O'REILLY®

4. Auflage

Windows PowerShell 5

kurz & gut

O'REILLYS TASCHENBIBLIOTHEK

Rolf Masuch
Thorsten Butz

4. AUFLAGE

Windows PowerShell 5

kurz & gut

Rolf Masuch und Thorsten Butz

O'REILLY®

Rolf Masuch und Thorsten Butz

Lektorat: Alexandra Follenius
Korrektorat: Claudia Lötschert
Herstellung: Susanne Bröckelmann
Umschlaggestaltung: Michael Oréal, www.oreal.de
Satz: III-Satz, www.drei-satz.de
Druck und Bindung: Media-Print Informationstechnologie,
www.mediaprint-druckerei.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 978-3-96009-027-4
PDF 978-3-96010-060-7
ePub 978-3-96010-061-4
mobi 978-3-96010-062-1

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.

4. Auflage

Wieblinger Weg 17
69123 Heidelberg

5 4 3 2 1 0

Inhalt

Windows PowerShell 5 – kurz & gut

Die Windows PowerShell eröffnet Administratoren der Windows-Plattform grundlegend neue Möglichkeiten. Mit der objektbasierten Befehlsshell, der Skriptsprache und zahlreichen Erweiterungen gehen Ihnen vormals mühselige Aufgaben jetzt mit Leichtigkeit von der Hand. Was als Werkzeug zur Automation begann, ist mittlerweile in einigen Fällen die einzige Option zur Konfiguration einer Software oder eines Diensts in Microsofts stetig wachsendem Portfolio. Die PowerShell entwickelt sich langsam aber stetig zur primären Arbeitsumgebung in den Rechenzentren und begründet einen neuen Typus des IT-Spezialisten: Die »DevOps-Bewegung« ist geboren, wie Jeffrey Snover, Vater der PowerShell, die Verschmelzung aus Entwicklung (»Development«) und Administration (»Operations«) gern nennt.

Ein Großteil der Leistungsfähigkeit der PowerShell beruht darauf, dass Sie Zugriff auf leistungsstarke Technologien erhalten: eine einprägsame Skriptsprache, reguläre Ausdrücke, das .NET Framework, Windows Management Infrastructure (MI), Component Object Model (COM), die Registry, integrierte Fernabfragefunktionen und nicht zuletzt Schnittstellen von Softwareherstellern wie VMWare oder Citrix, die schon frühzeitig PowerShell-Erweiterungen in die eigenen Produkte integriert haben.

Dieses Buch führt Sie in die Grundlagen der Befehlsshell und Skriptsprache ein, gibt Ihnen einen Überblick über den Leistungsumfang der verschiedenen PowerShell-Versionen und korrespondierenden Betriebssysteme und bietet eine übersichtliche Referenz über die wichtigsten Aufgaben.

Einleitung

Seit der Veröffentlichung der Windows PowerShell im Jahre 2006 müssen Windows-Administratoren nicht mehr neidisch zu ihren Unix-Kollegen hinüber schielen, wenn diese von der Kommandozeile aus komplexe Administrationsaufgaben über ihre Befehlsshell erledigen. Die PowerShell bietet alles, was von einer modernen Shell erwartet wird, und geht dabei noch einen Schritt weiter.

Anders als bei anderen Shells handelt es sich bei den Ergebnissen der einzelnen Befehle der PowerShell nicht um Text, sondern um Objekte, die Methoden und Eigenschaften besitzen. Die PowerShell arbeitet somit vollständig objektorientiert. Die Arbeit mit Objekten bietet gegenüber den herkömmlichen Shells einen entscheidenden Vorteil: Die Ergebnisse müssen nicht mehr mit einem Textparser durchsucht und aufbereitet werden – ein Vorgang, der gerade bei komplexen Ergebnissen fehleranfällig und umständlich ist.

