11.13 Zugriff auf entfernte Computer außerhalb der eigenen Domäne 270
11.14 Verwaltung des WS-Management-Dienstes 274
11.15 PowerShell Direct für Hyper-V 275
11.16 Praxisbeispiel zu PowerShell Direct 277
11.17 Aufgaben zu diesem Kapitel 279

12 Verwendung von .NET-Klassen 281
12.1 Microsoft Developer Network (MSDN) 281
12.2 Erzeugen von Instanzen 282
12.3 Parameterbehaftete Konstruktoren 284
12.4 Initialisierung von Objekten 286
12.5 Nutzung von Attributen und Methoden 286
12.6 Statische Mitglieder in .NET-Klassen und statische .NET-Klassen 289

12.7 Generische Klassen nutzen ... 293
12.8 Zugriff auf bestehende Objekte ... 294
12.9 Laden von Assemblies ... 294
12.10 Objektanalyse ... 297
12.11 Auflistungen (Enumerationen) ... 298
12.12 Verknüpfen von Aufzählungswerten ... 299
12.13 Aufgaben zu diesem Kapitel ... 299

13 Verwendung von COM-Klassen ... 301
13.1 Erzeugen von COM-Instanzen ... 301
13.2 Nutzung von Attributen und Methoden ... 302
13.3 Liste aller COM-Klassen ... 303
13.4 Holen bestehender COM-Instanzen ... 304
13.5 Distributed COM (DCOM) ... 304
13.6 Aufgaben zu diesem Kapitel ... 305

14 Zugriff auf die Windows Management Instrumentation (WMI) ... 307
14.1 WMI in der PowerShell ... 307
14.2 Open Management Infrastructure (OMI) ... 309
14.3 Abruf von WMI-Objektmengen ... 309
14.4 Fernzugriffe ... 310
14.5 Filtern und Abfragen ... 311
14.6 Liste aller WMI-Klassen ... 315
14.7 Hintergrundwissen: WMI-Klassenprojektion mit dem PowerShell-WMI-Objektadapter ... 315
14.8 Beschränkung der Ausgabeliste bei WMI-Objekten ... 320
14.9 Zugriff auf einzelne Mitglieder von WMI-Klassen ... 322
14.10 Werte setzen in WMI-Objekten ... 322
14.11 Umgang mit WMI-Datumsangaben ... 324
14.12 Methodenaufrufe ... 325
14.13 Neue WMI-Instanzen erzeugen ... 326
14.14 Instanzen entfernen ... 327
14.15 Commandlet Definition XML-Datei (CDXML) ... 328
14.16 Aufgaben zu diesem Kapitel ... 330

15 Fehlersuche ... 333
15.1 Detailinformationen ... 333
15.2 Einzelschrittmodus ... 335
15.3 Zeitmessung ... 336
15.4 Ablaufverfolgung (Tracing) ... 336
15.4.1 Tracesources ... 336

15.4.2 Verfolgung eines Einzelbefehls 337
15.4.3 Generelle Ablaufverfolgung 337
15.5 Erweiterte Protokollierung aktivieren 338
15.6 Script-Debugging in der ISE 339
15.7 Kommandozeilenbasiertes Script-Debugging 340

16 Standardeinstellungen ändern mit Profilskripten 343
16.1 Profilpfade 343
16.2 Ausführungsreihenfolge 345
16.3 Beispiel für eine Profildatei 345
16.4 Starten der PowerShell ohne Profilskripte 346

17 PowerShell-Module 347
17.1 Überblick über die Commandlets 347
17.2 Modularchitektur 348
17.3 Aufbau eines Moduls 349
17.4 Module aus dem Netz herunterladen und installieren mit PowerShellGet 350
17.5 Module manuell installieren 357
17.6 Doppeldeutige Namen 357
17.7 Importieren von Modulen 360
17.8 Entfernen von Modulen 363

18 Praxislösungen 365
18.1 Leere Ordner löschen 365
18.2 Fotos nach Aufnahmedatum sortieren 366
18.3 Zufällige Dateisystemstruktur erzeugen 368
18.4 Freigaben anlegen 369
18.4.1 WMI-Klassen 369
18.4.2 Freigaben auflisten 371
18.4.3 Freigaben anlegen (mit WMI) 371
18.4.4 Berechtigungen auf Freigaben setzen 373
18.4.5 Freigaben anlegen (mit PowerShell-Commandlets) 376
18.4.6 Freigaben anlegen auf Basis einer XML-Datei 379
18.5 Netzwerkkonfiguration 381
18.6 Massenanlegen von Registry-Schlüsseln 383
18.7 Massenanlegen von Active-Directory-Benutzerkonten 384
18.7.1 Benutzer und Gruppen anlegen aus einer Datenbank (Microsoft Access oder Microsoft SQL Server) 385
18.7.2 Benutzer und Gruppen anlegen aus einer CSV-Datei 399
18.8 Massenanlegen von IIS-Websites 402

18.9 Softwareinstallation ... 404
18.10 Virtuelles System in Hyper-V anlegen ... 406

19 PowerShell Core 6.0 für Windows, Linux und macOS ... 409

19.1 Geschichte der PowerShell Core ... 409
19.2 Vergleich zwischen Windows PowerShell und PowerShell Core ... 410
19.3 Motivation für den Einsatz der PowerShell Core auf Linux und macOS ... 411
19.4 PowerShell Core installieren und testen ... 413
19.4.1 Installation und Test auf Windows ... 413
19.4.2 Installation und Test auf Ubuntu Linux ... 415
19.4.3 Installation und Test auf macOS ... 417
19.5 Funktionsumfang der PowerShell Core ... 418
19.6 Entfallene Commandlets in PowerShell Core ... 420
19.6.1 PowerShell-Kern-Befehle, die in PowerShell Core fehlen ... 420
19.6.2 Befehle, die zusätzlich unter Linux und macOS fehlen ... 423
19.7 Erweiterungsmodule nutzen in PowerShell Core ... 425
19.7.1 Eingebaute Module in PowerShell Core ... 425
19.7.2 Windows-Module in PowerShell Core nutzen ... 427
19.7.3 Module der PowerShell Gallery in PowerShell Core ... 427
19.8 Geänderte Funktionen in PowerShell Core ... 428
19.8.1 Änderungen der Parameter von pwsh.exe ... 428
19.8.2 Geänderte Pfade ... 428
19.8.3 Weitere Änderungen ... 430
19.9 Neue Funktionen der PowerShell Core ... 430
19.9.1 Neue eingebaute Variablen ... 430
19.9.2 Neue Commandlets ... 431
19.9.3 Sonstige Verbesserungen in PowerShell Core ... 431
19.10 PowerShell-Core-Konsole ... 432
19.11 VSCode-PowerShell als Editor für PowerShell Core ... 433
19.11.1 PowerShell Core zur Skriptausführung konfigurieren ... 434
19.11.2 PowerShell Core für das Terminalfenster in VSCode festlegen ... 436
19.12 Verwendung auf Linux und macOS ... 438
19.12.1 Praxisbeispiel: Linux-Benutzerliste auswerten ... 440
19.12.2 Praxisbeispiel: Offene Ports auswerten ... 441
19.12.3 Praxisbeispiel: Dateien unter Linux und macOS verstecken und versteckte Dateien auflisten ... 443
19.13 PowerShell-Remoting via SSH ... 444
19.13.1 OpenSSH auf Windows ... 444
19.13.2 PowerShell Remoting mit SSH ... 446
19.14 Dokumentation zur PowerShell Core ... 448
19.15 Quellcode zur PowerShell Core ... 450

20 Weiterführende Literatur 453

21 Lösungen 457

Index 473

Vorwort

Liebe Leserin, lieber Leser,

herzlich willkommen zu meinem Buch „Windows PowerShell und PowerShell Core – Der schnelle Einstieg: Skriptbasierte Systemadministration für Windows, Linux und macOS“!

Was ist das Konzept dieses Buchs?

Seit vielen Jahren veröffentliche ich sehr erfolgreich das „PowerShell-Praxisbuch“ im Carl Hanser Verlag, welches mittlerweile auf über 1200 Seiten angewachsen ist. Mit dem vor Ihnen liegenden Buch „PowerShell – Der schnelle Einstieg“ erhalten Sie eine deutlich kompaktere Einführung in die PowerShell. Gegenüber dem Praxisbuch enthält dieses Einsteigerbuch am Ende der meisten Kapitel Aufgaben mit Lösungen zur Kontrolle oder Verfestigung des Lernerfolgs.

Der Schwerpunkt dieses Buchs liegt auf der Windows PowerShell 5.1 im Einsatz auf Windows. Am Ende gibt es aber auch ein Kapitel, das die Unterschiede zu der plattformneutralen PowerShell Core 6.0 und ihrem Einsatz auf Linux und macOS beschreibt.

Wer bin ich?

Mein Name ist Holger Schwichtenberg, ich bin derzeit 45 Jahre alt und habe im Fachgebiet Wirtschaftsinformatik promoviert. Ich lebe (in Essen, im Herzen des Ruhrgebiets) davon, dass mein Team und ich im Rahmen unserer Firma *www.IT-Visions.de* anderen Unternehmen bei der Entwicklung von .NET-, Web- und PowerShell-Anwendungen beratend und schulend zur Seite stehen. Zudem entwickeln wir im Rahmen der 5Minds IT-Solutions GmbH & Co. KG Software (*www.5Minds.de*) im Auftrag von Kunden aus zahlreichen Branchen.

Es ist mein Hobby und Nebenberuf, IT-Fachbücher zu schreiben. Dieses Buch ist, unter Mitzählung aller nennenswerten Neuauflagen, das 68. Buch, das ich allein oder mit Co-Autoren geschrieben habe. Meine weiteren Hobbys sind Mountain Biking, Laufsport, 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 3.0, 4.0, 5.x oder 6.x unterstützten Betriebssystemen? (Windows 7/8/8.1/10, Windows Server 2008/2008 R2/2012/2012 R2/2016) Hinweis: Die PowerShell Core 6.0 für Linux und macOS wird nur als Randthema kurz in diesem Buch behandelt, da es hier bislang kaum Befehle für die PowerShell gibt!	○ 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/PowerShell Core 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 (oder meinem umfangreicheren „PowerShell Praxisbuch“) nachzuschlagen, da das Buch auf rund 400 Seiten nicht alle Details erläutern kann?	○ Ja	○ Nein

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

Sind in diesem Buch alle Features der PowerShell beschrieben?

Die PowerShell umfasst mittlerweile über 1500 Commandlets mit jeweils zahlreichen Optionen. Zudem gibt es unzählige Erweiterungen mit vielen hundert weiteren Commandlets. Zudem existieren zahlreiche Zusatzwerkzeuge. Es ist allein schon aufgrund der Vorgaben des Verlags für den Umfang des Buchs nicht möglich, alle Commandlets und Parameter hier auch nur zu erwähnen. Zudem habe ich – obwohl ich selbst fast jede Woche mit der PowerShell in der Praxis arbeite – immer noch nicht alle Commandlets und alle Parameter jemals eingesetzt. Ich beschreibe in diesem Buch, was ich selbst in der Praxis, in meinen Schulungen und bei Kundeneinsätzen verwende. Es macht auch keinen Sinn, jedes Detail der PowerShell hier zu dokumentieren. Stattdessen gebe ich Ihnen **Hilfe zur Selbsthilfe**, damit Sie die Konzepte gut verstehen und sich dann Sonderfälle selbst erarbeiten können.

Wie aktuell ist dieses Buch?

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

- Es erscheinen in kurzer Abfolge immer neue Produkte.
- Produkte erscheinen schon in frühen Produktstadien als „Preview" mit Versionsnummern wie 0.1.
- Produkte ändern sich häufig. Aufwärts- und Abwärtskompatibilität ist kein Ziel mehr. Es wird erwartet, dass Sie Ihre Lösungen ständig den neuen Gegebenheiten anpassen.
- Produkte werden nicht mehr so ausführlich dokumentiert wie früher. Teilweise erscheint die Dokumentation erst deutlich nach dem Erscheinen der Software.
- Produkte werden schnell auch wieder abgekündigt, wenn sie sich aus der Sicht der Hersteller bzw. aufgrund des Nutzerfeedbacks nicht bewährt haben.

