

1 PowerShell für Kurzentzschlossene	1
1.1 Das WMF im Überblick.	1
1.2 Die .NET-Laufzeit	3
1.2.1 Assemblys	3
1.3 Die Objekt-Pipeline	5
1.4 PSProvider und PSDrives	6
1.4.1 Dynamische Parameter	7
1.5 Functions, Aliase, Workflows und Configurations.	8
1.6 Erweiterbarkeit.	8
1.6.1 Ein Modulmanager in Gestalt des PowerShellGet-Moduls.	10
1.7 Befehlssyntax.	11
1.8 Die moderne Konsole.	12
1.9 Hilfe	13
1.10 Andere Plattformen	14
2 Der Umgang mit Typen	15
2.1 Alles ist ein Objekt.	15
2.1.1 Objekte basieren auf Typen	15
2.1.2 Den Typ eines Objekts herausfinden	16
2.2 Typen als Objekte.	16
2.3 Die Typen eines Arrays erhalten	17
2.3.1 Auflisten der Konstruktoren einer Klasse	17
2.4 Neue Objekte anlegen mit dem New-Object-Cmdlet	19
2.4.1 Neue Objekte anlegen über die statische Methode New	19
2.4.2 Objekte mit Argumenten anlegen	19
2.4.3 Neue Objekte anlegen per [PSCustomObject].	20
2.5 Objektschreibweise	20
2.6 Das versteckte Member PSObject	21
2.7 Typen erweitern	21

2.8	Objekte erweitern über das Add-Member-Cmdlet.	23
2.9	Typen definieren über externen Code	24
2.9.1	Zusammenfassung	26
3	Klassen definieren mit dem <i>class</i>-Befehl.	27
3.1	Einleitung.	27
3.1.1	Befehlswörter für die Definition von Klassen	28
3.2	Klassen definieren	28
3.3	Hinzufügen von Eigenschaften	29
3.4	Aus Klassen Objekte machen	31
3.5	Statische Member.	31
3.6	Enumerationen	32
3.7	Aus Klassen werden Objekte.	33
3.8	Klassen mit einem Konstruktor	34
3.8.1	Konstruktor mit Parameter.	35
3.8.2	Versteckte Members.	35
3.8.3	Überladene Konstruktoren	37
3.8.4	Klassen mit einem statischen Konstruktor.	38
3.9	Methoden definieren	38
3.9.1	Methoden überladen.	40
3.9.2	Statische Methoden	41
3.10	Klassen ableiten (Vererbung).	42
3.10.1	Ableiten mit einem Konstruktor, der nicht parameterlos ist	44
3.10.2	Ableiten von Klassen der .NET-Klassenbibliothek und der PowerShell-Bibliotheken	45
3.10.3	Members überschreiben.	45
3.11	Typenformatierung mit eigenen Klassen	49
3.12	Zusammenfassung	51
4	Functions für Fortgeschrittene	53
4.1	Was macht eine Function „Advanced“?	53
4.1.1	Die Rolle der Attribute.	53
4.1.2	Das Prinzip der Parameter-Zuordnung	54
4.2	Das CmdletBinding-Attribut	55
4.3	Das Parameter-Attribut	56
4.4	Die Parameterbindung im Detail	57
4.4.1	Wenn die Parameterbindung nicht funktioniert	58
4.4.2	Die Parameterbindung sichtbar machen	59
4.5	Functions mit Pipeline-Parametern	59
4.6	Functions, die die Pipeline abarbeiten	61
4.7	Functions mit einer eingebauten Bestätigungsanforderung.	62
4.7.1	Die Auswirkung des Confirm-Parameters	63
4.7.2	Die Auswirkung des WhatIf-Parameters	63

