

Das
Standard-
werk

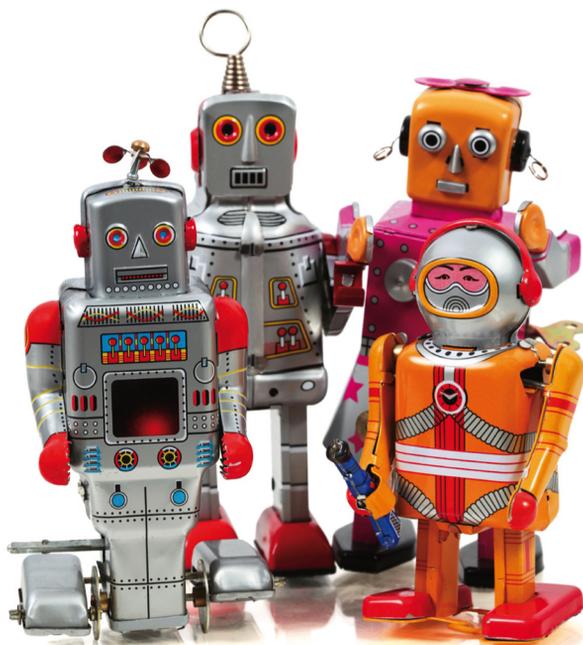
Für
PowerShell
2-5

PowerShell 5

Windows-Automation für
Einsteiger und Profis



Dr. Tobias Weltner



O'REILLY®

PowerShell 5 – Windows-Automation für Einsteiger und Profis

Dr. Tobias Weltner

PowerShell 5 – Windows-Automation für Einsteiger und Profis

O'REILLY®

Dr. Tobias Weltner

Lektorat: Ariane Hesse

Korrektorat: Sibylle Feldmann

Satz: mediaService, www.mediaservice.tv

Herstellung: Susanne Bröckelmann

Umschlaggestaltung: Michael Oreal, www.oreal.de, unter Verwendung eines Fotos von
Valerie Loiseleux / iStock. by Getty Images

Druck und Bindung: Druckerei C.H. Beck, www.becksche.de

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.

ISBN:

Print 978-3-96009-009-0

PDF 978-3-96010-033-1

ePub 978-3-96010-034-8

mobi 978-3-96010-035-5

1. Auflage 2016 Print 978-3-96009-009-0

1. korrigierter Nachdruck 2017

2. korrigierter Nachdruck 2018

Dieses Buch erscheint in Kooperation mit O'Reilly Media, Inc. unter dem Imprint »O'REILLY«. O'REILLY ist ein Markenzeichen und eine eingetragene Marke von O'Reilly Media, Inc. und wird mit Einwilligung des Eigentümers verwendet.

Copyright © 2016 dpunkt.verlag GmbH

Wieblinger Weg 17

69123 Heidelberg

Die vorliegende Publikation ist urheberrechtlich geschützt. Alle Rechte vorbehalten. Die Verwendung der Texte und Abbildungen, auch auszugsweise, ist ohne die schriftliche Zustimmung des Verlags urheberrechtswidrig und daher strafbar. Dies gilt insbesondere für die Vervielfältigung, Übersetzung oder die Verwendung in elektronischen Systemen.

Es wird darauf hingewiesen, dass die im Buch verwendeten Soft- und Hardware-Bezeichnungen sowie Markennamen und Produktbezeichnungen der jeweiligen Firmen im Allgemeinen warenzeichen-, marken- oder patentrechtlichem Schutz unterliegen.

Die Informationen in diesem Buch wurden mit größter Sorgfalt erarbeitet. Dennoch können Fehler nicht vollständig ausgeschlossen werden. Verlag, Autoren und Übersetzer übernehmen keine juristische Verantwortung oder irgendeine Haftung für eventuell verbliebene Fehler und deren Folgen.

Inhalt

Einführung	23
Wer braucht eigentlich PowerShell?	23
»Windows PowerShell« 5 vs. »PowerShell« 6	23
»Windows PowerShell«	24
... und »PowerShell«	24
Warum PowerShell lernen?	24
Moderne Lernkurve	25
Computer – ich/ich – Computer: PowerShell als Fremdsprache	26
PowerShell als Ersatz für VBScript und Batch	26
Grenzenloses PowerShell	26
Strategische Plattform und Orchestrierung	27
Anwendungsentwicklung	27
Persönliche Entwicklung	27
Wie Sie dieses Buch nutzen	28
Noch mehr Unterstützung	28
1 PowerShell kennenlernen	29
Die PowerShell-Konsole einrichten	32
PowerShell-Konsole starten	33
PowerShell-Version kontrollieren	34
Symbol an Taskleiste heften	35
Bessere Schrift für die Konsole	35
Neue Konsolenfunktionen bei Windows 10 aktivieren	38
Sprungliste: Administratorrechte und ISE	39
32-Bit- und 64-Bit-Versionen	40
PowerShell ISE einsetzen	40
Erste Schritte mit PowerShell	44
Wichtige Vorsichtsmaßnahmen	44
Befehle eingeben	44
Ergebnisse empfangen	45
Farbcodierungen verstehen	51
Rechnen mit PowerShell	52
Umwandlungen	54
Zahlenreihen	55
Unvollständige und mehrzeilige Eingaben	55
Skriptausführung erlauben	56
Tippfehler vermeiden und Eingaben erleichtern	57
Autovervollständigung	57
Pfadnamen vervollständigen	58
Befehlszeilen erneut verwenden	59
Befehlsnamen autovervollständigen	59
Parameter-Autovervollständigung	59
Argument-Autovervollständigung	60

Inhaltsverzeichnis

PowerShell-Hilfe aus dem Internet nachladen	60
Hilfe ohne Internetzugang installieren	62
Hilfe bei PowerShell 3	62
Klassische Konsole oder moderner ISE-Editor?	63
Einschränkungen des ISE-Editors	64
Einschränkungen der klassischen Konsole	65
PowerShell-2.0-Testumgebung	66
Testen Sie Ihr Wissen!	66
Teil A Interaktive Befehlskonsole	69
2 Cmdlets – die PowerShell-Befehle	71
Alles, was Sie über Cmdlets wissen müssen	73
Cmdlets für eine Aufgabe finden	74
Suche nach Tätigkeit oder Tätigkeitsbereich	74
Mit ISE nach Cmdlets suchen	81
Mit der Hilfe nach Cmdlets suchen	86
Mit Parametern Wünsche formulieren	89
Parameter wecken das volle Potenzial der Cmdlets	89
Drei universelle Parametertypen	96
Common Parameter – allgemeine Parameter für alle Cmdlets	101
Fehlermeldungen unterdrücken	102
Unvollständige Parameternamen	103
Parameter-Binding sichtbar machen	106
Neue Cmdlets aus Modulen nachladen	107
Neue Module automatisch nachladen	109
Auslaufmodell: Snap-Ins	112
Alias: Zweitname für Cmdlets	112
Aliase sind keine neuen Befehle	113
Befehlstypen ermitteln	113
Klassische cmd.exe-Interpreterbefehle sind Cmdlets	114
Eigene Aliase anlegen	115
Testaufgaben	116
3 PowerShell-Laufwerke	123
Dateisystemaufgaben meistern	125
Ordner anlegen	127
Dateien anlegen und Informationen speichern	128
Dateien finden	130
Dateien und Ordner kopieren	135
Dateien umbenennen	138
Dateien und Ordner löschen	140
Größe eines Laufwerks ermitteln	141
Größe eines Ordners ermitteln	142
Umgebungsvariablen	143
Alle Umgebungsvariablen auflisten	144
Auf einzelne Umgebungsvariablen zugreifen	145
Umgebungsvariablen ändern	145
Windows-Registrierungsdatenbank	145
Schlüssel suchen	146