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

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

Wem ist zu danken?

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

- Frau Sylvia Hasselbach, die mich schon seit 20 Jahren als Lektorin begleitet und die dieses Buchprojekt beim Carl Hanser Verlag koordiniert und vermarktet,
- Frau Petra Kienle, die meine Tippfehler gefunden und sprachliche Ungenauigkeiten eliminiert hat,
- meiner Frau und meinen Kindern dafür, dass sie mir das Umfeld geben, um neben meinem Hauptberuf an Büchern wie diesem zu arbeiten.

Woher bekommen Sie die Beispiele aus diesem Buch?

Unter *http://www.powershell-doktor.de/leser* biete ich ein **ehrenamtlich betriebenes** Webportal für Leser meiner Bücher an. In diesem Portal können Sie

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

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

Wie sind die Programmcodebeispiele organisiert?

Sie erhalten die Beispiele zu diesem Buch in Form einer RAR-Archivdatei. Die Beispiele sind im Archiv organisiert nach Kapitelnamen (verkürzt). In diesem Buch wird für den Zugriff auf die Beispieldateien das X:-Laufwerk verwendet. Bitte legen Sie entweder ein Laufwerk X: an oder ändern Sie den Laufwerksbuchstaben in den Skripten.

```
PowerShell
PS x:\> Dir


    Directory: W:\PSE_Skripte


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----       04.03.2018     20:23                =Praxislösungen
d-----       29.06.2017     23:56                Aliase
d-r---       05.07.2017     09:43                Ausgaben
d-----       02.07.2017     23:29                Benutzereingaben
d-r---       21.04.2017     19:13                COM
d-r---       30.05.2017     00:28                Commandlets
d-r---       04.03.2018     20:22                DOTNET
d-----       08.03.2018     18:21                ErsteSchritte
d-----       24.04.2017     09:55                Fehlersuche
d-r---       29.06.2017     23:34                Hilfe
d-r---       04.03.2018     18:08                Module
d-r---       26.03.2014     12:49                Navigation
d-r---       04.06.2017     11:21                Pipelining
d-----       22.09.2017     19:14                PowerShellLanguage
d-----       29.05.2017     23:57                PowerShellOOP
d-r---       30.06.2017     20:26                Profile
d-----       07.03.2018     18:14                PSCore
d-----       04.07.2017     17:52                Remoting
d-r---       30.08.2017     13:24                Scripting
d-r---       04.11.2017     07:19                Werkzeuge
d-r---       17.05.2016     13:28                WMI
d-r---       26.03.2014     12:49                WPS versus VBS


PS x:\> _
```

Wo können Sie sich schulen oder beraten lassen?

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

Zum Schluss des Vorworts ...

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

Dr. Holger Schwichtenberg

Essen, im Frühjahr 2018

Über den Autor Dr. Holger Schwichtenberg

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

- Softwareprojektleiter im Bereich Microsoft/.NET bei der 5minds IT-Solutions GmbH & Co. KG in Gelsenkirchen (*www.5minds.de*)

- Über 65 Fachbücher beim Carl Hanser Verlag, bei O'Reilly, Microsoft Press und Addison-Wesley sowie mehr als 1000 Beiträge in Fachzeitschriften
- Gutachter in den Wettbewerbsverfahren der EU gegen Microsoft (2006–2009)
- Ständiger Mitarbeiter der Zeitschriften iX (seit 1999), dotnetpro (seit 2000) und Windows Developer (seit 2010) sowie beim Online-Portal *heise.de* (seit 2008)
- Regelmäßiger Sprecher auf nationalen und internationalen Fachkonferenzen (z. B. Microsoft TechEd, Microsoft Technical Summit, Microsoft IT Forum, BASTA, BASTA-on-Tour, .NET Architecture Camp, Advanced Developers Conference, Developer Week, OOP, DOTNET Cologne, MD DevDays, Community in Motion, DOTNET-Konferenz, VS One, NRW.Conf, Net.Object Days, Windows Forum)

- Zertifikate und Auszeichnungen von Microsoft:
  - Bereits 14-mal ausgezeichnet als Microsoft Most Valuable Professional (MVP)
  - Zertifiziert als Microsoft Certified Solution Developer (MCSD)
- Thematische Schwerpunkte:
  - Softwarearchitektur, mehrschichtige Softwareentwicklung, Softwarekomponenten, SOA
  - Microsoft .NET Framework, Visual Studio, C#, Visual Basic
  - .NET-Architektur/Auswahl von .NET-Technologien
  - Einführung von .NET Framework und Visual Studio/Migration auf .NET
  - Webanwendungsentwicklung und Cross-Plattform-Anwendungen mit HTML, ASP.NET, ASP.NET Core, JavaScript/TypeScript und Webframeworks wie Angular
  - Enterprise .NET, verteilte Systeme/Webservices mit .NET insbes. Windows Communication Foundation und WebAPI
  - Relationale Datenbanken, XML, Datenzugriffsstrategien
  - Objektrelationales Mapping (ORM), insbesondere ADO.NET Entity Framework und EF Core
  - Windows PowerShell, PowerShell Core und Windows Management Instrumentation (WMI)
- Ehrenamtliche Community-Tätigkeiten:
  - Vortragender für die International .NET Association (INETA)
  - Betrieb diverser Community-Websites: *www.dotnetframework.de*, *www.entwickler-lexikon.de*, *www.windows-scripting.de*, *www.aspnetdev.de* u. a.
- Firmenwebsites: *http://www.IT-Visions.de* und *http://www.5minds.de*
- Weblog: *http://www.dotnet-doktor.de*
- Kontakt: *kundenteam@IT-Visions.de* sowie *Telefon 02 01-64 95 90-0*

1 Erste Schritte mit der PowerShell

Das DOS-ähnliche Kommandozeilenfenster hat viele Windows-Versionen in beinahe unveränderter Form überlebt. Mit der Windows PowerShell (WPS) besitzt Microsoft seit dem Jahr 2006 einen Nachfolger, der es mit den Unix-Shells aufnehmen kann und diese in Hinblick auf Eleganz und Robustheit in einigen Punkten auch überbieten kann. Die PowerShell ist eine Adaption des Konzepts von Unix-Shells auf Windows unter Verwendung des .NET Frameworks und mit Anbindung an die Windows Management Instrumentation (WMI). Seit dem Jahr 2017 gibt es die PowerShell auch für Linux und macOS als „PowerShell Core".

1.1 Was ist die PowerShell?

In einem Satz: Die **Windows PowerShell (WPS)** ist eine .NET-basierte Umgebung für interaktive Systemadministration und Skripting auf der Windows-Plattform. **PowerShell Core (PS Core)** ist eine .NET Core-basierte Umgebung für interaktive Systemadministration und Skripting auf Windows, Linux und MacOS.

Die Kernfunktionen der PowerShell sind:

- Zahlreiche eingebaute Befehle, die „Commandlets" genannt werden
- Zugang zu allen Systemobjekten, die durch COM-Bibliotheken, das .NET Framework und die Windows Management Instrumentation (WMI) bereitgestellt werden
- Robuster Datenaustausch zwischen Commandlets durch Pipelines basierend auf typisierten Objekten
- Ein einheitliches Navigationsparadigma für verschiedene Speicher (z. B. Dateisystem, Registrierungsdatenbank, Zertifikatsspeicher, Active Directory und Umgebungsvariablen)
- Eine einfach zu erlernende, aber mächtige Skriptsprache mit wahlweise schwacher oder starker Typisierung
- Ein Sicherheitsmodell, das die Ausführung unerwünschter Skripte unterbindet
- Integrierte Funktionen für Ablaufverfolgung und Debugging
- Die PowerShell kann um eigene Befehle erweitert werden.
- Die PowerShell kann in eigene Anwendungen integriert werden (Hosting).

1.2 Windows PowerShell versus PowerShell Core

Die Windows PowerShell 5.1 ist weit mächtiger als die PowerShell Core 6.0, weil die PowerShell Core einen Neustart des PowerShell-Entwicklungsprojekts in Hinblick auf Plattformunabhängigkeit darstellt. In PowerShell Core fehlen viele Commandlets der Grundausstattung der Windows PowerShell und viele der verfügbaren PowerShell-Erweiterungsmodule laufen bisher nicht in der PowerShell Core.

Details zu den Funktionseinschränkungen der PowerShell Core lesen Sie in Kapitel 19 „PowerShell Core 6.0 für Windows, Linux und MacOS“.

TIPP: Wenn Sie unter Windows arbeiten, sollten Sie daher vorerst noch die Windows PowerShell (nach Möglichkeit in der aktuellen Version 5.1) verwenden.

Unter Linux und MacOS gibt es keine Windows PowerShell. Hier können Sie die PowerShell Core 6.0 verwenden. Der Wert der PowerShell Core unter Linux und MacOS liegt in den mächtigen Pipelining- sowie Ein- und Ausgabe-Commandlets. Für konkrete Zugriffe auf das Betriebssystem gibt es hingegen für die PowerShell Core unter MacOS und Linux noch fast keine Commandlets. Man wird also hier immer klassische Linux- und MacOS-Kommandozeilenbefehle mit zeichenkettenbasierter Verarbeitung in die PowerShell einbinden. Wie dies geht, wird im Kapitel 19 erklärt.

1.3 Windows PowerShell herunterladen und auf anderen Windows-Betriebssystemen installieren

Die Windows PowerShell 5.1 ist in Windows 10 (ab Anniversary Update) und Windows Server 2016 bereits im Standard installiert.

Wenn Sie nicht Windows 10 oder Windows Server 2016 benutzen, müssen Sie die PowerShell 5.1 erst installieren.

Die nachträgliche Installation der Windows PowerShell 5.1 ist auf folgenden Betriebssystemen möglich:

- Windows Server 2012 R2
- Windows Server 2012
- Windows 2008 R2
- Windows 8.1
- Windows 7

Die Windows PowerShell 5.1 wird auf diesen Betriebssystemen als Teil des Windows Management Framework 5.1 (WMF) installiert – *https://www.microsoft.com/en-us/download/details.aspx?id=54616*.

Bei der Installation ist zu beachten, dass jeweils das .NET Framework 4.5.2 oder höher vorhanden sein muss. Auch mit .NET Framework 4.6.x und 4.7 funktioniert die PowerShell 5.1.

Das WMF-5.1-Installationspaket betrachtet sich als Update für Windows (KB3191566 für Windows 7 und Windows Server 2008 R2 bzw. KB3191564 für Windows 8.1 und Windows Server 2012 R2 sowie KB3191565 für Server 2012).

✔	W2K12-KB3191565-x64.msu	20.6 MB
✔	Win7AndW2K8R2-KB3191566-x64.zip	64.9 MB
✔	Win7-KB3191566-x86.zip	42.7 MB
✔	Win8.1AndW2K12R2-KB3191564-x64.msu	19.0 MB
✔	Win8.1-KB3191564-x86.msu	14.5 MB

Bild 1.1 Installationspaket für PowerShell 5.1 als Erweiterung

Installationsordner

Die Windows PowerShell installiert sich in folgendes Verzeichnis: *%systemroot%\system32\WindowsPowerShell\V1.0* (für 32-Bit-Systeme).

ACHTUNG: Dabei ist das „V1.0“ im Pfad tatsächlich richtig: Microsoft hat dies seit Version 1.0 nicht verändert. Geplant war wohl eine „Side-by-Side“-Installationsoption wie beim .NET Framework. Doch später hat sich Microsoft dann entschieden, dass eine neue PowerShell immer die alte überschreibt.

Auf 64-Bit-Systemen gibt es die PowerShell zweimal, einmal als 64-Bit-Version in *%systemroot%\system32\WindowsPowerShell\V1.0* und einmal als 32-Bit-Version. Letztere findet man unter *%systemroot%\Syswow64\WindowsPowerShell\V1.0*. Die 32-Bit-Version braucht man, wenn man eine Bibliothek nutzen will, für die es keine 64-Bit-Version gibt, z. B. den Zugriff auf Microsoft-Access-Datenbanken.