4.7.3	Implementieren von Confirm und WhatIf an einem Beispiel	63
4.7.4	Eine Bestätigungsanforderung unabhängig von Confirm & Co . . .	66
4.8	Parameter-Validierung	67
4.8.1	Einen Bereich validieren mit ValidateRange	67
4.8.2	Eine Auswahlmenge validieren mit ValidateSet	67
4.8.3	Ein beliebiges Kriterium validieren per ValidateScript	67
4.8.4	Credential-Parameter implementieren	68
4.8.5	Eigene Parameterattribute definieren	68
4.9	Dynamische Parameter	69
4.10	Zusammenfassung	71
5	Aus Functions und Skripte werden Module	73
5.1	Module und Modultypen	73
5.1.1	Skriptmodule	74
5.1.2	Manifestmodule	74
5.1.3	Binärmodule	74
5.1.4	Dynamische Module	75
5.1.5	Weitere Modulverzeichnisse hinzufügen	76
5.2	Manifestmodule im Detail	76
5.3	Verschachtelte Module	78
5.4	Mehrere Versionen eines Moduls verwenden	78
5.5	Internationale Module – Zeichenketten in Psd1-Dateien auslagern	79
5.5.1	Die Rolle der Kultur und das CultureInfo-Objekt	81
5.5.2	Ändern der aktuellen Kultur	82
5.6	Praxisteil: Erstellen eines Manifestmoduls	83
5.7	Zusammenfassung	85
6	PowerShell-Skripte testen mit Pester	87
6.1	Was testet Pester?	87
6.1.1	Testen und DevOps	88
6.2	Das Pester-Modul im Überblick	88
6.3	Die ersten Schritte mit Pester	89
6.4	Einen Test-Rahmen mit New-Fixture anlegen	92
6.5	Test-Ergebnisse vergleichen mit Should – die Rolle der Assertions	93
6.6	PowerShell-Commands nachbilden über Mocks	95
6.6.1	Feststellen, ob ein Mock-Command ausgeführt wurde	97
6.7	Tests in einen Ablauf einbeziehen	97
6.8	Testen als (Lebens-)Philosophie	97
6.9	Zusammenfassung	98
7	Skripte und Module bereitstellen	99
7.1	Die PowerShell-Paketverwaltung im Überblick	99
7.1.1	Ein Blick hinter die Kulissen	100
7.1.2	Überblick über das PackageManagement-Modul	103

7.1.3	Die ersten Schritte mit der Paketverwaltung	103
7.1.4	Anwendungspakete über Chocolatey installieren	103
7.1.5	Package-Provider offline installieren	107
7.1.6	Das PowerShellGet-Modul für die Modul- und Skriptverwaltung	107
7.1.7	Die ersten Schritte mit PowerShellGet	107
7.2	Eigene Ablagen für Module und Skripte einrichten	109
7.2.1	Skripte über GitHub als „Gists“ abrufen	112
7.2.2	Repository statt Webverzeichnis	112
7.2.3	Einrichten eines Modul-Repositorys mit MyGet.	113
7.2.4	Skripte und Module in der PowerShell Gallery veröffentlichen ...	115
7.3	Arbeiten mit einer Versionsverwaltung	117
7.3.1	Schritt für Schritt	118
7.3.2	Git-Integration in Visual Studio Code	126
7.4	Aufsetzen einer Release-Pipeline für Module	129
7.4.1	Was genau ist eine Release-Pipeline?	129
7.4.2	Wo gibt es die Release-Pipeline?	130
7.4.3	Eine Release-Pipeline selber gebaut	131
7.4.4	Eine Release-Pipeline mit AppVeyor	134
7.4.5	Die ersten Schritte mit AppVeyor	135
7.4.6	Die Anatomie der Yaml-Datei	136
7.4.7	Ein PowerShell-Modul per AppVeyor bereitstellen	137
7.5	Zusammenfassung	139
8	DSC-Grundlagen	141
8.1	Ein erstes Beispiel	141
8.2	Ein wenig Theorie	148
8.2.1	MOF	148
8.2.2	DSC-Spracherweiterungen	149
8.2.3	Die Rolle der Ressourcen	149
8.2.4	Der Local Configuration Manager (LCM)	150
8.2.5	Pull statt Push	151
8.3	Verwenden von Konfigurationsdaten	152
8.4	Warum DSC?	154
8.5	Zusammenfassung	154
9	DSC in der Praxis	157
9.1	Auspacken von Zip-Dateien	157
9.2	Umgebungsvariablen anlegen	158
9.3	Dateien und Verzeichnisse anlegen	160
9.4	Lokale Benutzer und Gruppen anlegen	161
9.5	Registry-Schlüssel anlegen	162