Werte lesen	148
Neue Registry-Schlüssel anlegen	150
Werte hinzufügen, ändern und löschen	151
Virtuelle Laufwerke und Provider	152
Neue PowerShell-Laufwerke	154
Mit Pfadnamen arbeiten	156
Unterstützte Platzhalterzeichen in Pfadnamen	157
-Path oder -LiteralPath?	157
Existenz eines Pfads prüfen	159
Pfadnamen auflösen	160
Testaufgaben	160
4 Anwendungen und Konsolenbefehle	163
Programme starten	165
Optionen für den Programmstart festlegen	167
Nicht unterstützte Konsolenbefehle im ISE-Editor	169
Argumente an Anwendungen übergeben	171
Hilfe für Konsolenbefehle anzeigen	171
Beispiel: Lizenzstatus von Windows überprüfen	172
Argumentübergabe kann scheitern	174
Texteingaben an Konsolenbefehle senden	175
Ergebnisse von Anwendungen weiterverarbeiten	176
Error Level auswerten	177
Fragen an Benutzer stellen mit choice.exe	178
Rückgabertext auswerten	180
Rückgabertexte in Objekte verwandeln	182
Rückgabertext analysieren	185
Laufende Programme steuern	187
Feststellen, ob ein Prozess läuft	187
Auf einen Prozess warten	187
Einstellungen laufender Prozesse ändern	188
Prozesse vorzeitig abbrechen	190
Testaufgaben	190
5 Skripte einsetzen	195
PowerShell-Skripte verfassen	196
Skriptcode eingeben	196
Eingabehilfen spüren Tippfehler auf	197
Skript ausführen	198
Sicherheitseinstellungen und Ausführungsrichtlinien	198
Profilskripte – die Autostartskripte	199
Vier verschiedene Profilskripte – pro Host	199
Profilskript anlegen und öffnen	200
Typische Profilskriptaufgaben durchführen	200
Skripte außerhalb von PowerShell starten	201
PowerShell-Startoptionen	202
Befehlsfolgen extern ausführen	205
PowerShell-Code als geplante Aufgabe ausführen	205
Nicht-PowerShell-Skripte öffnen	207
Stapeldateien (Batchdateien)	207
VBScript-Dateien ausführen	209

Skripte digital signieren	210
Digitales Zertifikat beschaffen	211
Zertifikat aus PFX-Datei laden	213
Zertifikat aus Zertifikatspeicher laden	214
Skript digital signieren	217
Digitale Signaturen überprüfen	220
Signaturen mit einem Zeitstempel versehen	221

6 Die PowerShell-Pipeline	223
Aufbau der PowerShell-Pipeline	224
Prinzipieller Aufbau der Pipeline	225
Die sechs wichtigsten Pipeline-Befehle	227
Select-Object	227
Selbst festlegen, welche Informationen wichtig sind	228
Platzhalterzeichen verwenden	229
Weitere Informationen anfügen	232
-First, -Last und -Skip	233
Unsichtbare Eigenschaften sichtbar machen	234
Sonderfall -ExpandProperty	236
Wie ETS Objekte in Text verwandelt	241
Where-Object	243
Where-Object: stets nur zweite Wahl	243
Clientseitiger Universalfilter	244
Leere Elemente aussortieren	245
Fortgeschrittene Syntax bietet noch mehr Möglichkeiten	246
Dateiinhalte filtern	248
IP-Adressen bestimmen	248
Alternativen zu Where-Object	249
Sort-Object	250
Cmdlet-Ergebnisse sortieren	250
Nach mehreren Spalten sortieren	251
Sortierung mit anderen Cmdlets kombinieren	252
ForEach-Object	253
Grundprinzip: eine Schleife	253
Group-Object	258
Häufigkeiten feststellen	258
Gruppen bilden	260
Measure-Object	261
Statistische Berechnungen	263
Ordnergrößen berechnen	263
Mit »berechneten« Eigenschaften arbeiten	264
Datentyp der Sortierung ändern	264
Gruppierung nach bestimmten Textteilen	265
Umwandlung von Byte in Megabyte	265
Hashtables: mehrere Werte übergeben	266
Mehrere Spalten in umgekehrter Sortierung	267
Mehrspaltige Anzeigen	267
Berechnete Spalten hinzufügen	268
Spaltenbreite, Bündigkeit und Gruppenüberschriften	268
Frei wählbare Gruppierungskriterien	270

Pipeline und Performance: Optimierungen	273
Flaschenhals Pipeline	273
Klassische Schleifen sind wesentlich schneller	274
Die Pipeline ist wesentlich reaktionsfreudiger	274
Weniger Speicherbedarf oder mehr Geschwindigkeit?	275
Testaufgaben	275

7 Ergebnisse ausgeben und formatieren	283
Ergebnisse als Text darstellen	284
Ergebnisse in Textdateien schreiben	285
Textformatierung ändern	285
Automatische Textformatierung verstehen	287
Mehrspaltige Anzeige	288
Out-Printer: Ergebnisse zu Papier bringen	291
Out-WinWord: Ergebnisse direkt an Microsoft Word senden	296
Out-PDF: mit Microsoft Word PDF-Reports erstellen	297
Ergebnisse als Objekte exportieren	299
Export-CSV: Export an Microsoft Excel und andere Programme	299
Ergebnisse serialisieren mit XML und JSON	300
HTML-Reports erstellen	303
Objekteigenschaften in HTML-Spalten umwandeln	303
HTML im Webbrowser anzeigen	303
HTML-Reports ansprechend und farbig gestalten	304
Testaufgaben	306

Teil B Objektorientierte Shell

8 Mit Objekten arbeiten	313
Eigenschaften und Methoden	316
Eigenschaften	316
Methoden	319
Eigenschaften und Methoden anzeigen	322
Hilfe für Objekteigenschaften und -methoden finden	323
Ergebnisse eines Befehls untersuchen	325
Unterschiedliche Objekttypen	328
Nach Objekttypen filtern	328
Änderbare Eigenschaften finden	329
Primitive Datentypen sind auch Objekte	331
Eigenschaften lesen	332
Eigenschaften von vielen Objekten gleichzeitig abrufen	333
Eigenschaften ändern	338
Änderungen wirksam werden lassen	338
Methoden aufrufen	345
Eine Methode mit mehreren Signaturen	347
Testaufgaben	349

9 Operatoren und Bedingungen	355
Operatoren – Aufbau und Namensgebung	356
Wie Operatornamen aufgebaut sind	357
Unäre Operatoren	357
Zuweisungsoperatoren	358
Vergleichsoperatoren	360
Unterscheidung zwischen Groß- und Kleinschreibung	361
Unterschiedliche Datentypen vergleichen	362
Vergleiche umkehren	364
Vergleiche kombinieren	365
Vergleiche auf Arrays anwenden	365
Bedingungen	367
if-Bedingung	367
Switch-Bedingung	369
Where-Object	371
10 Textoperationen und reguläre Ausdrücke	373
Texte zusammenfügen	374
Variableninhalte mit doppelten Anführungszeichen integrieren	375
Der Formatierungsoperator »-f«	377
Textstellen finden und extrahieren	383
Texte splitten	384
Informationen in Texten finden	386
Reguläre Ausdrücke: Textmustererkennung	389
Erste Schritte: Mit Textmustern arbeiten	390
Bestandteile eines regulären Ausdrucks	393
Längste oder kürzeste Fassung?	395
Textstellen ersetzen	396
Ersetzungen ohne Zeichenverlust	398
Bezug auf den Originaltext nehmen	398
Delegatfunktionen	399
Rückverweise im Muster	401
Split und Join: eine mächtige Strategie	401
Nach verschiedenen Zeichen splitten	402
Splitten ohne Trennzeichen(verlust): LookBehind und LookAround	403
Mehrere Treffer in einem Text finden	404
[Regex]::Matches einsetzen	405
Ein- und Mehrzeilenmodus	405
Testaufgaben	407
11 Typen verwenden	409
Typumwandlungen	410
Automatische Typzuweisung durch PowerShell	410
Explizite Umwandlung in einen anderen Typ	411
Testumwandlungen zum Validieren	413
Verkettete Umwandlungen	414
Umwandlungen bei Cmdlets	415
Neue Objekte durch Typumwandlungen	415
Implizite Umwandlung und typisierte Variablen	422
Typisierte Variablen	422
Typisierte Parameter	422