Es handelt sich auch dabei nicht um einen Tippfehler: Die 64-Bit-Version befindet sich in einem Verzeichnis, das „32“ im Namen trägt, und die 32-Bit-Version in einem Verzeichnis mit „64“ im Namen!

Die 32-Bit-Version der PowerShell und die 64-Bit-Version der PowerShell sieht man im Startmenü: Die 32-Bit-Version hat den Zusatz „(x86)“. Die 64-Bit-Version hat keinen Zusatz. Auch den Editor „ISE“ gibt es in einer 32- und einer 64-Bit-Version.

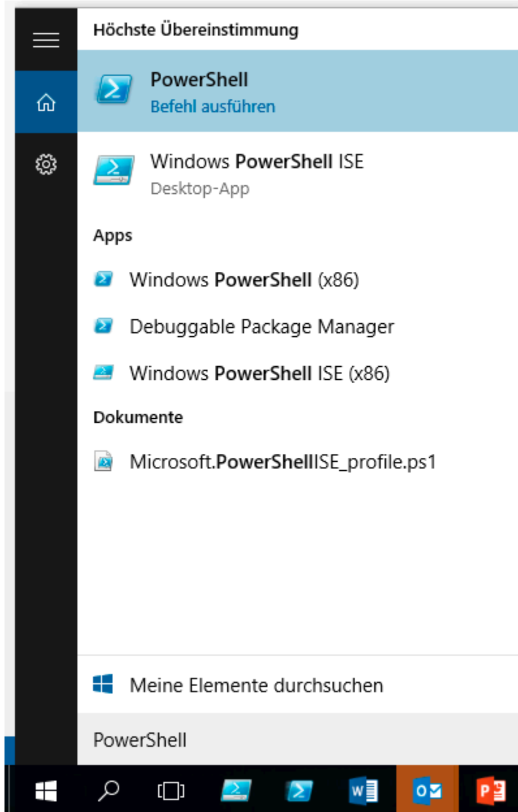

Bild 1.2 PowerShell-Einträge im Windows-10-Startmenü

TIPP: Unter Windows 8.x empfiehlt sich der Einsatz der Erweiterung *http://www.classicshell.net*, die das klassische Startmenü in Windows 8.x zurückbringt. Der Rückgriff auf ein Startmenü hat nicht nur mit Nostalgie zu tun, sondern auch ganz handfeste praktische Gründe: Der kachelbasierte Startbildschirm von Windows 8.x findet leider zum Suchbegriff „PowerShell" weder die PowerShell ISE noch die 32-Bit-Variante der PowerShell.

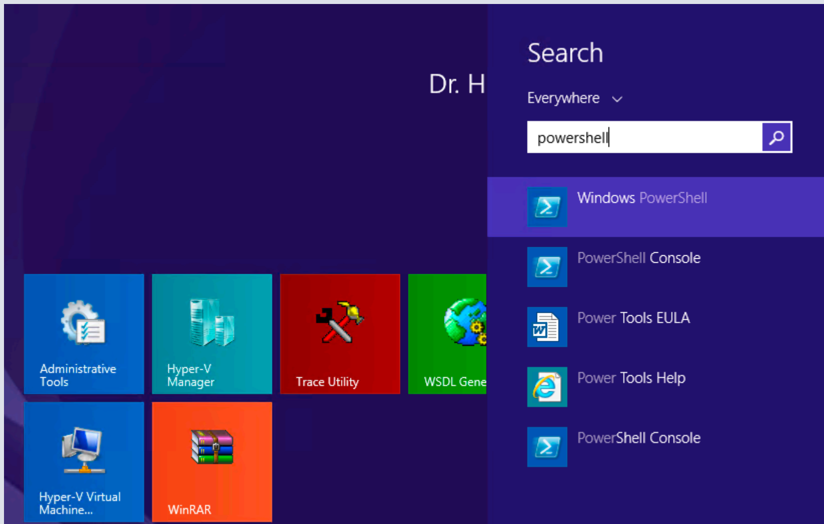

Bild 1.3 Versagen auf ganzer Linie: Der kachelbasierte Startbildschirm von Windows 8.x findet leider zum Suchbegriff „PowerShell“ weder die ISE noch die 32-Bit-Variante der PowerShell. In Windows 10 ist das Problem behoben.

Unter Windows 8.x geht es mit Classic Shell:

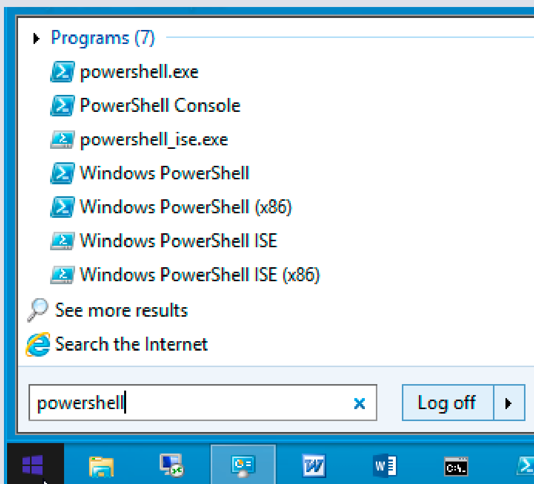

Bild 1.4 Die Classic Shell findet alle Einträge zur Windows PowerShell.

Ereignisprotokoll „PowerShell“

Durch die Installation der PowerShell wird in Windows auch ein neues Ereignisprotokoll „PowerShell“ angelegt, in dem die PowerShell wichtige Zustandsänderungen der PowerShell protokolliert.

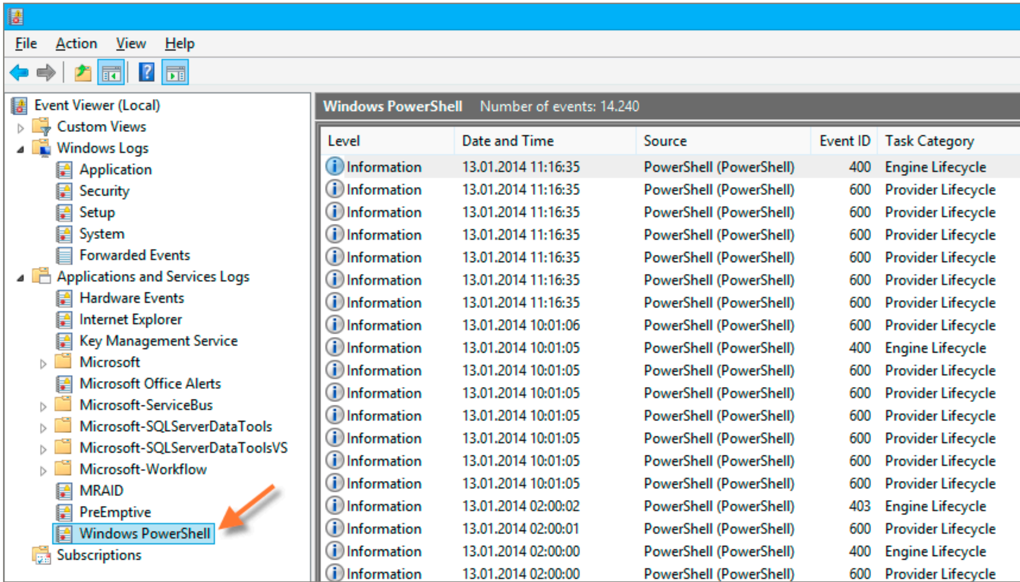

Bild 1.5 Ereignisprotokoll „Windows PowerShell“

Deinstallation

Falls man die PowerShell deinstallieren möchte, muss man dies in der Systemsteuerung unter „Programme und Funktionen/Installierte Updates anzeigen“ tun und dort das „Microsoft Windows Management Framework“ deinstallieren.

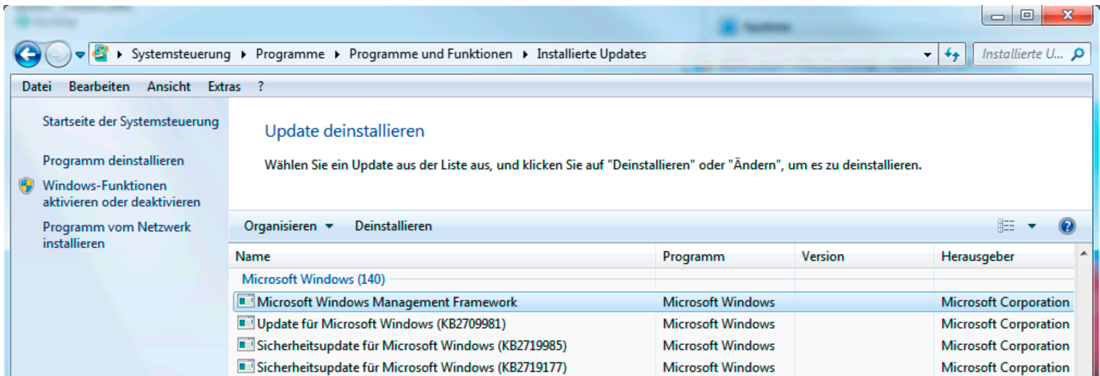

Bild 1.6 Deinstallation der PowerShell durch Deinstallation des WMF

1.4 Die Windows PowerShell testen

Dieses Kapitel stellt einige Befehle vor, mit denen Sie die PowerShell-Funktionalität ausprobieren können. Die PowerShell verfügt über zwei Modi (interaktiver Modus und Skriptmodus), die hier getrennt behandelt werden.

1.4.1 PowerShell im interaktiven Modus

Der erste Test verwendet die PowerShell im interaktiven Modus.

Starten Sie bitte die PowerShell. Es erscheint ein leeres PowerShell-Konsolenfenster. Auf den ersten Blick ist kein großer Unterschied zur herkömmlichen Konsole zu erkennen. Allerdings steckt in der PowerShell mehr Kraft im wahrsten Sinne des Wortes.

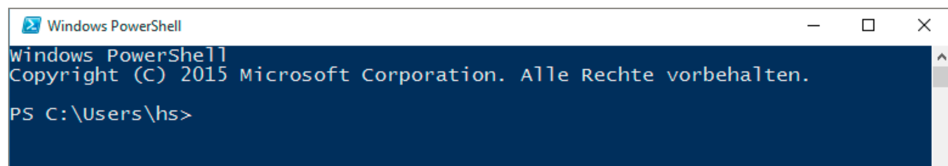

Bild 1.7 Leeres PowerShell-Konsolenfenster

Geben Sie an der Eingabeaufforderung „Get-Process“ ein (wobei die Groß-/Kleinschreibung irrelevant ist. Das gilt nicht nur für Windows, sondern auch für MacOS und Linux!) und drücken Sie dann die ENTER-Taste. Es erscheint eine Liste aller Prozesse, die auf dem lokalen Computer laufen. Dies war Ihre erste Verwendung eines einfachen PowerShell-Commandlets.

HINWEIS: Beachten Sie bitte, dass die Groß-/Kleinschreibung keine Rolle spielt, da PowerShell keine Unterschiede zwischen groß- und kleingeschriebenen Commandlet-Namen macht.