9.6	Windows-Feature installieren	164
9.7	Prozesse starten	164
9.8	Systemdienste einrichten	165
9.9	DSC-Log-Meldungen schreiben	166
9.10	Beliebige Befehle ausführen	168
9.11	Einen Webserver einrichten	169
9.12	Eine Hyper-VM einrichten	173
9.13	Zusammenfassung	174
10	DSC für (etwas) Fortgeschrittene	175
10.1	Hinzufügen von DSC-Ressourcen	175
10.2	Ressourcenabhängigkeiten festlegen	176
10.3	Den LCM konfigurieren	177
10.4	Umgang mit Konfigurationsdaten	178
10.4.1	Konfigurationsdaten unterschiedliche Nodes zuordnen	179
10.4.2	Einzelne Nodes auswählen	180
10.4.3	Allgemeine Eigenschaften in den Konfigurationsdaten festlegen ..	181
10.4.4	Konfigurationsdaten, die nur allgemeine Einstellungen enthalten. .	182
10.4.5	Konfigurationsdaten mit strukturierten Werten	183
10.5	Kennwörter in einer Konfiguration verwenden	184
10.6	Einrichten eines Pull Servers	189
10.6.1	Überblick über das Einrichten eines Pull Servers	190
10.6.2	Einrichten eines webbasierten Pull Servers	190
10.6.3	Umstellen des LCM auf den Pull-Modus	195
10.6.4	Einen Pull Server testen	196
10.6.5	Die Rolle der Reportserver	197
10.7	DSC-Diagnose	197
10.8	Eigene Ressourcen definieren	199
10.8.1	Zusammengesetzte Ressourcen (Composite Resources)	203
10.9	Die PowerShell DSC-Cmdlets im Überblick	206
10.10	Zusammenfassung	208
11	Aus Text Objekte machen	209
11.1	Texte im CSV-Format konvertieren	210
11.2	Überschriften nachträglich hinzufügen oder vorhandene Überschriften ändern	211
11.3	Unregelmäßige Texte zerlegen	211
11.3.1	Texte mit dem Split-Operator zerlegen	211
11.3.2	Texte mit Hilfe regulärer Ausdrücke zerlegen	213
11.4	Objekte anlegen	215
11.4.1	Objekte mit dem New-Object-Cmdlet anlegen	215
11.4.2	Objekte mit einer Hashtable anlegen	216

11.4.3	Unregelmäßige Textdaten mit dem ConvertFrom-String-Cmdlet verarbeiten	216
11.4.4	XML-Daten verarbeiten	217
11.5	JSON-Daten verarbeiten	220
11.6	Zusammenfassung	221
12	Active Directory-Administration	223
12.1	AD DS und Domänencontroller einrichten	224
12.1.1	Aktualisieren der Hilfe	225
12.1.2	Authentifizierung	225
12.2	Suche nach Benutzerkonten per Get-AdUser	226
12.3	Die PowerShell-Abfragesyntax	226
12.3.1	Benutzerkonten auswählen über den Identity-Parameter	226
12.3.2	Die Rolle des Properties-Parameter bei Get-ADUser	227
12.3.3	Spezialfall zusammengesetzte Attribute	228
12.3.4	Eingrenzen der Suche	228
12.3.5	Suchen nach anderen AD-Objekten	228
12.4	Benutzer anlegen per New-AdUser	229
12.4.1	Benutzerkonten aktivieren	230
12.4.2	Benutzerkonten über eine CSV-Datei anlegen	230
12.5	Benutzerkonten ändern per Set-ADUser	230
12.5.1	Umgang mit Mehrwert-Attributen	231
12.6	Benutzerkonten löschen mit Remove-ADUser	231
12.7	Nach Kontenattributen suchen	232
12.8	Gruppenzugehörigkeiten verwalten	232
12.8.1	Nicht alles passt zusammen	233
12.9	Computerkonten abfragen per Get-AdComputer	234
12.10	Abfrageergebnisse mit Out-GridView kombinieren	234
12.11	Umgang mit Organisationseinheiten	235
12.12	Das AD-Laufwerk	235
12.13	Den AD-Papierkorb aktivieren	236
12.14	Einen AD LSD oder Open LDAP-Server ansprechen	237
12.15	Zusammenfassung	238
13	Azure-Administration per PowerShell	239
13.1	Ein erster Überblick	239
13.2	Der ARM im Überblick	240
13.2.1	Die Rolle der Resource Provider	240
13.2.2	Die Rolle der Templates	241
13.3	Zugriffssteuerung per BPAC	248
13.4	Die Azure PowerShell im Überblick	249
13.4.1	Abfragen der PowerShell-Version	249
13.4.2	Ein erster Überblick	250