Typisierte Eigenschaften und Argumente	423
Vergleichsoperationen	424
Verborgene Befehle in Typen	425
Statische Methoden verwenden	426
Eindeutige GUIDs generieren	427
Dateiextension ermitteln	427
Mathematische Funktionen	428
Zahlenformate konvertieren	429
DNS-Auflösung	430
Umgebungsvariablen	431
Pfade zu Systemordnern finden	432
Konsoleneinstellungen	432
Spezielle Datumsformate lesen	434
Statische Eigenschaften verwenden	436
Neue Objekte herstellen	436
Konstruktoren verstehen	437
Ein Credential-Object zur automatischen Anmeldung	438
Umgang mit XML-Daten	440
WMI-Remotezugriffe mit Anmeldung	441
COM-Objekte verwenden	442
Dialogfeld öffnen	443
Sprachausgabe	445
Office-Automation	446
Zugriff auf Datenbanken	446
Automatische Updates	448
Verknüpfungen anlegen und ändern	448
Netzwerkmanagement	449
Welche COM-Objekte gibt es sonst noch?	450
Webdienste ansprechen	451
SOAP-basierte Webdienste	451
RESTful-Webdienste	453
Neue .NET-Typen finden	454
Type Accelerators untersuchen	454
.NET-Assemblies durchsuchen	456
Typen nach Stichwort suchen	459
Typen mit bestimmten Befehlen finden	461
Typen nachladen	462
Assembly-Namen feststellen	462
Aktuell geladene Assemblies auflisten	463
Zusätzliche Assembly nachladen	463
Assembly aus Datei nachladen	463
Testaufgaben	464

Teil C Automationsprache 467

12 Einfache Funktionen 469

Alles Wichtige: ein Überblick	471
Eigene Funktionen herstellen	471
Parameter definieren	473
Parameter implementieren	475

Funktionen per Modul überall verfügbar machen	476
Hilfe – Bedienungsanleitung hinzufügen	478
Eine bessere Prompt-Funktion	481
Zwingend erforderliche Parameter	481
Eine Funktion mit zwingend erforderlichen Parametern	482
Automatische Nachfrage	483
Argumente ohne Parameter	483
Rückgabewerte festlegen	484
Mehrere Rückgabewerte werden zum Array	484
Return-Anweisung	485
Write-Output	486
Unerwünschte Rückgabewerte unterdrücken	486
13 Skriptblöcke	489
Skriptblöcke: Transportcontainer für Code	490
Einsatzbereiche für Skriptblöcke	492
Skripte sind dateibasierte Skriptblöcke	493
Code in separater PowerShell ausführen	493
Code remote ausführen	494
Rückgabewerte	495
Pipeline-Fähigkeit	499
Schleifen und Bedingungen	500
Gültigkeitsbereiche	502
Zeitkapsel: Closures	503
Delegates	504
Zugriff auf AST und Parser	505
Codesicherheit	508
14 Pipeline-fähige Funktionen	511
Anonyme Pipeline-Funktion	512
Prototyping	513
Pipeline-fähige Funktion erstellen	513
Benannte Parameter	514
Where-Object durch eine Funktion ersetzen	516
Kurzes Resümee	518
Parameter und Pipeline-Kontrakt	518
»ISA«-Kontrakt: Pipeline-Eingaben direkt binden	518
»HASA«-Kontrakt: Objekteigenschaften lesen	521
»HASA« und »ISA« kombinieren	522
CSV-Dateien direkt an Funktionen übergeben	524
Aliasnamen für Parameter	525
Modularer Code mit Pipeline-fähigen Funktionen	526
Ausgangspunkt: ein unleserliches Skript	526
Teil 1: Get-NewFilenameIfPresent	528
Teil 2: ConvertTo-AbsoluteURL	529
Teil 3: Get-ImageFromWebsite	530
Teil 4: Download-File	531

15 Objektorientierte Rückgabewerte	535
Mehrere Informationen zurückgeben	536
Objekte speichern mehrere Informationen strukturiert	537
Eigene Objekte mit Ordered Hashtables anlegen	538
Primär sichtbare Eigenschaften festlegen	540
16 Fortgeschrittene Parameter	543
Argumentvervollständigung	544
Statische Autovervollständigung für Parameter	545
Autovervollständigung über Enumerationsdatentyp	545
Autovervollständigung über ValidateSet	548
Vordefinierte Enumerationsdatentypen finden	548
Eigene Enumerationsdatentypen erstellen	550
Dynamische Argumentvervollständigung	553
Zuweisungen mit Validierern überprüfen	554
ValidateSet	554
ValidateRange	555
ValidateLength	555
ValidatePattern	556
ValidateCount	556
ValidateScript	557
Nullwerte und andere Validierer	557
Parameter in ParameterSets einteilen	558
Gegenseitig ausschließende Parameter	558
Binding über Datentyp	559
Parameter in mehreren Parametersätzen	559
Simulationsmodus (-WhatIf) und Sicherheitsabfrage (-Confirm)	560
Festlegen, welche Codeteile übersprungen werden sollen	561
Weiterleitung verhindern	562
Praxisbeispiel: Automatische Auslagerungsdateien aktivieren	563
Gefährlichkeit einer Funktion festlegen	564
Dynamische Parameter	566
Dynamische Parameter selbst definieren	568
Auf die Argumente dynamischer Parameter zugreifen	570
»Clever« dynamische Parameter	571
Performance und Caching	573
Parameter mit dynamischen Vorschlagslisten	574
Dynamische Währungslisten anzeigen	575
Objektgenerator mit dynamischen Parametern	577
Dynamische Parameter mit dynamischen ValidateSets	580
Splatting: mehrere Parameter gleichzeitig ansprechen	582
Splatting im Alltag einsetzen	583
Übergebene Parameter als Hashtable empfangen	583
Mit Splatting Parameter weiterreichen	584

17 Eigene Module erstellen	589
Module sind Ordner	590
Funktion erstellen und testen	590
Funktion in Modul aufnehmen	591
Modul manuell importieren	592
Module automatisch importieren	592
Manifestdatei für ein Modul	594
Neue Manifestdatei anlegen	594
Wirkung einer Manifestdatei	598
ETS-Anweisungen zu Modul hinzufügen	599
Eindeutige Typnamen zuweisen	600
Neue Formatierungsangaben in ETS einfügen	601
Formatdefinition in Modul integrieren	601
ETS-Formatierung testen	602
Aufbau von FormatData-Definitionen	603
Module entfernen	603
18 PowerShellGet – Module verteilen und nachladen	605
PowerShell Gallery nutzen	607
NuGet-Provider installieren	608
Repository	609
Module finden und installieren	609
Modul herunterladen	610
Modul testweise ausführen	611
Modul dauerhaft installieren	612
Module aktualisieren	613
Side-by-Side-Versionierung	614
Eigene Module veröffentlichen	614
Privates Repository einrichten	615
Freigaben zum Lesen und Schreiben	615
Repository anlegen	615
Modul in Repository übertragen	616
Modul aus Repository installieren	617
19 Gültigkeitsbereiche	619
Gültigkeitsbereiche verstehen	620
Unterschied zwischen Lesen und Schreiben	621
Aufpassen bei Objekten und Referenzen	622
Parameter liefern Referenzen	623
Gemeinsam genutzte Variablen	624
Skriptglobale »Shared« Variable	624
Globale Variable	625
Direkter Zugriff auf Gültigkeitsbereiche	626
Dot-Sourcing: Skripte im Aufruferkontext	631
»Dot-Sourcing« verstehen	632
Aufruftyp eines Skripts testen	633
Gültigkeitsbereiche in Modulen	634
Modulcode greift auf Aufruferkontext zu	634
Aufruferkontext greift auf Modulkontext zu	635