Geben Sie an der Eingabeaufforderung „Get-Service i*“ ein. Jetzt erscheint eine Liste aller installierten Dienste auf Ihrem Computer, deren Namen mit dem Buchstaben „i“ beginnen. Hier haben Sie ein Commandlet mit Parametern verwendet.

```
Windows PowerShell

PS C:\Documents\hs> get-process

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
     90       5     6504       7752    79     0,03   1336 Bildschirmpausenreminderdienst
    107       3     1236       1260    30     0,06   4232 cidaemon
     82       3     1260       1008    30     0,06   4712 cidaemon
     82       3     1336       1008    30     0,06   4892 cidaemon
    414       8     2764        636    43     0,95   1376 cisvc
   1002       9     2460       6360    29     5,41   1652 csrss
     69       3      496       2876    16     0,09   3936 ctfmon
     29       1      368       1612    16     0,00   1408 DefWatch
    214       4     2364       4088    44     0,16   1468 dlpsp
    130      10     1436       3088    30     0,02   2244 dlpwdnt
     57       2      668       2172    18     0,00   1416 dlsdbnt
     96     126     6532       6148   316     0,03   2420 exmgmt
    563      17    21588      19532   129    28,61   2276 explorer
    226      14    13120      20056    81    35,30   4004 FolderShare
    180       5     2832        672    65     0,08   2640 GoogleToolbarNotifier
    106       4     1732       5848    44     0,09   3804 GrooveMonitor
      0       0        0         28     0               0 Idle
    696      19    32596       6396   178     2,91   4724 iexplore
    643      63    17816      17040   118     0,69   1504 inetinfo
     63       3      528       2080    17     0,23   1524 ISRService
     33       2      864       3388    27     0,02   4072 Launcher
    712      26     9504      11588    57     1,52   1868 lsass
     55       3     3588       7404    34     0,27   3852 Matrox.PowerDesk SE
    304       9    24688      24204   156     1,59   3996 Matrox.PowerDesk.PDeskNet
     29       1      284       1500    14     0,02   1572 Matrox.PowerDesk.Services
     29       1      284       1500    14     0,00   1588 Matrox.PowerDesk.Services
     92       3     1048       3508    30     0,03   1604 mdm
    418       8     9484        600    74     0,19    816 MOMHost
    235       5     2996        460    44     0,41    856 MOMHost
    754       9     6148       8376    60     1,06   1684 MOMService
    251     131     4636       8088    47     0,13   2624 mqsvc
     51       2     1900       4040    24     0,39   5536 mscorsvw
    165      16     1752       4384    25     0,05   1148 msdtc
    281      15     9472        820    75     0,28   1076 ntfrs
     86       3     2228       4096    33     0,17   3688 NvMixerTray
     83       3     1132       4304    33     0,05   3640 nvraidservice
    232       6    29212      28372   133     3,33   5884 powershell
    136       5     2504       5080    44     0,09   3964 rapimgr
    329      10    25620      28608   110     0,63    444 Rtvscan
    465      13     2300       4592    73    22,88   1856 services
     18       1      164        504     4     0,06   1352 smss
    385      14    11544      11492   155     4,17   5284 SnagIt32
    287       9     6100       8612    54     0,36   1112 spoolsv
     61       3      716       2400    14     0,00   1632 sqlbrowser
    335       9    37472       1392  1493     0,38   1788 sqlservr
     70       2     1412       4048    20     0,00   1768 sqlwriter
    372      21     1748       4316    24     1,50    344 svchost
   1169      58    21616      29404   191    49,17    400 svchost
    156       7     3900       4684    41     0,03    456 svchost
    175       6     1184       3360    22     0,02    536 svchost
     39       1      300       1284     7     0,00   1332 svchost
     56       2      544       2120    16     0,00   1444 svchost
     85       3     1072       3220    21     0,09   2028 svchost
    158      11     2968       5980    39     0,11   2084 svchost
    122       4     3596       5180    26     0,05   2200 svchost
    155       7     4156       7028    35     0,06   2652 svchost
     77       3     2144       4052    19     0,02   2672 svchost
    163       5     2420       4448    56     0,08   3032 svchost
    221       8     2896       4004    32     0,09   3800 svchost
   2436       0        0        236     2    20,47      4 System
     35       2      656       2916    26     0,02   4620 TscHelp
     91       5    26972      29004    68    18,66   2460 TSVNCache
    106       4     3896       7344    38     0,89   2112 TTTvRc
     64       2     1876       3420    24     0,00   3896 unsecapp
    103       3     2324       4180    36     0,14   2476 VPTray
    251      18    24148      33600   346     5,61   5328 w3wp
    113       5     1352       4868    43     0,19   1860 wcescomm
    594      86     7568       3144    53     1,30   1740 winlogon
    589      22    50268      82920   400    18,36   4484 WINWORD
    115       3     1404       4344    25     0,05   2136 wmiprvse
    196       5     4212       6540    42     2,06   2464 wmiprvse
    200       5     2392       6996    43     1,95   2772 wmiprvse

PS C:\Documents\hs>
```

Bild 1.8 Die Liste der Prozesse ist das Ergebnis nach Ausführung des Commandlets „Get-Process“.

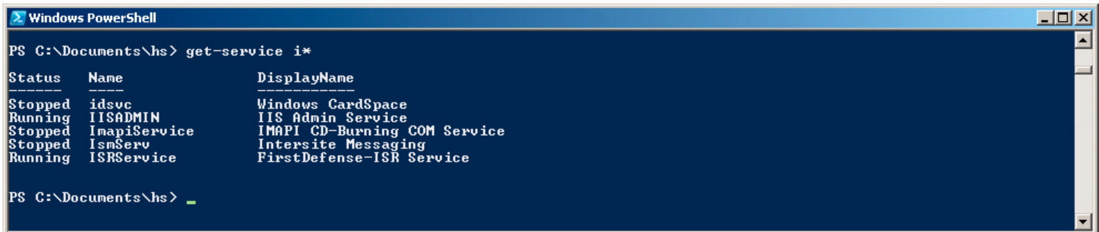

Bild 1.9 Eine gefilterte Liste der Windows-Dienste

Geben Sie „Get-“ ein und drücken Sie dann mehrmals die Tab-Taste. Die PowerShell zeigt nacheinander alle Commandlets an, die mit dem Verb „get“ beginnen. Microsoft bezeichnet diese Funktionalität als „Tabulatorvervollständigung“. Halten Sie bei „Get-Eventlog“ an. Wenn Sie Enter drücken, fordert die PowerShell einen Parameter namens „LogName“ an. Bei „LogName“ handelt es sich um einen erforderlichen Parameter (Pflichtparameter). Nachdem Sie „Application“ eingetippt und die Enter-Taste gedrückt haben, erscheint eine lange Liste der aktuellen Einträge in Ihrem Anwendungsereignisprotokoll.

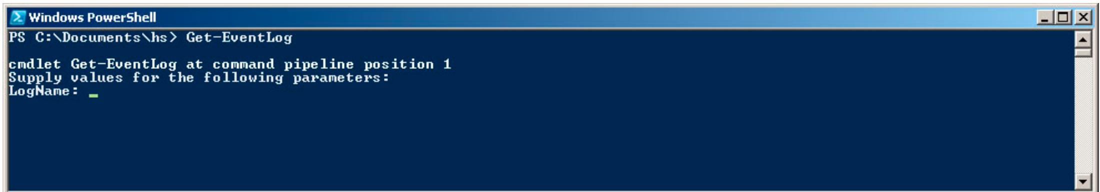

Bild 1.10 PowerShell fragt einen erforderlichen Parameter ab.

Der letzte Test bezieht sich auf die Pipeline-Funktionalität der PowerShell. Auch geht es darum, die Listeneinträge aus dem Windows-Ereignisprotokoll aufzulisten, doch dieses Mal sind nur bestimmte Einträge interessant. Die Aufgabe besteht darin, die letzten zehn Ereignisse abzurufen, die sich auf das Drucken beziehen. Geben Sie den folgenden Befehl ein, der aus drei Commandlets besteht, die über Pipes miteinander verbunden sind:

```
Get-EventLog system | Where-Object { $_.source -eq "print" } | Select-Object -first 10
```

Die PowerShell scheint einige Sekunden zu hängen, nachdem die ersten zehn Einträge ausgegeben wurden. Dieses Verhalten ist korrekt, da das erste Commandlet (`Get-EventLog`) alle Einträge empfängt. Dieses Filtern geschieht durch aufeinanderfolgende Commandlets (`Where-Object` und `Select-Object`). Leider besitzt `Get-EventLog` keinen integrierten Filtermechanismus.

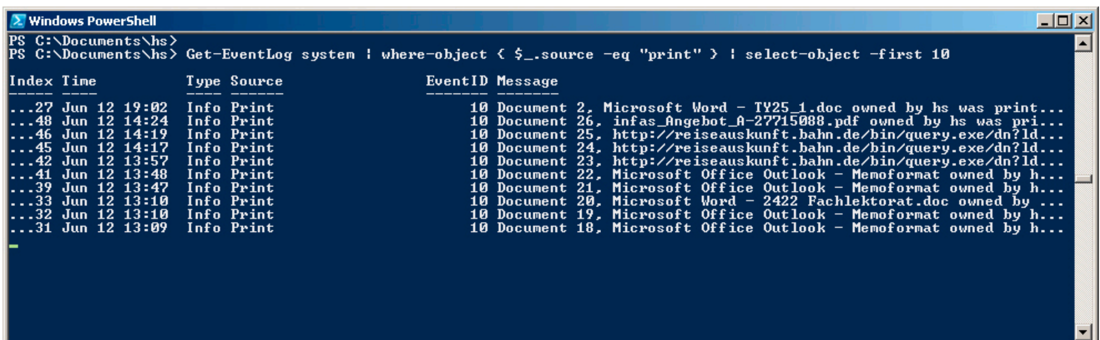

Bild 1.11 Die Einträge des Ereignisprotokolls filtern

1.4.2 Installierte Version ermitteln

Die Windows PowerShell gibt bei ihrem Start ihre Versionsnummer nicht direkt preis. Nur die Jahreszahl im Copyright-Vermerk deutet indirekt auf die Versionsnummer hin. 2015 steht hier für die PowerShell 5.0, 2016 für die PowerShell 5.1. Die PowerShell Core meldet sich ohne Jahreszahl.

Die präzisere Versionsinformation ermittelt man durch den Abruf der eingebauten Variablen `$PSVersionTable`. Neben der PowerShell-Version erhält man auch Informationen über die Frameworks und Protokolle, auf denen die PowerShell aufsetzt.

Die „CLR-Version" steht dabei für die Version der „Common Language Runtime" (CLR), die Laufzeitumgebung des Microsoft .NET Framework. Es fehlt in der Versionstabelle leider die Information, dass die PowerShell 5.1 zwar mit der CLR-Version 4.0 zufrieden ist, aber die .NET-Klassenbibliothek in der Version 4.5.2 oder höher braucht, was eine Installation des .NET Frameworks 4.5.2 oder höher voraussetzt.

PowerShell Core 6.0 erfordert .NET Core 2.0. Allerdings braucht man .NET Core nicht separat installieren: Es wird beim Installationspaket von PowerShell Core mitgeliefert.

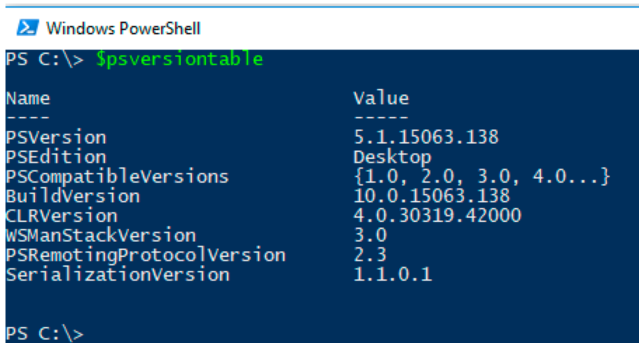

Bild 1.12 Abruf der Versionsinformationen zur PowerShell 5.1 (hier unter Windows 10, Update-Stand 21. 04. 2017)

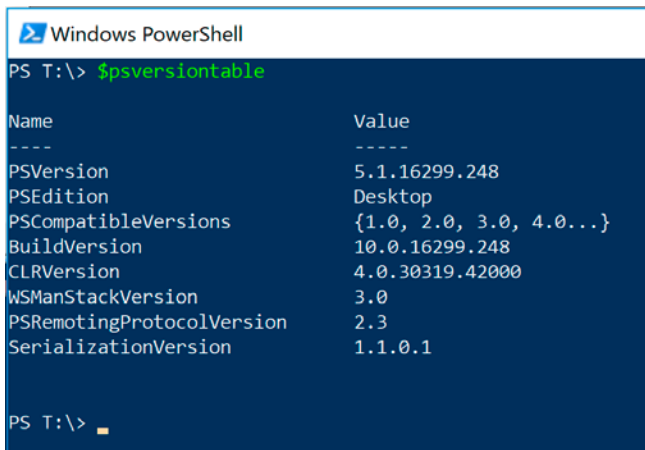

Bild 1.13 Abruf der Versionsinformationen zur PowerShell 5.1 (hier unter Windows 10, Update-Stand 02.03. 2018)

Unter PowerShell Core hat Microsoft einige Anzeigen der Versionstabelle geändert. Am auffälligsten sind der Wert „Core“ statt „Desktop“ bei „PSEdition“ sowie die hinzugefügten Einträge „Platform“ und „OS“ für das aktuelle Betriebssystem. Platform hat die Werte Win32NT, MacOS X und Unix. Die „CLRVersion“ wird hier nicht mehr angezeigt. Microsoft verbirgt hier, welche Version von .NET Core bei PowerShell Core mitgeliefert wird.