13.4.3	Die ersten Schritte mit der Azure PowerShell	250
13.4.4	Beispiele aus der Praxis	252
13.4.5	Eine virtuelle Maschine über ein Template anlegen	257
13.4.6	Azure und DSC	258
13.5	Custom Script Extension als Alternative zu DSC	262
13.6	Zusammenfassung	265
14	Debugging für etwas Fortgeschrittene	267
14.1	Unsichtbare Variablen beim Debuggen	267
14.2	Der Debug-Modus im Überblick	268
14.3	Über das Wesen eines Haltepunktes	268
14.3.1	Allgemeine Haltepunkte setzen	268
14.3.2	Haltepunkt entfernen und deaktivieren	269
14.3.3	Haltepunkte für einen Befehl setzen	269
14.3.4	Haltepunkte für eine Variable setzen	269
14.3.5	Haltepunkte mit einer Bedingung verknüpfen	270
14.3.6	Ein Skript ohne Haltepunkte debuggen	270
14.4	Remote-Debugging von Skripten	270
14.4.1	Haltepunkte über Invoke-Command setzen	271
14.5	Einen Workflow debuggen	271
14.6	Runspaces debuggen	271
14.7	Zusammenfassung	273
15	Sicherheit	275
15.1	Die Rolle der Ausführungsrichtlinie	275
15.2	Umgang mit Credentials	276
15.2.1	Einen Secure String lesbar machen	278
15.2.2	Einen Secure String anlegen	279
15.2.3	Secure String-Dateien sicher speichern	279
15.3	Umgang mit Zertifikaten	280
15.4	Weiterer Umgang mit Zertifikaten	281
15.4.1	Auflisten bestimmter Zertifikate	281
15.5	Zertifikate anlegen	281
15.5.1	Selbstsignierte Zertifikate erstellen	282
15.6	Skripte signieren	285
15.7	Zeichenketten verschlüsseln	286
15.7.1	Texte verschlüsseln und entschlüsseln mit CipherNet	286
15.7.2	Zeichenketten mit Zertifikaten verschlüsseln	287
15.8	Umgang mit Zugriffsberechtigungen	288
15.8.1	Entfernen von Zugriffsberechtigungen	290
15.9	Die Befehlsausführung protokollieren	291
15.9.1	Befehlsprotokollierung per Gruppenrichtlinien steuern	291
15.9.2	Mehr Möglichkeiten bei Start-Transcript	293

15.10	PowerShell-Remoting-Endpunkte sichern mit Just Enough Administration (JEA)	294
15.10.1	JEA in der Praxis	295
15.11	Eine Session ohne Remoting einschränken	298
15.12	Das Invoke-Expression-Cmdlet und warum es potentiell gefährlich ist . . .	299
15.12.1	Invoke-Expression per Scriptblock-Logging überwachen	300
15.13	Zusammenfassung	301
16	PowerShell für Linux	303
16.1	Ein erster Überblick	303
16.2	PowerShell unter Linux installieren	304
16.3	Die ersten Schritte unter Linux	305
16.4	Navigieren im Dateisystem	305
16.5	PowerShell-Remoting mit SSH	305
16.6	PowerShell versus PowerShell Core	307
16.7	Zusammenfassung	307
17	Wie man gute Skripte schreibt	309
17.1	Kommentare	309
17.2	Variableninitialisierung erzwingen	309
17.3	Aliase vermeiden	310
17.4	Datentypen für Parameter	310
17.5	Keine „Spuren“ hinterlassen	310
17.6	Der PSScriptAnalyzer	311
17.6.1	Eigene Regeln definieren	311
17.7	Zusammenfassung	313
18	Spaß mit der PowerShell	315
18.1	Zufallszahlen	315
18.2	Farbige Ausgaben	317
18.2.1	Farbe in der Konsole dank VT100-Unterstützung	318
18.3	Ein etwas anderer Prompt	320
18.3.1	Ein farbiger Prompt	321
18.4	Sounddateien abspielen	322
18.4.1	Systemsonds abspielen	322
18.4.2	Töne erzeugen	323
18.4.3	PowerShell-Musik	325
18.5	Die PowerShell lernt sprechen	325
18.5.1	Die .NET-Laufzeit kann auch sprechen	326
18.6	ASCII-Art und die 80er-Jahre	327
18.6.1	Bewegte ASCII-Art	327
18.7	Ein Zitat, bitte	328
18.7.1	Einen Internet-Zeitserver abfragen	329

18.8	Ein Matrix-Style-Bildschirmschoner	330
18.9	HAL ist IBM – der unwiderlegbare Beweis	330
18.10	April, April.	330
18.11	Ein Spielhallenklassiker.	332
18.12	Zusammenfassung	332
19	PowerShell für Entwickler	333
19.1	Unterschiede und Gemeinsamkeiten mit C#	333
19.2	Umgang mit Assemblys	334
19.3	Assemblys in eine PowerShell-Sitzung laden	335
19.3.1	Assemblys über ihren Pfad laden	336
19.3.2	Assemblys über ihren Namen laden	336
19.3.3	Alle geladenen Assemblys auflisten	337
19.3.4	Klassendefinitionen sichtbar machen	338
19.3.5	Den Inhalt einer Assembly sichtbar machen.	338
19.4	Assemblys erstellen	340
19.4.1	Herunterladen von NuGet-Packages	341
19.5	Umgang mit generischen Typen.	341
19.5.1	Generische Listen	342
19.5.2	Generische Methodenaufrufe	342
19.5.3	Aufruf einer generischen privaten Methode	343
19.6	Umgang mit Events	344
19.7	Das erweiterbare Typensystem	346
19.8	Cmdlets definieren	347
19.9	Benutzeroberflächen mit WPF	349
19.10	Win32-API-Funktionen aufrufen	353
19.11	Zusammenfassung	354
	Glossar	355
	Stichwortverzeichnis	359