20 Debugging – Fehler finden	637
Syntaxfehler erkennen und beheben	639
Folgefehler und der Blick auf das Wesentliche	639
Formale Regeln missachten	642
Laufzeit- und Logikfehler aufspüren	644
Falsche Verwendung von Operatoren	644
Tippfehler ändern den Code	646
Nicht initialisierte Variablen	646
Versehentliche Verwendung von Arrays	647
Fehlendes Verständnis für Objektreferenzen	649
Falsche Verwendung von Klammern	652
Falsche Zuordnung von Skriptblöcken	653
Schrittweise Ausführung mit dem Debugger	654
Haltepunkte setzen und Code schrittweise ausführen	655
Codeausführung fortsetzen	657
Ad-hoc-Debugging	657
Dynamische Haltepunkte setzen	657
Anhalten, wenn Variablen sich ändern	657
Anhalten, wenn Cmdlets oder Funktionen aufgerufen werden	659
Anhalten, wenn Variablen bestimmte Werte enthalten	660
Debugging-Meldungen ausgeben	662
Ein Skript mit Write-Debug testen	663
Write-Debug in echte Haltepunkte umwandeln	664
Tracing einsetzen	667
Fremde Prozesse debuggen	668
TestszENARIO: PowerShell-Code in fremdem Prozess	669
Testskript in fremdem Host starten	669
Verbindung mit fremdem Host aufnehmen	669
Runspace auswählen und debuggen	670
Debug-Sitzung beenden	670
Remote-Debugging	671
Mit Remotecomputer verbinden	671
Remoteskript laden und debuggen	671
21 Fehlerhandling	673
Fehlermeldungen unterdrücken	674
Bestimmen, wie Cmdlets auf Fehler reagieren	675
Fehler nachträglich analysieren	676
Fehler mitprotokollieren lassen	678
Erfolg eines Befehlsaufrufs prüfen	680
Fehlerhandler einsetzen	680
Lokaler Fehlerhandler: try...catch	681
Globaler Fehlerhandler: Trap	686
Spezifische Fehlerhandler	689
Exception-Namen ermitteln	689
Spezifische Fehlerhandler nutzen	690
Spezifische Fehlerhandler in Traps	691
Spezifität des Fehlerhandlers justieren	697
Eigene Fehler auslösen	699
Mit Throw einen Fehler auslösen	700
Spezifische Fehler auslösen	702

Upstream-Kommunikation in der Pipeline	704
Pipeline vorzeitig abbrechen	704
Pipeline mit Select-Object abbrechen	705
Pipeline manuell abbrechen	705
Testaufgaben	706
Teil D Remoting und Parallelverarbeitung	711
22 Fernzugriff und Netzwerk-Troubleshooting	713
Klassische Fernzugriffe	714
Dateisystem	714
Konsolenbefehle	714
Remotefähige Cmdlets	715
Troubleshooting für Fernzugriffe	716
Firewall für DCOM einrichten	716
Namensauflösung überprüfen	717
Remote-Registrierungszugriff erlauben	718
Access Denied: mit passenden Rechten anmelden	719
LocalAccountTokenFilterPolicy	720
Ports überprüfen	721
23 Windows PowerShell-Remoting	723
PowerShell-Remoting aktivieren	724
Ohne Kerberos und Domäne	726
TrustedHosts-Liste	727
PowerShell-Remoting überprüfen	727
Erste Schritte mit PowerShell-Remoting	729
Remoting-Unterstützung im ISE-Editor	730
Befehle und Skripte remote ausführen	731
Kontrolle: Wer besucht »meinen« Computer?	732
Remotefähigen Code entwickeln	733
Argumente an Remotecode übergeben	733
Ergebnisse vom Remotecode an den Aufrufer übertragen	735
Fan-Out: integrierte Parallelverarbeitung	735
ThrottleLimit: Parallelverarbeitung begrenzen	737
Double-Hop und CredSSP: Anmeldeinfos weiterreichen	738
Eigene Sitzungen verwenden	742
Eigene Sitzungen anlegen	742
Parallelverarbeitung mit PSSessions	743
Sitzungen trennen und wiederverbinden	746
Aktive PSSessions trennen	746
Beispiel: Invoke-Command -InDisconnectedSession	747
Endpunkte verwalten	749
Vorhandene Endpunkte anzeigen	749
Standardendpunkt festlegen	750
Remote auf Endpunkte zugreifen	751
Neue Endpunkte anlegen	752
Zugriffsberechtigungen für Endpunkt festlegen	752
Benutzerkontext ändern	753

Eingeschränkte Endpunkte anlegen	755
Bereitgestellte Befehle weiter einschränken	758
Eigene Funktionen definieren und bereitstellen	761
Reguläre PowerShell-Sitzungen einschränken	765
Limit-Runspace: Befehlsumfang und Sprachmodus bestimmen	765
Endpunkte per Skript absichern	769
Eigene Funktionen einsetzen	770
Sitzungsinhalte importieren	771
Cmdlets eines Remotesystems importieren	771
Remotesitzungen als Modul exportieren	772
Datentransfer und Optimierung	774
Deserialisierte Objekte	774
Serialisierungsvorgang	777
Performanceoptimierung	778
Datentransfer mit PowerShell-Remoting	780
Weitere Remoting-Einstellungen	781
Clientseitige Sitzungsoptionen	781
Zugriff auf Remoting-Einstellungen	782
Einstellungen ändern	783
Fehler finden und beheben	784
RPC-Server nicht verfügbar	784
Zugriff verweigert	785
Kerberos-Fehlermeldung	786
Öffentliche Netzwerke entdeckt	787
Andere Fehler	788
24 Just Enough Admin (JEA)	789
Just Enough Admin (JEA)	790
Virtuelle Administratorkonten	790
Rollenmodell	790
Virtuelle Administratorkonten	791
Voraussetzungen	791
PowerShell-Remoting überprüfen	792
JEA-Endpunkt einrichten	792
JEA-Endpunkt verwenden	794
Gefahren bei der Elevation	798
Unterschieben bössartiger Module und Befehle	799
Angriffsszenario mit Administratorrechten	799
Angriffsszenario ohne Administratorrechte	802
Best Practices für JEA-Endpunkte	804
Rollenbasierte Rechtevergabe	805
Rollen anlegen	805
Befehlsumfang der Rollen definieren	807
Praxisbeispiel: Verwaltung im Active Directory	810
Endpunkt für die Verwaltung des Active Directory	811
Modul für die Verwaltung des Active Directory	812
Rollenbasierte Verwaltung	813

25 Hintergrundjobs und Parallelverarbeitung	817
Hintergrundjobs	818
Mehrere Aufgaben parallelisieren	819
Integrierte Hintergrundjobs	822
Hintergrundjobs auf Remotecomputern	824
Multithreading	825
Einen separaten Thread erzeugen	826
Hintergrundüberwachungen einrichten	828
Foreach-Schleife mit Parallelbearbeitung	829
Foreach-Parallel nachrüsten	829
Maximale Thread-Anzahl festlegen	833
Maximale Ausführungszeit festlegen	834
Lokale Variablen einblenden	835
Hintergrundjobs auf Thread-Basis	837
Teil E DevOps und Enterprise	843
26 Workflows	845
Brauche ich Workflows?	846
Wie relevant sind Workflows wirklich?	847
Aufgaben orchestrieren	848
Orchestrierung in klassischer Funktion	848
Orchestrierung mit Workflow: sequence und parallel	849
Workflows sind kein PowerShell-Code	851
Syntaktische Unterschiede und Kompatibilitätsprobleme	853
InlineScript: echten PowerShell-Code ausführen	854
Variablen und Gültigkeitsbereiche	856
Zugriff auf »fremde« Variablen	856
Workflow-globale Variablen	857
Variablen in InlineScript	859
Informationen mit Parametern übergeben	859
Ergebnisse über Rückgabewerte zurückliefern	860
Funktionen verwenden	862
Verschachtelte Workflows	864
Remotezugriff	864
Parallelverarbeitung	865
Globale Variablen synchronisieren parallele Aufgaben	866
Parallelbearbeitung in einer Schleife	867
Throttling für Parallelschleifen	868
Unterbrechung und Wiederaufnahme	870
Manuelle Unterbrechung	870
Automatische Unterbrechung	872
Persistierende Hintergrundjobs	874
Prüfpunkte für eine kontrollierte Wiederaufnahme	875
Workflow-Server: Schutz vor unkontrollierten Abstürzen	876
Workflows optimieren	879