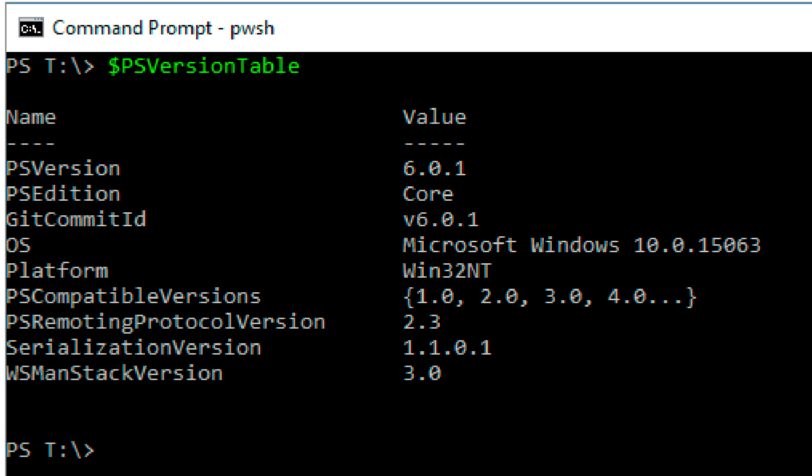

Bild 1.14 Abruf der Versionsinformationen zur PowerShell Core 6.0.1 (hier unter Windows 10, Update-Stand 02.03.2018)

1.4.3 PowerShell im Skriptmodus

Bei einem PowerShell-Skript handelt es sich um eine Textdatei, die Commandlets und/oder Elemente der PowerShell-Skriptsprache (PSL) umfasst. Das zu erstellende Skript legt ein neues Benutzerkonto auf Ihrem lokalen Computer an.

1.4.4 Skript eingeben

Öffnen Sie den Windows-Editor „Notepad“ (oder einen anderen Texteditor) und geben Sie die folgenden Skriptcodezeilen ein, die aus Kommentaren, Variablendeklarationen, COM-Bibliotheksaufrufen und Shell-Ausgabe bestehen:

Listing 1.4 Ein Benutzerkonto erstellen
[1_Basiswissen/ErsteSchritte/LocalUser_Create.ps1]

```
### PowerShell-Script
### Lokales Benutzerkonto anlegen
### (C) Holger Schwichtenberg
```

```
# Eingabewerte
$Name = "Dr. Holger Schwichtenberg"
$Accountname = "HolgerSchwichtenberg"
$Description = "Autor dieses Buchs / Website: www.powershell-doktor.de"
$Password = "secret+123"
$Computer = "localhost"

"Anlegen des Benutzerskonto $Name auf $Computer"

# Zugriff auf Container mit der COM-Bibliothek "Active Directory Service Interface"
$Container = [ADSI] "WinNT://$Computer"

# Benutzer anlegen
$objUser = $Container.Create("user", $Accountname)
$objUser.Put("Fullname", $Name)
$objUser.Put("Description", $Description)
# Kennwort setzen
$objUser.SetPassword($Password)
# Änderungen speichern
$objUser.SetInfo()

"Benutzer angelegt: $Name auf $Computer"
```

Speichern Sie die Textdatei unter dem Namen „createuser.ps1“ in einem Ordner auf der Festplatte, z. B. *x:\temp*. Beachten Sie, dass die Dateinamenserweiterung „.ps1“ lauten muss.

HINWEIS: Ab PowerShell 5.1 gibt es auch einen eleganteren Weg zum Anlegen lokaler Benutzer per Commandlet `New-LocalUser`.

1.4.5 Skript starten

Starten Sie die PowerShell-Konsole. Versuchen Sie dort nun, das Skript zu starten. Geben Sie dazu

```
x:\temp\createuser.ps1
```

ein. Für die Ordner- und Dateinamen können Sie die Tabulatorvervollständigung verwenden! Der Versuch scheitert zunächst wahrscheinlich, da die Skriptausführung auf den meisten Windows-Betriebssystemversionen standardmäßig in der PowerShell nicht zulässig ist. Dies ist kein Fehler, sondern eine Sicherheitsfunktionalität. Denken Sie an den „Love Letter“-Wurm für den Windows Script Host!

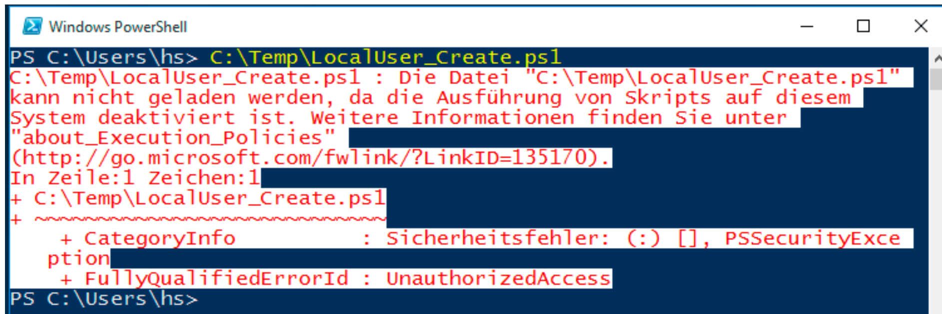

Bild 1.15 Die Skriptausführung ist standardmäßig verboten.

HINWEIS: Bisher war die PowerShell-Skriptausführung auf allen Betriebssystemen im Standard verboten. Erstmals in Windows Server 2012 R2 hat Microsoft sie im Standard erlaubt, sofern das Skript auf der lokalen Festplatte liegt. Entfernte Skripte können nur mit digitaler Signatur gestartet werden. Diese Einstellung nennt sich „RemoteSigned". In anderen Betriebssystemen gibt es aber keine Änderung des Standards, der „Restricted" lautet.

1.4.6 Skriptausführungsrichtlinie ändern

Um ein PowerShell-Skript auf Betriebssystemen wie Windows 7, Windows 8.x und Windows 10, wo dies im Standard nicht erlaubt ist, überhaupt starten zu können, müssen Sie die Skript-Ausführungsrichtlinie ändern. Später in diesem Buch lernen Sie, welche Optionen es dafür gibt. Für den ersten Test wird die Sicherheit ein wenig abgeschwächt, aber wirklich nur ein wenig. Mit dem folgenden Befehl lässt man die Ausführung von Skripten zu, die sich auf dem lokalen System befinden, er verbietet aber Skripten von Netzwerkressourcen (das Internet eingeschlossen) die Ausführung, wenn diese keine digitale Signatur besitzen.

```
Set-ExecutionPolicy RemoteSigned
```

Später in diesem Buch lernen Sie, wie Sie PowerShell-Skripte digital signieren. Außerdem erfahren Sie, wie Sie Ihr System auf Skripte beschränken, die Sie oder Ihre Kollegen signiert haben.

Überprüfen Sie die vorgenommenen Änderungen mit dem Commandlet `Get-ExecutionPolicy`.

Es kann nun sein, dass Sie `Set-ExecutionPolicy` gar nicht ausführen können und eine Fehlermeldung wie die nachstehende sehen, dass die Änderung in der Registrierungsdatenbank mangels Rechten nicht ausgeführt werden konnte.

```
Windows PowerShell
PS C:\Users\hs> Set-ExecutionPolicy RemoteSigned

Ausführungsrichtlinie ändern
Die Ausführungsrichtlinie trägt zum Schutz vor nicht vertrauenswürdigen
Skripts bei. Wenn Sie die Ausführungsrichtlinie ändern, sind Sie
möglicherweise den im Hilfethema "about_Execution_Policies" unter
"http://go.microsoft.com/fwlink/?LinkID=135170" beschriebenen
Sicherheitsrisiken ausgesetzt. Möchten Sie die Ausführungsrichtlinie
ändern?
[J] Ja  [A] Ja, alle  [N] Nein  [K] Nein, keine  [H] Anhalten
[?] Hilfe(Standard ist "N"): j
Set-ExecutionPolicy : Der Zugriff auf den Registrierungsschlüssel "HKEY_
LOCAL_MACHINE\SOFTWARE\Microsoft\PowerShell\1\ShellIds\Microsoft.PowerSh
ell" wurde verweigert. Starten Sie zum Ändern der Ausführungsrichtlinie
für den Standardbereich (LocalMachine) Windows PowerShell mit der
Option "Als Administrator ausführen". Führen Sie zum Ändern der
Ausführungsrichtlinie für den aktuellen Benutzer "Set-ExecutionPolicy
-Scope CurrentUser" aus.
In Zeile:1 Zeichen:1
+ Set-ExecutionPolicy RemoteSigned
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : PermissionDenied: (:) [Set-ExecutionPolic
   y], UnauthorizedAccessException
    + FullyQualifiedErrorId : System.UnauthorizedAccessException,Micros
   oft.PowerShell.Commands.SetExecutionPolicyCommand
PS C:\Users\hs>
```

Bild 1.16 Die Benutzerkontensteuerung verbietet die Änderung der Skriptausführungsrichtlinie.

Dies ist die Benutzerkontensteuerung, die Microsoft seit Windows Vista in Windows mitliefert. Benutzerkontensteuerung (User Account Control, UAC) bedeutet, dass alle Anwendungen seit Windows Vista immer unter normalen Benutzerrechten laufen, auch wenn ein Administrator angemeldet ist. Wenn eine Anwendung höhere Rechte benötigt (z. B. administrative Aktionen, die zu Veränderungen am System führen), fragt Windows explizit in Form eines sogenannten Consent Interface beim Benutzer nach, ob der Anwendung diese Rechte gewährt werden sollen.

HINWEIS: Nur mit Windows Server ab Version 2012 startet der eingebaute Administrator (Konto „Administrator“) alle Skripte, die Konsole und andere .exe-Anwendungen unter vollen Rechten. Alle anderen Administratoren unterliegen der Benutzerkontensteuerung.

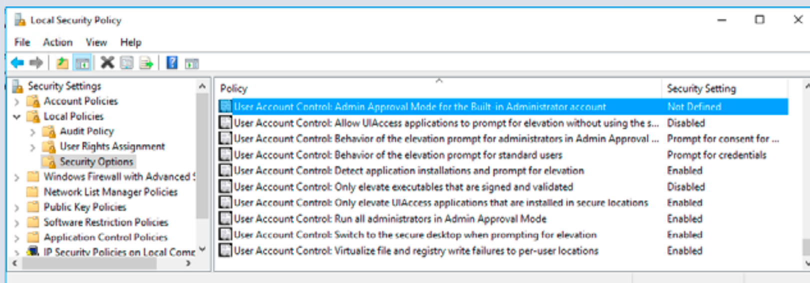

Bild 1.17 Die besondere Einstellung für den eingebauten Administrator in den Sicherheitsrichtlinien von Windows Server

Um die PowerShell mit vollen Rechten zu starten, wählen Sie aus dem Startmenü (oder einer Verknüpfung z. B. in der Taskleiste) die PowerShell mit der rechten Maustaste aus und klicken auf „Als Administrator ausführen".