Versionen und Updatemöglichkeiten

Die erste Version der PowerShell wurde in 2006 als Update für Windows XP, Windows Server 2003 (R2) und Windows Vista veröffentlicht. Seit Windows Server 2008 ist sie integrierter Bestandteil des Betriebssystems Windows. Mit jeder neuen Betriebssystemgeneration veröffentlichte Microsoft seither eine neue Revision der PowerShell, mal mit zahlreichen spektakulären Neuerungen, mal mit granularen Verbesserungen.

TIPP

Die PowerShell baut wesentlich auf dem .NET Framework auf:

PowerShell 1 und 2 erfordern (mindestens) .NET Framework 2.
PowerShell 3 erfordert (mindestens) .NET Framework 4.
PowerShell 4 und 5 erfordern (mindestens) .NET Framework 4.5.
PowerShell 5.1 erfordert (mindestens) .NET Framework 4.6.

Die hieraus resultierenden Versionsabhängigkeiten können zu Inkompatibilitäten mit der von Ihnen verwendeten Software führen. Einige Features der PowerShell können auch eine höhere Version des .NET Frameworks benötigen als die zugrunde liegende PowerShell-Version.

In vielen Fällen kann die PowerShell auf eine neuere Version aktualisiert werden, wobei die Aktualisierung über das Windows Management Framework (WMF) erfolgt, sodass neben der PowerShell weitere Verwaltungskomponenten (WinRM, WMI etc.) auf den jeweils aktuellen Stand gebracht werden.

Tabelle 1: Übersicht der Betriebssysteme mit den jeweiligen PowerShell-Versionen

Betriebssystem	1	2	3	4	5
WS 2008	Integriert	Update	Update		
Windows 7		Integriert	Update	Update	Update
WS 2008 R2		Integriert	Update	Update	Update
Windows 8			Integriert		
WS 2012			Integriert	Update	Update
Windows 8.1				Integriert	Update
WS 2012 R2				Integriert	Update
Windows 10					Integriert
WS 2016					Integriert

Mit der Veröffentlichung von Windows 10 (Build 10240) im Juli 2015 erschien die PowerShell in der Version 5. Dem neuen Paradigma »Windows as a Service« folgend erhielt (und erhält) das Betriebssystem Windows 10 kontinuierliche Aktualisierungen, sodass diese ursprüngliche Version der PowerShell 5 nicht im klassischen Sinn als RTM-Version (»release to manufacturing«) bezeichnet werden kann.

Mit dem sogenannten Windows 10 »November-Update« (Version 1511, Build 10586) gab das PowerShell-Team dem Windows Management Framework (WMF) 5 RTM-Status und bot Updatepakete für ältere Betriebssysteme an. Dieses Update musste aufgrund eines Fehlers in der Installationsroutine jedoch noch einmal zurückgerufen werden. Ende Februar 2016 fand das Update dann seinen Weg zurück in das Microsoft-Download-Center.

Windows 8 kann kostenfrei auf Windows 8.1 aktualisiert werden, ein Update des WMF für Windows 8 wird nicht angeboten.

Mit dem Erscheinen des Windows 10 »Anniversary Updates« (Version 1607) veröffentlichte Microsoft die PowerShell in der Version 5.1 und unterschied erstmalig zwischen einer »Core« und einer »Desktop« Edition. Darüber hinaus kündigte Microsoft an, die PowerShell für alternative Betriebssysteme zu portieren. Mehr dazu lesen Sie im Abschnitt »Die Zukunft: Powershell Core vs. Desktop Edition« auf Seite 10.

Funktionsumfang der PowerShell

Es mag hilfreich sein, sich die PowerShell als Werkbank vorzustellen. An diesem speziellen Arbeitsplatz, der »Shell«, liegen einige Basiswerkzeuge, die »Operatoren«, bereit, die jedoch ohne Werkstoffe, die »Objekte«, wenig Nutzen entfalten. Das Betriebssystem und die installierten Applikationen (einschließlich Anwendungsservern wie Microsoft Exchange, Microsoft System Center etc.) liefern den Programmcode, den Sie schlussendlich mittels PowerShell-Befehlen, den »Cmdlets«, oder eigenen Funktionen ausführen.