27 Desired State Configuration (DSC)	881
Workflows und DSC – eine Reise ins DevOps-Land	883
Voraussetzungen für DSC	884
Architektur	885
Was sind Ressourcen?	886
Mitgelieferte Ressourcen untersuchen	886
Integrierte Testfunktion in Ressourcen	887
Aktuellen Zustand ermitteln	888
Änderung durchführen	888
Was sind Konfigurationen?	889
Eine einfache DSC-Konfiguration erstellen	889
Konfiguration als MOF-Datei speichern	891
Konfiguration auf Computer anwenden	892
Was ist der Local Configuration Manager?	894
Die Master-Konfiguration	894
Reset: LCM zurücksetzen	895
Konfiguration überprüfen	896
Konfigurationshistorie abrufen	898
Das »Was?« vom »Wo?« abgrenzen	899
Schlecht: Umgebungsinformationen in Konfigurationen	899
Umgebungsinformationen ausgliedern	900
Umgebungsinformationen als ConfigurationData übergeben	901
Komplexere Umgebungen definieren	903
Identität und Umgang mit Geheimnissen	908
Konfigurationen im Benutzerkontext ausführen	908
Sensible Informationen verschlüsseln	913
Kennwörter in DSC-Konfigurationen verschlüsseln	917
Mitgelieferte Ressourcen	920
Verfügbare Ressourcen auflisten	920
Ressource »Archive«	922
Ressource »Environment«	925
Ressource »Group«	927
Zusätzliche Ressourcen	927
Ressourcen aus der PowerShell Gallery beziehen	928
Eigene DSC-Ressourcen schreiben	936
ResourceDesigner installieren	936
Ressource konzipieren	937
Ressource anlegen	938
Get-TargetResource entwerfen	941
Test-TargetResource	942
Set-TargetResource	943
Neue Ressource testen	949
Neue Ressource in DSC-Konfiguration nutzen	951
Orchestrierung	952
Abhängigkeiten definieren	952
Systemübergreifende Abhängigkeiten	954
Partielle Konfigurationen	956
LCM konfigurieren	960
Letzte Konfiguration wiederherstellen	962
Konfigurationen automatisch überwachen und anwenden	964

Teil F	Spezielle Techniken	967
28	Ereignisverarbeitung mit Events	969
	Ereignisse verwenden	970
	Ein Ereignis asynchron überwachen	970
	Ein Ereignis synchron überwachen	972
	Hintergrundjobs überwachen	973
	Manuelle Überwachung	973
	Automatische Überwachung	974
	Ordner überwachen	975
	Aufgaben regelmäßig durchführen	976
	WMI-Ereignisse empfangen	977
	Details zum Event erfahren	978
	Systemänderungen erkennen	979
	Eigene Ereignisse auslösen	980
	Automatische Variablenüberwachung einrichten	980
29	Extended Type System (ETS)	983
	PowerShell-Objekte verstehen	984
	Erweiterte PowerShell-Objekte	984
	Objekte mit Add-Member erweitern	986
	Dauerhafte Erweiterungen	988
	AliasProperty: Eigenschaften »umbenennen«	991
	NoteProperty: Objekte »taggen«	992
	ScriptProperty: »berechnete« Eigenschaften	992
	Lesbare Eigenschaften	992
	Lesbare und schreibbare Eigenschaften	994
	ScriptMethod und ParameterizedProperty	996
	Membertypen für den internen Gebrauch	1000
	PropertySet: Gruppen von Eigenschaften	1000
	MemberSet: Wie soll PowerShell das Objekt behandeln?	1001
30	Proxyfunktionen verstehen und einsetzen	1005
	Eine Proxyfunktion erstellen	1006
	Bestehende Cmdlets erweitern	1007
	Automatische Protokollfunktion	1007
	Get-ChildItem mit neuen Parametern	1009
	Proxyfunktion anlegen	1010
	Logik implementieren	1012
	Proxyfunktionen für Remoting	1017
	Eine Remotesitzung erstellen	1017
	Einen Remotebefehl in lokale Sitzung importieren	1018
31	Benutzeroberflächen gestalten	1019
	Eigene Fenster mit WPF erzeugen	1020
	Ein Fenster gestalten – allein mit Code	1021
	Ein Fenster gestalten – mit XAML	1022
	Auf Elemente des Fensters zugreifen	1024
	Auf Ereignisse reagieren	1025

Werkzeuge für das WPF-Design	1028
Mit dem WPF-Designer von Visual Studio arbeiten	1029
Neue grafische Elemente einfügen	1030
Elemente im Fenster anordnen	1033
StackPanels	1033
Grids	1034
DockPanels	1035
Datenbindungen	1040
Ereignisse behandeln	1044
Ereignisse mit Werkzeugen erforschen	1045
Ereignisse manuell erforschen	1050
Multithreading	1051
Fenster anzeigen und weiterarbeiten	1051
Oberfläche aktualisieren	1053
Aufgaben im Hintergrund durchführen	1058
Mehrere Threads verwenden	1060
Thread-übergreifende Aktionen	1064
Thread-übergreifendes Databinding	1068
Bilder in WPF einbetten	1071
Bild in Text verwandeln	1071
Text in Bild verwandeln	1071
32 Pester – »Test-Driven Development«	1075
Pester installieren oder aktualisieren	1076
Eine simple Funktion auf TDD-Art entwickeln	1078
Eine neue Funktion entwerfen	1078
Einen Test formulieren	1079
Test ausführen	1079
Funktionalität implementieren	1080
Architektur der Pester-Tests	1081
Gefahren und Vorsichtsmaßnahmen	1082
Funktionen nachträglich erweitern	1082
Assertions – Ergebnisse überprüfen	1085
Die richtige Assertion wählen	1085
Eine Assertion testen	1086
Mehrere Assertions kombinieren	1087
Simulationen und Alltagstests	1088
Befehle vorübergehend außer Kraft setzen	1088
Mock-Funktion über Parameter auswählen	1090
Reale Tests und notwendige Aufräumarbeiten	1091
TestDrive – ein Ort für temporäre Daten	1094
»Test Cases« und Wiederverwertung	1095
»Code Coverage« und Eigenentwicklungen	1096
Testabdeckung überprüfen	1096
Eigene Testtools auf Pester aufsetzen	1096
33 PowerShell-Umgebung anpassen	1099
Profilskripte einsetzen	1100
Vier Profilskripte	1100
Profilskripte öffnen und ändern	1101
Profilskripte bei Bedarf nachladen	1103

Inhaltsverzeichnis

Eingabeaufforderung anpassen	1103
Konsolendarstellung verbessern	1105
ISE-Editor erweitern	1106
Auf den Editor zugreifen	1106
Befehle über das Menü anbieten	1109
Add-On-Tools verwenden	1112
Zugriff auf die Skriptstruktur	1115
Zwei Parser: Text in Token verwandeln	1115
Kommentare entfernen	1117
Aliase auflösen	1118
Syntaxfehler finden	1119
Abstract Syntax Tree (AST)	1120
34 .NET Framework und PowerShell	1123
Auf API-Funktionen zugreifen	1124
API-Funktionen in PowerShell einblenden	1125
API-Funktion einsetzen	1125
Wiederverwertbare PowerShell-Funktion herstellen	1126
PowerShell-Klassen einsetzen	1127
Neue Klasse zur besseren Prozesse-Verwaltung	1129
Statische Eigenschaften und Methoden	1133
Vererbung von Klassen	1134
Eine abgeleitete Klasse erstellen	1135
Abgeleitete Klasse einsetzen	1136
Die wesentlichen Aspekte der Vererbung	1136
Eine weitere abgeleitete Klasse	1137
C#-Code verwenden	1138
Klasse mit PowerShell-Mitteln erstellen	1138
Klasse mit C#-Code entwickeln	1142
Index	1147

Einführung

Wer braucht eigentlich PowerShell?

Jeder, der eine Aufgabe zweimal tut, sollte über Automation nachdenken. In der IT müssen viele Aufgaben nicht nur zweimal, sondern hundert- oder tausendfach erledigt werden. Und jeder kennt die wenig geliebten und immer wiederkehrenden Aufgaben: Monatliche Reports müssen erstellt oder Logbuchdaten durchforstet werden.