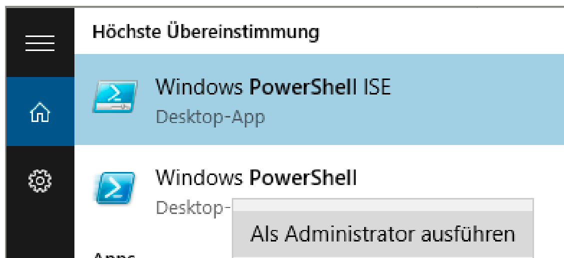

Bild 1.18 PowerShell „Als Administrator ausführen"

Dass die PowerShell als Administrator gestartet ist, sehen Sie an dem Zusatz „Administrator:" in der Fenstertitelzeile der Konsole.

Geben Sie in diesem Fenster erneut ein:

```
Set-ExecutionPolicy RemoteSigned
```

Dies sollte nun funktionieren wie in Bild 1.19 gezeigt.

Starten Sie nun das Skript erneut mit:

```
x:\temp\createuser.ps1
```

Jetzt sollte die Nachricht erscheinen, dass das Benutzerkonto erstellt worden ist.

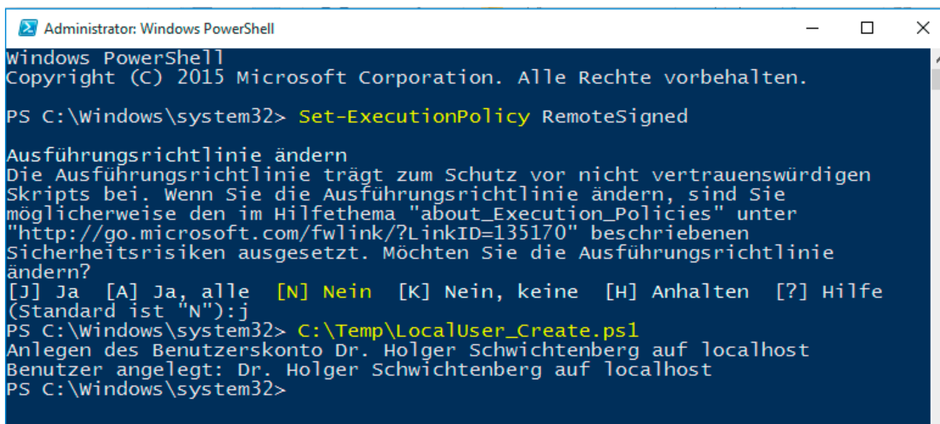

Bild 1.19 Erfolgreiches Ändern der Skriptausführungsrichtlinien und Start des Skripts „LocalUser_Create.ps1"

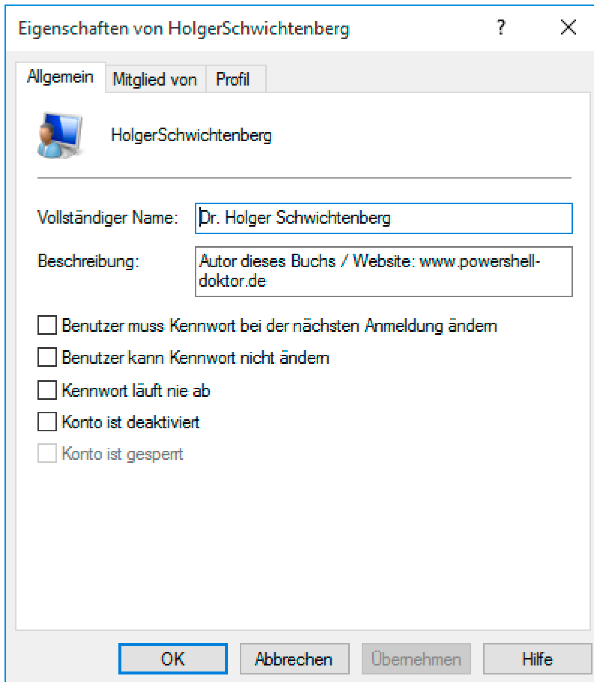

Bild 1.20 Das neu erstellte lokale Benutzerkonto

1.4.7 Farben ändern

Die PowerShell verwendet leider einige Farben mit wenig Kontrast. So werden Zeichenketten in einfachen oder doppelten Anführungszeichen in „DarkCyan“ auf dunkelblauem Grund dargestellt. Falls Sie dies nicht gut lesen können, ändern Sie doch die Farbe auf Cyan:

```
Set-PSReadlineOption -TokenKind String -ForegroundColor Cyan
```

```
PS x:\> "Zeichenkette vorher"
Zeichenkette vorher
PS x:\> (Get-PSReadlineOption).StringForeGroundColor
DarkCyan
PS x:\> Set-PSReadlineOption -TokenKind String -ForegroundColor Cyan
PS x:\> "Zeichenkette nacher"
Zeichenkette nacher
PS x:\> (Get-PSReadlineOption).StringForeGroundColor
Cyan
PS x:\> _
```

Bild 1.21 Auswirkung der Farbänderung

Falls Sie beim Eingeben schon einen Fehler gemacht haben, haben Sie rote Schrift auf blauem Untergrund gesehen. Wenn Sie das nicht gut lesen können, geben Sie bitte ein:

```
(Get-Host).PrivateData.ErrorBackgroundColor = "white"
```

Damit stellen Sie um auf rote Schrift auf weißem Grund für Fehlerausgaben.

So einen Befehl legt man in der PowerShell-Profilskriptdatei ab, damit er immer beim Start der PowerShell automatisch ausgeführt wird, siehe Kapitel 16 *Standardeinstellungen ändern mit Profilskripten*.

1.5 Woher kommen die Commandlets?

Die Windows PowerShell umfasste in der Version 1.0 nur 129 Commandlets (und Funktionen). In PowerShell 2.0 waren es 236, in PowerShell 3.0 waren es 322 und in PowerShell 4.0 sind es auch immer noch „nur“ 328 und in PowerShell 5.0 unter Windows 10 sind es 340, in Windows 10 Creators Update (Redstone 2, Version 1703 vom April 2017) mit PowerShell 5.1 sind es 370. Als Kern der PowerShell werden hier alle Commandlets und Funktionen bezeichnet, die sich in einem der PowerShell-Module befinden, die mit Windows ausgeliefert werden bzw. mit dem WMF-Add-on installiert werden und die auf allen unterstützten Betriebssystemen verfügbar sind (und daher das Wort „PowerShell“ im Modulnamen tragen und in der Dokumentation „Core Modules“ genannt werden).

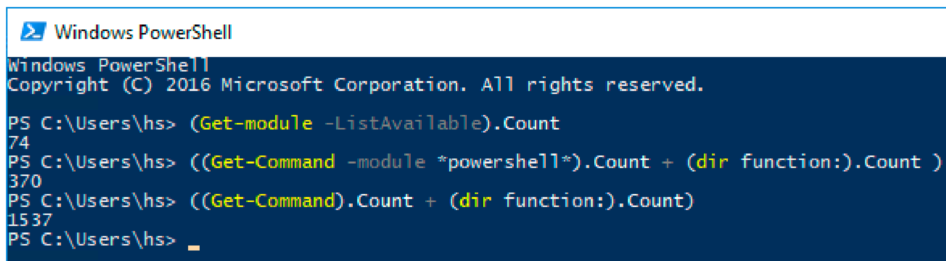

Bild 1.22 Zählen der Commandlets und Funktionen unter Windows 10 (Stand Creators Update, Versionsnummer 1703, Redstone 2)

Es gibt noch viel mehr Commandlets als die oben genannten, diese gehören aber nicht zur Windows PowerShell im engeren Sinne, sondern zu optionalen Erweiterungen oder der jeweiligen Windows-Betriebssystemversion.

Schon kurz nach Version 1.0 der Windows PowerShell gab es erste Erweiterungen wie zum Beispiel die PowerShell Community Extensions (siehe Abschnitt 1.6).

Mit Windows 7 bzw. Windows Server 2008 R2 hat Microsoft begonnen, Zusatzmodule direkt mit dem Betriebssystem auszuliefern. Diese Zusatzmodule bringen in Windows 8.1 die Anzahl der Commandlets auf über 1000. In Windows 10 (Stand Creators Update, Versionsnummer 1703) sind es dann 1537.

ACHTUNG: Anders als die Erweiterungsmodule, die es oft für mehrere (auch ältere) PowerShell-Versionen gibt, kann man die zum Betriebssystem gehörenden Module nicht in einem älteren Betriebssystem verwenden. In dem zum Redaktionsschluss dieses Buchs aktuellen Stand der PowerShell 6.0.1 kann man viele, aber noch nicht alle zum Windows-Betriebssystem gehörenden PowerShell-Module auch in PowerShell Core unter Windows verwenden.

1.6 PowerShell Community Extensions (PSCX) herunterladen und installieren

Bei den „PowerShell Community Extensions" (kurz PSCX) handelt es sich um ein Open-Source-Projekt (ursprünglich auf Codeplex.com, mittlerweile auf Github.com, siehe *https://github.com/Pscx/Pscx*), das zusätzliche Funktionalität mit Commandlets für die Windows PowerShell realisiert, wie zum Beispiel `Get-DHCPServer`, `Get-DomainController`, `Get-MountPoint`, `Get-TerminalSession`, `Set-VolumneLabel`, `Write-Tar` und viele weitere. Das Projekt steht unter Führung von Microsoft, aber jeder .NET-Softwareentwickler ist eingeladen, daran mitzuwirken. In regelmäßigen Abständen werden neue Versionen veröffentlicht. Die aktuelle Version zum Reaktionsschluss dieses Buchs ist die Version 3.3.2.

TIPP: In diesem Buch werden an einigen Stellen Commandlets aus den PSCX verwendet. Daher sollten Sie die PSCX installieren.

Die Installation der PSCX führt man heutzutage über das Commandlet `Install-Module` aus. Dieses Commandlet lädt das Modul aus der PowerShell Gallery (*https://www.powershellgallery.com*), einem von Microsoft betriebenen Online-Portal mit PowerShell-Erweiterungen, und installiert das Modul. Alternativ dazu können Sie auf Github ein ZIP-Paket laden (*https://github.com/Pscx/Pscx/releases*) und die Installation manuell vornehmen.

Für die automatische Installation führen Sie in einer PowerShell-Konsole, die administrative Rechte besitzt, bitte aus:

```
Install-Module Pscx -Scope CurrentUser -AllowClobber
```

Die nächste Bildschirmabbildung erklärt die Bedeutung des Parameters `AllowClobber`: In den PSCX gibt es einige Befehle, die mittlerweile in die PowerShell fest eingebaut sind.

```
PowerShell
PS T:\> Install-Module PSCX -Scope CurrentUser

Untrusted repository
You are installing the modules from an untrusted repository. If you trust this repository, change its
InstallationPolicy value by running the Set-PSRepository cmdlet. Are you sure you want to install the modules from
'PSGallery'?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"): y
PackageManagement\Install-Package : The following commands are already available on this system:'gcb,Expand-Archive,For
mat-Hex,Get-Hash,help,prompt,Dismount-VHD,Get-ADObject,Get-Clipboard,Get-Help,Mount-VHD,Set-Clipboard'. This module
'Pscx' may override the existing commands. If you still want to install this module 'Pscx', use -AllowClobber
parameter.
At C:\Program Files\WindowsPowerShell\Modules\PowerShellGet\1.0.0.1\PSModule.psm1:1809 char:21
+ ...          $null = PackageManagement\Install-Package @PSBoundParameters
+
    + CategoryInfo          : InvalidOperation: (Microsoft.Power....InstallPackage:InstallPackage) [Install-Package],
   Exception
    + FullyQualifiedErrorId : CommandAlreadyAvailable,Validate-ModuleCommandAlreadyAvailable,Microsoft.PowerShell.Pack
   ageManagement.Cmdlets.InstallPackage

PS T:\> Install-Module PSCX -Scope CurrentUser -AllowClobber

Untrusted repository
You are installing the modules from an untrusted repository. If you trust this repository, change its
InstallationPolicy value by running the Set-PSRepository cmdlet. Are you sure you want to install the modules from
'PSGallery'?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"): y
PS T:\> _
```