Computer mit gleicher PowerShell-Version sind demnach nicht zwingend funktionsgleich.

TIPP

Ein Beispiel: Seit Windows 8 listet Ihnen das Cmdlet *Get-NetAdapter* die Netzwerkschnittstellen des Systems auf. Der Befehl greift dabei auf die (W)MI-Klasse `MSFT_NetAdapter` im Namespace *root/StandardCimv2* zurück.

In Windows 7 ist die Klasse nicht implementiert, der Befehl *Get-NetAdapter* steht Ihnen unabhängig von der installierten PowerShell-Version nicht zur Verfügung.

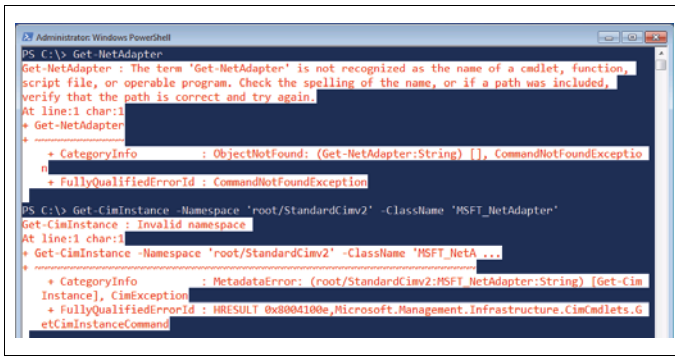

Abbildung 1: »Get-NetAdapter nicht verfügbar (Windows 7)«

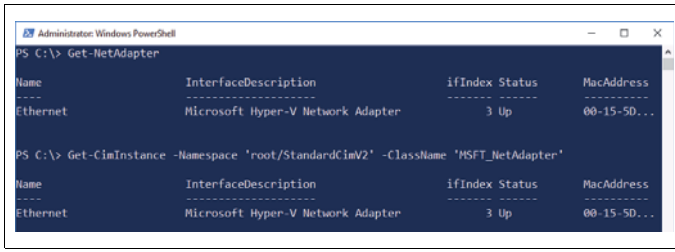

Abbildung 2: »Get-NetAdapter verfügbar (Windows 10)«

Windows-Server-Editionen enthalten zahlreiche Programme, »Rollen und Features« genannt, die nicht zum Funktionsumfang von Windows Clients gehören. Durch die Installation der »Remote Server Administration Tools« (RSAT) ergänzen Sie nicht nur die zur Verfügung stehenden grafischen Verwaltungswerkzeuge, sondern Sie erweitern auch den Funktionsumfang der PowerShell (mehr erfahren Sie im Kapitel *Die PowerShell erweitern* auf Seite 114). Beachten Sie, dass die aktuelle Version der RSAT in der Regel nur für die aktuelle Windows Client-Version zur Verfügung steht. Analog stehen Ihnen bei vielen Applikationsservern eigenständige Installationspakete zur Installation auf dem Client zur Verfügung.

Als Konsequenz sollten Sie zur Administration Ihrer IT stets das aktuellste Windows nutzen, zur Drucklegung dieses Buchs ist dies

Windows 10. Selbst wenn Sie mittels Fernaufrufen ältere Geräte administrieren, profitieren Sie auf Ihrem Administrations-PC vom erweiterten Funktionsumfang der RSAT sowie von den Verbesserungen der Plattform wie z.B. einer leistungsstarken Entwicklungsumgebung, dem »Integrated Scripting Environment« (ISE), oder der Syntaxhervorhebung in der PowerShell-Konsole.

Die Evolution der PowerShell

PowerShell 1

Die erste Version der PowerShell erschien 2006 als Updatepaket und wurde in Windows Server 2008 als optionales Feature integriert.

Die Version 1 kennt einen PowerShell-Host: powershell.exe, auch PowerShell-Konsole genannt. Die 129 Cmdlets bieten nur in wenigen Fällen eine integrierte Fernabfragefunktion.