Was immer es ist und ganz gleich, ob beruflich oder privat motiviert: PowerShell ist die Sprache, mit der Sie solche Aufgaben beschreiben und danach so oft Sie mögen – und wann Sie mögen – von PowerShell ausführen lassen können. Das ist nicht nur bequemer, es kann auch im Not- oder Krisenfall wichtig sein. Wer dann die entsprechenden Notfallskripte zur Hand hat, lässt die in ruhigen Zeiten erarbeiteten Aufgaben blitzschnell abarbeiten.

Während Sie kleinere Aufgaben auf Ihrem eigenen Computer mit PowerShell automatisieren, ist diese Automationsprache auch für flächendeckende und unternehmensweite Automation hervorragend geeignet. Es gibt hier kaum Grenzen, und die PowerShell-Automation kann ganze Unternehmensstrukturen abbilden.

Über das sogenannte »PowerShell Remoting« (Kapitel 23) kommunizieren PowerShells über das Netzwerk miteinander, und dank PowerShell 6 steht PowerShell auch in heterogenen Landschaften zu Beispiel auf Linux und MacOS zur Verfügung.

Was die Frage aufwirft: Warum handelt dieses Buch eigentlich von PowerShell 5, wenn es offenbar auch ein PowerShell 6 gibt?

»Windows PowerShell« 5 vs. »PowerShell« 6

Seit der ersten »Windows PowerShell«-Version in 2006 ist inzwischen »Windows PowerShell« 5 auf allen Windows-Systemen ab Windows 7 und Server 2008R2 der empfohlene Standard. Sie müssen allenfalls noch Ihre »Windows PowerShell«-Version auf die aktuellste Version bringen, wobei Ihnen Kapitel 1 zur Seite steht. »Windows PowerShell« ist nun »fertiggestellt« und wird sich auch nicht mehr wesentlich ändern.

Stattdessen arbeitet Microsoft jetzt an »PowerShell« 6 weiter.

Einführung

Huh? Ist es also doch nicht fertig? Die Wahrheit ist: »PowerShell« 6 ist nicht der Nachfolger von »Windows PowerShell« 5. Lassen Sie uns also kurz ein wenig Klarheit in die PowerShell-Versionen bringen, damit Sie wissen, welche PowerShell für Sie richtig ist.

»Windows PowerShell« ...

Die fest in Windows integrierte PowerShell nennt Microsoft »Windows PowerShell«. Sie wird auch künftig in Windows integriert sein und basiert auf dem vollständigen .NET Framework, also den vollständigen Programmierschnittstellen des Windows-Betriebssystems. »Windows PowerShell« kann deshalb alle Aspekte und Funktionen von Windows verwalten und automatisieren.

Diese »Windows PowerShell« gilt nun in Version 5 als ausgereift und fertiggestellt. Sie haben auf Jahre Planungssicherheit.

... und »PowerShell«

»PowerShell« 6 ist eine Neuentwicklung, die auf einem eingeschränkten und portablen .NET Framework basiert (dem sogenannten .NET Core). Es ist auch auf Linux und MacOS lauffähig und steht ausschließlich als optionaler Download von <https://github.com/PowerShell/PowerShell> zur Verfügung.

Der Vorzug von PowerShell 6 ist nicht etwa ein noch größerer Funktionsumfang. Wegen des eingeschränkten .NET Framework ist eher das Gegenteil der Fall. Der Vorzug von PowerShell 6 ist seine universelle Einsetzbarkeit. Es ist nicht mehr auf Windows beschränkt.

Zwar ist es technisch möglich, aber derzeit nicht besonders sinnvoll, auf Windows-Systemen die »PowerShell« 6 einzusetzen. Greifen Sie besser zur eingebauten »Windows PowerShell«. Sie kann deutlich mehr. Nur wenn Sie Nicht-Windows-Systeme in Ihre Automationspläne integrieren möchten, verwenden Sie dort »PowerShell« 6.

Über das »PowerShell Remoting« aus Kapitel 23 können »Windows PowerShell« und »PowerShell« 6 quer über das Netzwerk miteinander kommunizieren.

Warum PowerShell lernen?

PowerShell-Analphabetismus konnte man als Windows-Administrator in den ersten Jahren noch gut kaschieren, doch inzwischen lässt sich dieses Defizit kaum noch verbergen. Zu allgegenwärtig ist PowerShell im Betriebssystem und in vielen Softwareprodukten, und wer es nicht beherrscht, kann häufig schlicht einen Teil seines Jobs nicht mehr erledigen.

Umgekehrt wird ein angenehmerer Schuh daraus: Wer sich bislang erfolgreich vor PowerShell »gedrückt« hat, kann sich mit einigen Wochen des Lernens (natürlich verbunden mit den üblichen Frustrationen, die zu jedem Erlernen dazugehören) das gesamte Funktionsspektrum der aktuellsten Windows-PowerShell-Version 5 zu eigen machen. Seit es PowerShell 6 auf Linux und MacOS gibt, können Sie Ihr PowerShell-Wissen sogar ohne großen Aufwand auch auf andere Betriebssysteme übertragen, und wer PowerShell beherrscht, wird für sein Unternehmen ein immer wertvollerer Mitarbeiter.

Moderne Lernkurve

Wer vor der PowerShell-Konsole sitzt, sieht vor allem Befehle – die *Cmdlets*. Sie funktionieren etwa so wie Befehle in anderen Konsolen (*Shells*). Allerdings unterstützt PowerShell so viele davon und ist auf so gleichartige Weise erweiterbar, dass man mit ganz geringen (und vor allen Dingen vollkommen einheitlichen und konsistenten) Grundkenntnissen fast alles damit administrieren kann. Weil alle Cmdlets auf denselben Grundregeln basieren, kann man das Wissen auch leicht auf andere Aufgabenbereiche und Cmdlets anwenden.

Die ersten Schritte mit Cmdlets sind einfach und benötigen kaum PowerShell-Kenntnisse. Auch der Umgang mit der PowerShell-Pipeline ist zunächst recht einfach. Dann aber steht man vor einer Steilwand, die autodidaktisch oft nur schwer zu meistern ist. Hier werden viele wichtige Grundlagen gelegt. Man beschäftigt sich mit Operatoren, Objekten und der wahren Natur von PowerShell. Diese Steilwand meistern Sie mit diesem Buch.

Sobald die Grundlagen einmal gelegt sind, flacht die Lernkurve stark ab, und alles wird gut: Ob Sie mit Microsoft Exchange ein Postfach anlegen, mit SharePoint eine Site veröffentlichen oder einfach nur Protokolldateien parsen oder Server verwalten wollen – Ihr PowerShell-Wissen befähigt Sie dazu, über den Tellerrand zu blicken und Ihre PowerShell-Erfahrung auch in ganz anderen IT-Bereichen einzusetzen. Entscheidend ist nun nur noch Ihr Fachwissen im jeweiligen Bereich.

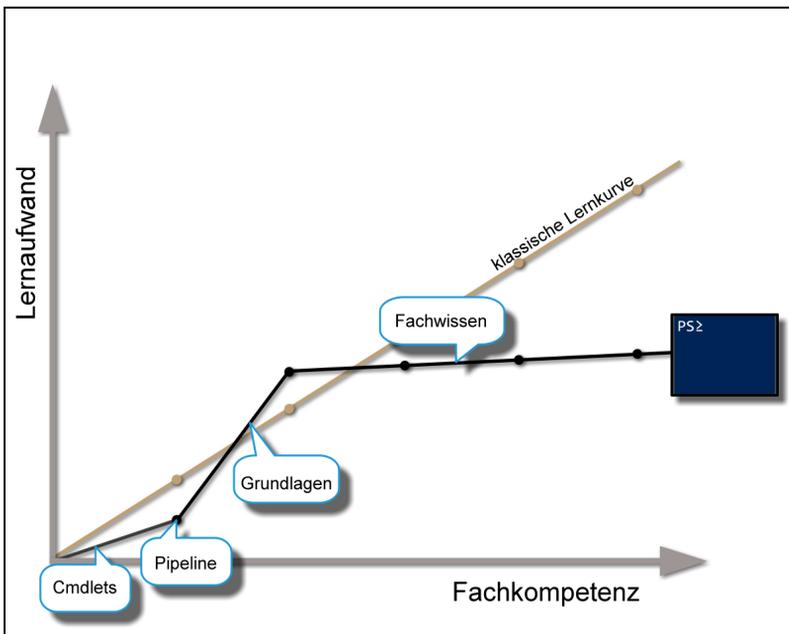


Abbildung E.1: Die PowerShell-Lernkurve ist nicht linear, sondern baut auf gemeinsamen Standards auf.