Bild 1.23 Installation der PSCX

Geben Sie nun `Get-DomainController` ein (wenn Ihr Computer Mitglied eines Active Directory ist) oder testen Sie die PSCX mit dem Befehl `Ping-Host`, der auf jedem Computer im Netzwerk funktioniert. Wie Sie in der Bildschirmabbildung an der Ausgabe zu `Ping-Host` lesen können: Es ist ein Commandlet, für das es mittlerweile in der PowerShell einen Ersatz (hier: `Test-Connection`) gibt.

```
Windows PowerShell
Windows PowerShell
Copyright (C) 2012 Microsoft Corporation. All rights reserved.

PS C:\Users\hs.ITV> Get-DomainController

SiteName             CurrentTime          Name
--------             -----------          ----
Default-First-Site   20.02.2013 21:39:24  E02.IT-Visions.local


PS C:\Users\hs.ITV> ping-host www.IT-Visions.de
WARNING: The PSCX\Ping-Host cmdlet is obsolete and will be removed in the next version of PSCX. Use the built-in
Microsoft.PowerShell.Management\Test-Connection cmdlet instead.
Pinging www.it-visions.de [195.234.228.210] with 32 bytes of data:
    Reply from 195.234.228.210 bytes=32 time=22ms TTL=117
    Reply from 195.234.228.210 bytes=32 time=21ms TTL=117
    Reply from 195.234.228.210 bytes=32 time=22ms TTL=117
    Reply from 195.234.228.210 bytes=32 time=22ms TTL=117

Ping statistics for www.it-visions.de:
    Packets: Sent = 4 Received = 4 (0% loss)
    Approximate round trip time: min = 21ms, max = 22ms, avg = 21ms


PS C:\Users\hs.ITV> _
```

Bild 1.24 PSCX-Befehle Get-DomainController und Ping-Host testen

1.7 Den Windows PowerShell-Editor „ISE“ verwenden

Integrated Scripting Environment (ISE) ist der Name des Skripteditors, den Microsoft seit der Windows PowerShell 2.0 mitliefert und der in Windows PowerShell 3.0 nochmals erheblich verbessert wurde. Die ISE startet man mit dem Symbol „PowerShell ISE“ oder indem man in der PowerShell den Befehl „ise“ ausführt.

Die ISE verfügt über zwei Fenster: ein Skriptfenster (im Standard oben, alternativ über das „View“-Menü einstellbar rechts) und ein interaktives Befehlseingabefenster (unten bzw. links). Optional kann man ein drittes Fenster einblenden, das „Command Add-On“, in dem man Befehle suchen kann und eine Eingabehilfe für Befehlsparameter erhält.

Geben Sie unten im interaktiven Befehlseingabefenster in der ISE ein:

```
Get-Process
```

Nachdem Sie mindestens einen Buchstaben eingegeben haben, können Sie die Eingabe mit der Tabulatortaste vervollständigen. Alternativ können Sie **STRG+LEERTASTE** drücken für eine Eingabehilfe mit Auswahlfenster (IntelliSense). Die Ausgaben des interaktiven Bereichs erscheinen dann direkt unter den Befehlen, wie bei der PowerShell-Konsole. Einen dedizierten Ausgabebereich wie in der ISE in PowerShell 2.0 gibt es nicht mehr.

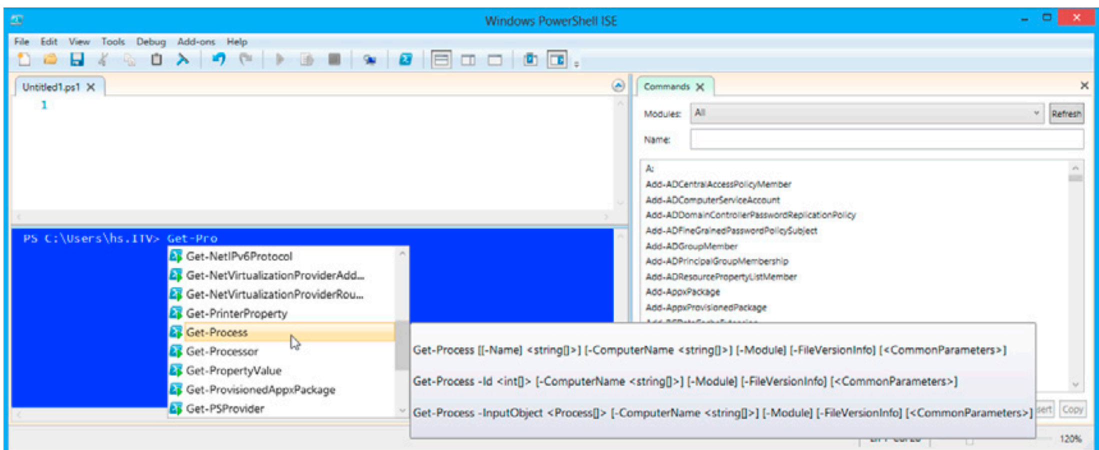

Bild 1.25 IntelliSense-Eingabehilfe

Um die ISE im Skriptmodus zu verwenden, erstellen Sie eine neue Skriptdatei (Menü „File/New“) oder öffnen Sie eine vorhandene *.ps1*-Datei (Menü „File/Open“). Öffnen Sie als Beispiel die Skriptdatei *CreateUser.ps1*, die Sie zuvor erstellt haben. Es sind Zeilennummern zu sehen. Die verschiedenen Bestandteile des Skripts sind in unterschiedlichen Farben dargestellt. Auch hier funktioniert die Eingabeunterstützung mit der Tabulatortaste und IntelliSense.

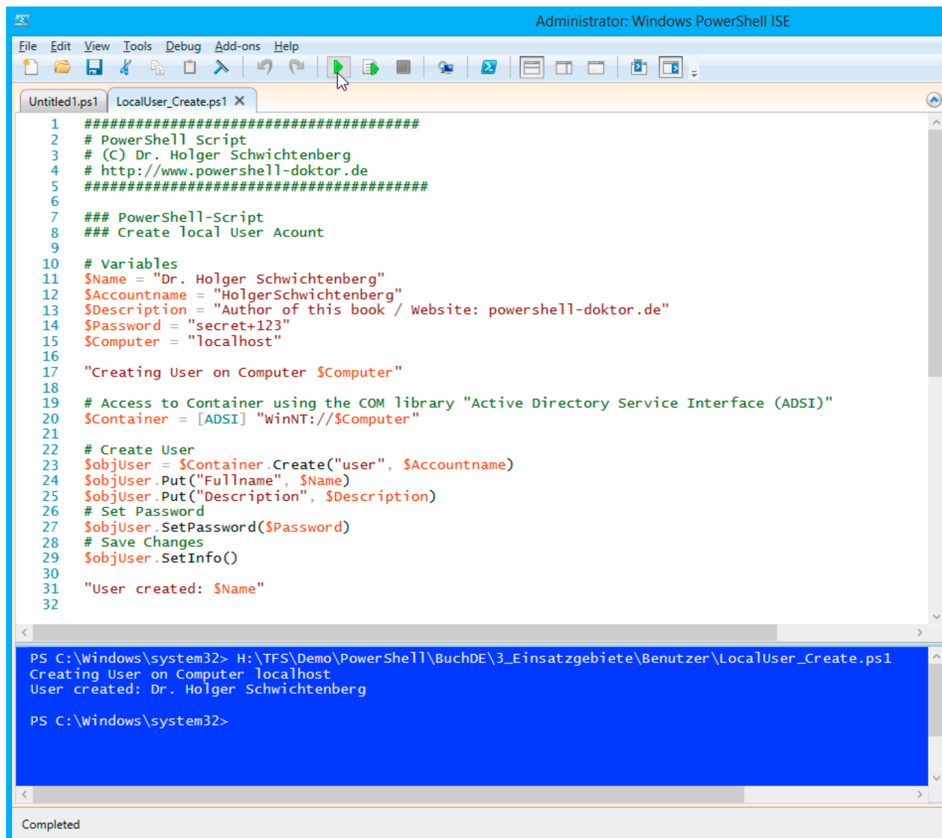

Bild 1.26 Die ISE im Skriptmodus

Um das Skript auszuführen, klicken Sie auf das Start-Symbol in der Symbolleiste oder drücken Sie **F5**. Auch hier wird das Ergebnis im interaktiven Bereich angezeigt.

TIPP: Stellen Sie sicher, dass Sie die ISE als Administrator ausführen und dass das Benutzerkonto noch nicht existiert, bevor Sie das Skript ausführen.

Ein interessantes Feature ist das Debugging, mit dem Sie ein Skript Zeile für Zeile durchlaufen und währenddessen den Zustand der Variablen betrachten können.

Setzen Sie dazu den Cursor auf eine beliebige Zeile in Ihrem Skript und tippen Sie dann auf **F9** (oder wählen Sie „Toogle Breakpoint“ im Kontextmenü oder im Menü „Debug“). Daraufhin erscheint die Zeile in Rot – ein sogenannter „Haltepunkt“.

Starten Sie das Skript nun mit **F5**. Die ISE stoppt in der Zeile mit dem Haltepunkt und diese wird orange. Mit der Taste **F10** springen Sie zum nächsten Befehl. Diese wird dann gelb und die Zeile mit dem Haltepunkt wird wieder rot.

HINWEIS: Die gelbe Zeile ist immer die nächste Zeile, die ausgeführt wird.

Administrator: Windows PowerShell ISE

File Edit View Tools Debug Add-ons Help

Untitled1.ps1 [Read Only] LocalUser_Create.ps1 [Read Only]

```
10  # Variables
11  $Name = "Dr. Holger Schwichtenberg"
12  $Accountname = "HolgerSchwichtenberg"
13  $Description = "Author of this book / Website: powershell-doktor.de"
14  $Password = "secret+123"
15  $Computer = "localhost"
16
17  "Creating User on Computer $Computer"
18
19  # Access to Container using the COM library "Active Directory Service Interface (ADSI)"
20  $Container = [ADSI] "WinNT://$Computer"
21
22  # Create User
23  $objUser = $Container.Create("user", $Accountname)
24  $objUser.Put("Fullname", $Name)
25  $objUser.Put("Description", $Description)
26  # Set Password
27  $objUser.SetPassword($Password)
28  # Save Changes
29  $objUser.SetInfo()
30
31  "User created: $Name"
32
```

Zeile mit Haltepunkt (rot)

Nächste auszuführende Zeile (gelb)

```
[DBG]: PS C:\Windows\system32>> $Computer
localhost

[DBG]: PS C:\Windows\system32>> $Container


distinguishedName :
Path              : WinNT://localhost



[DBG]: PS C:\Windows\system32>> |
```

Bild 1.27 Skript-Debugging mit der ISE

Im interaktiven Bereich können Sie im Haltemodus den aktuellen Zustand der Variablen abfragen, indem Sie dort z. B. eingeben

```
$Computer
```

oder

```
$Container
```

Man kann auch Werte interaktiv ändern. Um das Skript fortzusetzen, drücken Sie wieder **F5**. Über das Menü „Debug" sind weitere Steuerbefehle möglich.

HINWEIS: Sie müssen den Debugger vorher beenden (Menüpunkt „Debug/Stop Debugger"), wenn Sie das Skript erneut ändern möchten.

2 Fakten zur PowerShell

2.1 Geschichte der PowerShell

In der Vergangenheit war Active Scripting manchen Administratoren zu komplex, weil es viel Wissen über objektorientiertes Programmieren und das Component Object Model (COM) voraussetzt. Die vielen Ausnahmen und Ungereimtheiten im Active Scripting erschwerten das Erlernen von Windows Script Host (WSH) und der zugehörigen Komponentenbibliotheken.

Schon im Zuge der Entwicklung des Windows Server 2003 gab Microsoft zu, dass man Unix-Administratoren zum Interview über ihr tägliches Handwerkszeug gebeten hatte. Das kurzfristige Ergebnis war eine große Menge zusätzlicher Kommandozeilenwerkzeuge. Langfristig setzt Microsoft jedoch auf eine Ablösung des DOS-ähnlichen Konsolenfensters durch eine neue Skripting-Umgebung.

Mit dem Erscheinen des .NET Frameworks im Jahre 2002 wurde lange über einen WSH.NET spekuliert. Microsoft stellte jedoch die Neuentwicklung des WSH für das .NET Framework ein, als abzusehen war, dass die Verwendung von .NET-basierten Programmiersprachen wie C# und Visual Basic .NET dem Administrator nur noch mehr Kenntnisse über objektorientierte Softwareentwicklung abverlangen würde.

Microsoft beobachtete in der Unix-Welt eine hohe Zufriedenheit mit den dortigen Kommandozeilen-Shells und entschloss sich daher, das Konzept der Unix-Shells, insbesondere das Pipelining, mit dem .NET Framework zusammenzubringen und daraus eine .NET-basierte Windows Shell zu entwickeln. Diese sollte noch einfacher als eine Unix-Shell, aber dennoch so mächtig wie das .NET Framework sein.

In einer ersten Beta-Version wurde die neue Shell schon unter dem Codenamen „Monad“ auf der Professional Developer Conference (PDC) im Oktober 2003 in Los Angeles vorgestellt. Nach den Zwischenstufen „Microsoft Shell (MSH)“ und „Microsoft Command Shell“ trägt die neue Skriptumgebung seit Mai 2006 den Namen „Windows PowerShell“.

Die PowerShell 1.0 erschien am 6.11.2006 zeitgleich mit Windows Vista, war aber dort nicht enthalten, sondern musste heruntergeladen und nachinstalliert werden.

Die PowerShell 2.0 ist zusammen mit Windows 7/Windows Server 2008 R2 erschienen am 22.7.2009.

Die PowerShell 3.0 ist zusammen mit Windows 8/Windows Server 2012 erschienen am 15.8.2012.

Die PowerShell 4.0 ist zusammen mit Windows 8.1/Windows Server 2012 R2 am 9.9.2013 erschienen.

Die PowerShell 5.0 ist als Teil von Windows 10 erschienen am 29.7.2015. Abweichend von den bisherigen Gepflogenheiten ist die PowerShell 5.0 als Erweiterung für Windows Server 2008 R2 (mit Service Pack 1) und Windows Server 2012/2012 R2 erst deutlich später am 16.12.2015 erschienen. Für Windows 7 und Windows 8.1 sollte es erst gar keine Version mehr geben. Doch am 18.12.2015 hatte Microsoft ein Einsehen mit den Kunden und lieferte die PowerShell 5.0 auch für diese Betriebssysteme nach. Kurioserweise musste Microsoft den Download dann am 23.12.2015 wegen eines gravierenden Fehlers für einige Wochen vom Netz nehmen. Microsoft hatte das Produkt im neuen Agilitätswahn nicht richtig getestet.

Windows Server 2016 (erschienen am 26.9.2016) enthält PowerShell 5.1 und Windows 10 und wurde mit dem Windows 10 Anniversary Update (Version 1607, Codename „Redstone 1“) am 2.8.2016 aktualisiert. PowerShell 5.1 ist erst seit 19.1.2017 als Add-on für Windows 7, Windows 8.1, Windows Server 2008 R2, Windows 2012 und Windows 2012 R2 verfügbar.

Eine reduzierte „Core“-Version der Windows PowerShell ist als „Windows PowerShell Core 5.1“ enthalten in Windows Nano Server, im ersten Release 2016 als Standardpaket, im zweiten in Release „1709“ als Option.

Die erste Version der plattformneutralen PowerShell Core (ohne Windows im Namen!) ist mit der Versionsnummer 6.0 am 20.01.2018 erschienen.

HINWEIS: Mit Windows 10 hat Microsoft das Auslieferungsverfahren auf „Windows as a Service“ umgestellt. Dies bedeutet, dass Microsoft über Windows Update im Sinne der neuen „agilen“ Strategie nun auch ständig neue Funktionen ausliefert. Dies betrifft ebenso die Windows PowerShell, die dann zukünftig auch auf diesem Wege häufigere Aktualisierungen erfahren kann. Wie häufig dies sein wird, ist zum Redaktionsschluss dieses Buchs noch offen.

Microsoft hat sich seit dem Jahr 2015 für andere Betriebssysteme und die Entwicklung als „Open Source Software“ (OSS) geöffnet. Dies betrifft nun auch die PowerShell: Die PowerShell Core, die am 20.1.2018 als Version 6.0 (in Nachfolge von Windows PowerShell 5.1) erschienen ist, ist OpenSource und läuft nicht nur auf Windows, sondern auch auf macOS und Linux.

2.2 Warum PowerShell einsetzen?

Falls Sie eine Motivation brauchen, sich mit der PowerShell zu beschäftigen, wird dieses Kapitel Sie Ihnen liefern. Es stellt die Lösung für eine typische Scripting-Aufgabe sowohl im „alten“ Windows Script Host (WSH) als auch in der „neuen“ PowerShell vor und Sie werden schnell erkennen, welche Vorteile Ihnen die PowerShell bietet.

Zur Motivation, sich mit der PowerShell zu beschäftigen, soll folgendes Beispiel aus der Praxis dienen. Es soll ein Inventarisierungsskript für Software erstellt werden, das die installierten MSI-Pakete mit Hilfe der Windows Management Instrumentation (WMI) von mehreren Computern ausliest und die Ergebnisse in einer CSV-Datei *(softwareinventar.csv)* zusammenfasst. Die Namen (oder IP-Adressen) der abzufragenden Computer sollen in einer Textdatei *(computernamen.txt)* stehen.

Die Lösung mit dem WSH benötigt 90 Codezeilen (inklusive Kommentare und Parametrisierungen). In der PowerShell lässt sich das Gleiche in nur 13 Zeilen ausdrücken. Wenn man auf die Kommentare und die Parametrisierung verzichtet, dann reicht sogar genau eine Zeile. Das PowerShell-Skript läuft in der Windows PowerShell und auch in der PowerShell Core unter Windows, aber nicht unter Linux und macOS, da es dort noch keine Implementierung des für den Zugriff auf die installierte Software notwendigen Web Based Enterprise Management (WBEM) und des Common Information Model (CIM) für die PowerShell gibt.

Listing 2.1 Softwareinventarisierung – Lösung 1 mit dem WSH
[3_Einsatzgebiete/Software/Software_Inventory.vbs]

```
' ------------------------------------------
' Skriptname: Software_inventar.vbs
' Autor: Dr. Holger Schwichtenberg
' ------------------------------------------
' Dieses Skript erstellt eine Liste
' der installierten Software
' ------------------------------------------
Option Explicit

' --- Einstellungen
Const Trennzeichen = ";" ' Trennzeichen für Spalten in der Ausgabedatei
Const Eingabedateiname = "computernamen.txt"
Const Ausgabedateiname = "softwareinventar.csv"
Const Bedingung = "SELECT * FROM Win32_Product where not Vendor like '%Microsoft%'"

Dim objFSO ' Dateisystem-Objekt
Dim objTX  ' Textdatei-Objekt für die Liste der zu durchsuchenden computer
Dim i  ' Zähler für Computer
Dim computer ' Name des aktuellen computers
Dim Eingabedatei' Name und Pfad der Eingabedatei
Dim Ausgabedatei' Name und Pfad der Ausgabedatei

' --- Startmeldung
WScript.Echo "Softwareinventar.vbs"
WScript.Echo "(C) Dr. Holger Schwichtenberg, http://www.Windows-Scripting.de"

' --- Global benötigtes Objekt
Set objFSO = CreateObject("Scripting.FileSystemObject")

' --- Ermittlung der Pfade
Eingabedatei = GetCurrentPfad & "\" & Eingabedateiname
Ausgabedatei = GetCurrentPfad & "\" & Ausgabedateiname

' --- Auslesen der computerliste
Set objTX = objFSO.OpenTextFile(Eingabedatei)

' --- Meldungen
```

```
WScript.Echo "Eingabedatei: " & Eingabedatei
WScript.Echo "Ausgabedatei: " & Ausgabedatei

' --- Überschriften einfügen
Ausgabe _
"computer" & Trennzeichen & _
"Name" & Trennzeichen & _
    "Beschreibung" & Trennzeichen & _
    "Identifikationsnummer" & Trennzeichen & _
    "Installationsdatum" & Trennzeichen & _
    "Installationsverzeichnis" & Trennzeichen & _
    "Zustand der Installation" & Trennzeichen & _
    "Paketzwischenspeicher" & Trennzeichen & _
    "SKU Nummer" & Trennzeichen & _
    "Hersteller" & Trennzeichen & _
    "Version"

' --- Schleife über alle Computer
Do While Not objTX.AtEndOfStream
    computer = objTX.ReadLine
    i = i + 1
    WScript.Echo "=== Computer #" & i & ": " & computer

GetInventar computer

Loop

' --- Eingabedatei schließen
objTX.Close
' --- Abschlußmeldung
WScript.echo "Softwareinventarisierung beendet!"

' === Softwareliste für einen computer erstellen
Sub GetInventar(computer)

Dim objProduktMenge
Dim objProdukt
Dim objWMIDienst

' --- Zugriff auf WMI
Set objWMIDienst = GetObject("winmgmts:" &_
    "{impersonationLevel=impersonate}!\\" & computer &_
    "\root\cimv2")
' --- Liste anfordern
Set objProduktMenge = objWMIDienst.ExecQuery _
    (Bedingung)
' --- Liste ausgeben
WScript.Echo "Auf " & computer & " sind " & _
objProduktMenge.Count & " Produkte installiert."
For Each objProdukt In objProduktMenge
    Ausgabe _
    computer & Trennzeichen & _
    objProdukt.Name & Trennzeichen & _
    objProdukt.Description & Trennzeichen & _
    objProdukt.IdentifyingNumber & Trennzeichen & _
    objProdukt.InstallDate & Trennzeichen & _
    objProdukt.InstallLocation & Trennzeichen & _
    objProdukt.InstallState & Trennzeichen & _
```

```
    objProdukt.PackageCache & Trennzeichen & _
    objProdukt.SKUNumber & Trennzeichen & _
    objProdukt.Vendor & Trennzeichen & _
    objProdukt.Version
WScript.Echo     objProdukt.Name
Next
End Sub

' === Ausgabe
Sub Ausgabe(s)
Dim objTextFile
' Ausgabedatei öffnen
Set objTextFile = objFSO.OpenTextFile(Ausgabedatei, 8, True)
objTextFile.WriteLine s
objTextFile.Close
'WScript.Echo s
End Sub

' === Pfad ermitteln, in dem das Skript liegt
Function GetCurrentPfad
GetCurrentPfad = objFSO.GetFile (WScript.ScriptFullName).ParentFolder
End Function
```

Listing 2.2 Softwareinventarisierung – Lösung 2 als PowerShell-Skript
[3_Einsatzgebiete/Software/SoftwareInventory_WMI_Script.ps1]

```
# Einstellungen
$InputFileName = "computernamen.txt"
$OutputFileName = "softwareinventar.csv"
$Query = "SELECT * FROM Win32_Product where not Vendor like '%Microsoft%'"

# Eingabedatei auslesen
$Computers = Get-Content $InputFileName

# Schleife über alle Computer
$Software = $Computers | ForEach { Get-CimInstance -query $Query -computername $_ }
# Ausgabe in CSV
$Software | select Name, Description, IdentifyingNumber, InstallDate,
InstallLocation, InstallState, SKUNumber, Vendor, Version | export-csv
$OutputFileName -notypeinformation
```

Listing 2.3 Softwareinventarisierung – Lösung 3 als PowerShell-Pipeline-Befehl
[3_Einsatzgebiete/Software/SoftwareInventory_WMI_Pipeline.ps1]

```
Get-Content "computers.txt" | ForEach {Get-CimInstance -computername $_ -query
"SELECT * FROM Win32_Product where not Vendor like '%Microsoft%'" } | export-csv
"Softwareinventory.csv" -notypeinformation
```