Sie müssen zwar schon selbst wissen, »warum« Sie ein Exchange-Postfach anlegen oder eine SharePoint-Site umbenennen wollen – »wie« das geschieht, ist aber dank der gemeinsamen PowerShell-Standards nun keine Frage mehr. `Get-ADUser`, `Get-Mailbox`, `Get-SPSite` und `Get-VM` beschaffen Ihnen mit denselben PowerShell-Grundkenntnissen Active-Directory-Benutzerkonten, Exchange-Server-Mailboxen, SharePoint-Sites oder VMware virtuelle Maschinen. Kennen Sie erst einmal ein Cmdlet, dann kennen Sie alle.

Computer – ich/ich – Computer: PowerShell als Fremdsprache

Entfernt man sich ein wenig von PowerShell und betrachtet die Technik mit mehr Abstand, verschwimmen die einzelnen Befehle zu etwas Neuem: einer Sprache. Tatsächlich benimmt sich PowerShell in vielen Aspekten genau wie eine Fremdsprache, wie Spanisch oder Italienisch etwa, nur dass Sie sich diesmal nicht mit dem Eisverkäufer unterhalten, sondern mit Ihrem Computer. Auch der Lernprozess, den Sie mit diesem Buch vor sich haben, verläuft ganz ähnlich.

Zuerst steht Vokabelpauken auf dem Plan, und schon nach wenigen Minuten werden Sie mit den ersten Vokabeln bereits einfache Aufgaben lösen können. Dieser erste Lernschritt bildet den Teil »**Interaktive Befehlskonsole**« dieses Buchs (**Kapitel 2–7**). Hier lernen Sie den Umstieg von der klassischen *cmd.exe*-Befehlskonsole zur PowerShell, und für viele Administratoren reicht das Wissen aus diesem Teil bereits.

Dass PowerShell indes nicht eine klassische Textkonsole ist, sondern in Wirklichkeit eine hochmoderne objektorientierte Shell, entdecken Sie im Teil »**Objektorientierte Shell**« (**Kapitel 8–11**).

PowerShell als Ersatz für VBScript und Batch

Nachdem Sie die Sprache beherrschen, wird bei Ihnen mit einiger Wahrscheinlichkeit der Wunsch aufkommen, PowerShell nicht nur interaktiv zu verwenden, sondern auch als vollwertige Programmier- und Skriptsprache, um damit einfache und auch komplexe Automationsaufgaben zu lösen. Dies erwartet Sie im Teil »**Automationsprache**« (**Kapitel 12–21**): Hier geht es zum einen darum, wiederkehrende Aufgaben mit PowerShell-Lösungen ein für alle Mal zu lösen.

Zum anderen geht es aber vor allem darum, wie Sie in PowerShell selbst automatisieren, also dafür sorgen, dass Sie nicht ständig ein Rad neu erfinden müssen. Sie erfahren, wie sich Ihre Automationslösungen in neue eigene PowerShell-Befehle kapseln lassen und wie Sie diese neuen Befehle für sich selbst und Ihre Kollegen bereitstellen können. Mit jeder neuen Lösung, die Sie entwickeln, wächst so Ihr Lösungsspektrum.

Grenzenloses PowerShell

PowerShell wird zwar auf lokalen Computern ausgeführt, kennt aber kaum Grenzen. Es steht inzwischen nicht nur auf Windows-Betriebssystemen zur Verfügung, sondern auch auf Linux und MacOS. Alle PowerShells können über das Netzwerk miteinander sprechen, sich abstimmen und so die Grundlage für übergreifende und unternehmensweite Automationslösungen sein. Lesen Sie dazu mehr im Teil »**Remoting und Parallelverarbeitung**« (**Kapitel 22–25**).

Neben der »räumlichen« Unabhängigkeit enthält PowerShell auch Wege, um die Sicherheitskontexte und Identitäten zu wechseln, in denen Befehle ausgeführt werden. Auch diese Technik beruht im Kern auf dem Remoting und bildet die Grundlage für JEA (*Just Enough Admin*), mit der eine rollenbasierte Administration möglich wird, zum Beispiel für Self-Servicing-Portale.

Strategische Plattform und Orchestrierung

Wer seine Infrastruktur »via Code« definieren und ausrollen möchte, findet in PowerShell auch hierzu machtvolle Lösungen.

Hierbei trennt PowerShell das »Was« vom »Wo«. Das »Was« könnte die Definition eines Webservers oder Active Directory sein. Das »Wo« entspräche den Konfigurationsdaten, also den jeweiligen Server- und Webnamen. Das »Wann« kann wiederum beliebig gewählt werden, denn sobald Infrastruktur als Code definiert ist, kann man die dahinterliegenden Automationskripte jederzeit und beliebig häufig einsetzen. Mehr dazu zeigt Ihnen **»DevOps und Enterprise« (Kapitel 26/27)**.

Anwendungsentwicklung

Zwar war PowerShell ursprünglich als Automationsprache gedacht, die unbeaufsichtigt Aufgaben durchführt. Inzwischen wird PowerShell allerdings häufig auch zur Anwendungsentwicklung eingesetzt: Administratoren schreiben sich maßgeschneiderte Tools selbst oder stellen Tools dem Helpdesk zur Verfügung.

Wie man solche Oberflächen in PowerShell gestaltet, ist ebenso Thema des Teils **»Spezielle Techniken« (Kapitel 28–34)** wie innovative neue Wege, um Skripte zuverlässig zu entwickeln und weiterzupflegen. Mit dem *Test-Driven Development* (TDD), das ebenfalls aus der modernen Softwareentwicklung stammt, lassen sich PowerShell-Skripte äußerst zuverlässig und mit eingebauter Qualitätskontrolle herstellen.

Persönliche Entwicklung

PowerShell kann eine Karriereentscheidung sein. Wann immer Sie eine Aufgabe lösen, stehen dahinter unsichtbare Motivationen. Die offensichtlichste ist natürlich, die Aufgabe gut zu erledigen, denn dafür werden Sie (wahrscheinlich) bezahlt. Ebenso wichtig ist aber, was die Lösung dieser Aufgabe sonst noch für Sie bedeutet und wie sie in Ihre Lebensbilanz einfließt. Wer sich Tag für Tag durch Dialogfelder klickt, kann zwar enorm erfolgreich Aufgaben lösen, entwickelt seine Fähigkeiten aber nicht weiter, und wenn das Dialogfeld eines Tages nicht mehr da ist, gilt das vielleicht auch für den eigenen Arbeitsplatz.

Es mag Sie anfangs etwas mehr Zeit kosten, die Lösung damit zu automatisieren, aber bedenken Sie: Jede Extraminute, die Sie hier investieren, investieren Sie eigentlich in Ihre persönliche Fortbildung. Auch ein Arbeitgeber sollte dies als Chance verstehen und Freiräume dafür gestatten. Denn mit jeder erfolgreich gemeisterten PowerShell-Lösung wächst Ihre Sprachfertigkeit. Wer PowerShell am Ende fließend spricht, ist bestens aufgestellt für moderne IT-Landschaften.

Gerade wenn Sie vorher noch nie geskriptet haben, sehen Sie PowerShell als Chance: Wer den Zug zu VBScript-Zeiten vor zehn Jahren vielleicht verpasst hat, kann heute auf einen neuen Zug aufspringen.

Wie Sie dieses Buch nutzen

Dieses Buch setzt keinerlei Grundkenntnisse voraus, zumindest wenn Sie von vorn zu lesen beginnen. Wer unter Zeitdruck steht, kann auch quer einsteigen, und wer noch weniger Zeit hat, findet am Anfang jedes Kapitels eine Zusammenfassung, in der die jeweils wichtigsten Inhalte für Krisenzeiten zusammengefasst sind.

Die PowerShell-Beispiele im Buch sind jeweils in einer anderen Schriftart formatiert. Damit Sie leichter erkennen, welche Eingaben von Ihnen erwartet werden, wird bei allen Eingaben die PowerShell-Eingabeaufforderung `PS>` (einschließlich der Leerstelle hinter dem `>`) vorangestellt. Diese Eingabeaufforderung kann bei Ihnen auch anders aussehen und sollte in den Beispielen natürlich nicht mit eingegeben werden.

Viele PowerShell-Codebeispiele sind sehr kurz und können mit geringem Aufwand schnell eingetippt werden. Umfangreichere Beispiele sind mit einer Listing-Unterschrift gekennzeichnet. Unter dem dort genannten Dateinamen finden Sie die Codebeispiele auch in den Begleitmaterialien, die Sie hier herunterladen können: <http://downloads.oreilly.de/9783960090090>.

Sollte es zu diesem Buch Errata geben, finden Sie sie ebenfalls hier: <http://downloads.oreilly.de/9783960090090>.

Achtung

Bitte verwenden Sie die Begleitmaterialien immer im Kontext des entsprechenden Buchkapitels. Viele der Beispiele funktionieren nur, wenn Sie die entsprechenden Vorarbeiten im Kapitel beachtet haben, oder können auch unerwartete Resultate liefern, wenn man die Beispiele aus dem Zusammenhang des Kapitels reißt.

Noch mehr Unterstützung

Inzwischen gibt es in fast jedem Land eine starke PowerShell-Community. Auch in Ihrer Nähe finden sich »PowerShell User Groups«, in denen man kostenlos viel dazulernen und persönliche Kontakte aufbauen kann. Jährlich finden zudem internationale PowerShell-Konferenzen wie die »PowerShell Conference Europe« (<http://www.psconf.eu>) statt.

Falls bei der Arbeit mit diesem Buch Fragen auftauchen oder Sie Anregungen haben, besuchen Sie mich: <http://www.powertheshell.com>. Oder senden Sie mir eine Nachricht an meine Mailadresse tobias.weltner@email.de.

Bevor ich Ihnen jetzt viel Spaß mit PowerShell wünsche, geht noch ein großes Dankeschön an meine Lektorin Ariane Hesse und die Korrektorin Sibylle Feldmann, die dieses Buch mit allergrößtem Sachverstand und mit Sorgfalt begleitet haben.

Herzlichst Ihr

Dr. Tobias Weltner

Kapitel 1

PowerShell kennenlernen

In diesem Kapitel:

Die PowerShell-Konsole einrichten	32
PowerShell ISE einsetzen	40
Erste Schritte mit PowerShell	44
Skriptausführung erlauben	56
Tippfehler vermeiden und Eingaben erleichtern	57
PowerShell-Hilfe aus dem Internet nachladen	60
Klassische Konsole oder moderner ISE-Editor?	63
Testen Sie Ihr Wissen!	66

Ausführlich werden in diesem Kapitel die folgenden Aspekte erläutert:

- **PowerShell-Host:** PowerShell ist Bestandteil von Windows und kein einzelnes Programm. Programme, die den Zugriff auf PowerShell ermöglichen, werden »Host« (»Gastgeber«) genannt. PowerShell liefert zwei Hosts mit: die PowerShell-Konsole (*PowerShell.exe*) und den komfortableren ISE-Editor (*PowerShell_ise.exe*).
- **Groß- und Kleinschreibung:** Die Groß- und Kleinschreibung wird bei Befehlen und Parameternamen nicht unterschieden.
- **Farbcodierung während der Eingabe:** Ab PowerShell 5 färbt nicht nur der ISE-Editor, sondern nun auch die PowerShell-Konsole Eingaben ein. Die Farben unterscheiden zwischen Befehlen, Parametern und Argumenten. So kann man Eingaben mit einem kurzen Blick auf die Farben auf Richtigkeit überprüfen. Enthält die Eingabe Syntaxfehler, also formale Fehler wie fehlende Anführungszeichen, kennzeichnet ISE diesen Teil mit einer roten Wellenlinie. Die Konsole zeigt eine rote spitze Klammer am Ende des Eingabeprompts an.

- **Ausgabebefehle und Umwandlungsabkürzungen:** PowerShell gibt Resultate sofort aus. Ein spezieller Ausgabebefehl wie `echo` ist nicht nötig. Auch unterstützt PowerShell einfache Rechenaufgaben, bei denen die in der IT üblichen Größenordnungen wie KB oder GB direkt (ohne Leerzeichen) an eine Zahl angefügt werden können. Mit dem Präfix `0x` werden hexadezimale Zahlen markiert, und `..` liefert einen Zahlenbereich, zum Beispiel `1..49`.
- **Autovervollständigung:** Mit `↑` und `↓` gelangen Sie zurück zu Befehlsfolgen, die Sie schon einmal eingegeben haben. Möchten Sie einen Befehl nachträglich ändern oder erweitern, verwenden Sie die Pfeiltasten, um, anstatt den gesamten Befehl neu einzugeben, zu dem jeweiligen Befehl zurückzukehren und ihn zu ändern. Mit `↵` aktivieren Sie die eingebaute Autovervollständigung. Diese kann Befehlsnamen, Pfadnamen und andere Eingaben für Sie vervollständigen. Drücken Sie die Taste mehrmals, zeigt PowerShell bei jedem Druck einen anderen Vorschlag. In ISE steht außerdem das IntelliSense-Menü zur Verfügung, das über `[Strg]+[Leertaste]` Eingabevorschläge nicht sofort einfügt, sondern zuerst in einem Kontextmenü anbietet.
- **Zeilen löschen und Befehlsabbruch:** Wollen Sie die gesamte aktuelle Zeile löschen, drücken Sie `[Esc]`. Möchten Sie im Mehrzeilenmodus die aktuelle Zeile zwar nicht ausführen, aber auch nicht verlieren, drücken Sie `[Strg]+[C]`.
- **Skriptausführung:** Anfangs kann PowerShell nur interaktive Befehle ausführen, aber keine Skripte. Mit `Set-ExecutionPolicy` sollte die Skriptausführung so bald wie möglich aktiviert werden, weil viele interaktive Befehle aus Skriptdateien geladen werden und andernfalls nicht funktionieren.
- **Hilfe zu PowerShell-Befehlen:** PowerShell-Befehle sind gut dokumentiert, aber die Dokumentation muss zunächst mit `Update-Help` aus dem Internet heruntergeladen werden.
- **Unterschiedliche PowerShell-Versionen:** Es gibt aktuell fünf PowerShell-Versionen, die alle aufeinander aufbauen. Die aktuelle PowerShell-Version erfährt man zum Beispiel über den Befehl `$host.Version`.

Häufig wird PowerShell gleichgesetzt mit der typisch blauen Textkonsole, die kryptische Befehlseingaben von Ihnen erwartet. Dieses Bild wäre aber falsch. PowerShell ist keine Textkonsole.

PowerShell ist eine Automationssprache und vollkommen unabhängig von bestimmten Programmen. Sie ist als Teil des »Windows Management Framework« (WMF) fest in das Betriebssystem Windows integriert. Die PowerShell-Konsole (*PowerShell.exe*) ist also lediglich ein Programm, mit dem man Kontakt zur PowerShell aufnehmen kann. Andere Programme können das auch, und so ist die PowerShell-Konsole längst nicht das einzige Programm, in dem Ihnen die Sprache »PowerShell« begegnet.

Bettet ein Programm – so wie die PowerShell-Konsole – die PowerShell-Sprache ein, wird es »PowerShell-Host« oder einfach nur »Host« genannt. Dieses englische Wort steht für »Gastgeber«, und so sind PowerShell-Programme Gastgeber für die PowerShell, bieten also die Bühne und Ihnen die Möglichkeit, Befehle an PowerShell zu senden und Ergebnisse zu empfangen. Ein weiterer solcher Host ist ebenfalls Bestandteil von Windows: der modernere »ISE«-Editor (»Integrated Script Environment«, *PowerShell_ise.exe*).