Zu den Ausnahmen gehört jedoch das wichtige *Get-WMIObject*-Cmdlet, das den Zugriff auf die WMI-Schnittstelle ermöglicht und die PowerShell schon in ihrer ersten Version zu einem sehr nützlichen Alltagsbegleiter werden ließ.

```
Get-WmiObject -Class Win32_LogicalDisk -ComputerName sv1
```

Darüber hinaus ist der direkte Rückgriff auf die Objekte des .NET Frameworks möglich.

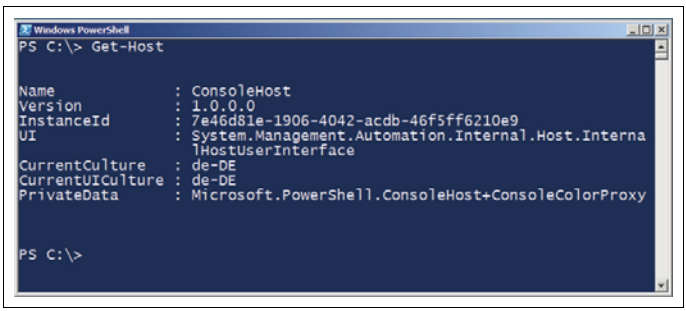

Abbildung 3: Die PowerShell 1 in Windows Server 2008

PowerShell 2

Die zweite Version ist integrierter Bestandteil von Windows 7 und Windows Server 2008 R2.

Die Anzahl der Basis-Cmdlets erhöht sich auf 236, zahlreiche Cmdlets bieten nun eine integrierte Fernabfragefunktion. Darüber hinaus verfügt die Version 2 unter der Bezeichnung »PowerShell Remoting« über eine allgemeine Fernabfragefunktion auf Grundlage des modernen »WS-Managements« (WS-Man), einem standardisierten Netzwerkprotokoll auf Basis von Webservices und dem »Simple Object Access Protocol« (SOAP). Mit dem Cmdlet *Invoke-Command* können nun beliebige Befehle auf entfernten Geräten ausgeführt werden, die einen WS-Man-Server bereitstellen.

Als Erweiterungsoption unterstützt die PowerShell nun Module als Alternative zu den PowerShell Snap-ins der ersten Version.

Erstmals steht neben der PowerShell-Konsole ein zweiter Host zur Verfügung: das »Integrated Scripting Environment« (ISE).

PowerShell 3

Die dritte Version, Bestandteil von Windows 8 und Windows Server 2012, wartet mit zahlreichen Neuerungen auf. Auffällig sind die runderneuerte ISE, das Autoloading von Modulen und die aktualisierbare Hilfe (»Updatable Help«).

Die Zahl der integrierten Befehle steigt sprunghaft auf 1.157 (440 Cmdlets, 717 Funktionen), was wesentlich der CDXML-Technologie (»Cmdlet Definition XML«) zu verdanken ist, mit der Cmdlets auf Grundlage von WMI-Klassen weitgehend automatisiert erzeugt werden.

Mit den CIM-Cmdlets (»Common Information Model«) führt Microsoft seine Bemühungen fort, eine standardisierte Verwaltungsschnittstelle zu schaffen, die Fernzugriffe über das moderne WS-Man-Protokoll nutzt. Die vertraute WMI-Technologie, die für den Fernzugriff DCOM (»Distributed COM«) nutzt, bleibt zwar erhalten, wird aber perspektivisch durch ihre modernen Pendants ersetzt.

TIPP

»Standards Based Management«: Unter dem Dach der »Distributed Management Task Force« (DMTF) fördert Microsoft die Entwicklung von Standards zur Verwaltung heterogener IT-Landschaften. So stellt Microsoft einen quelloffenen CIM/WS-Man-Server namens »OMI« für unixoide Betriebssysteme bereit, der den Netzwerkzugriff zwischen Unix- und Windows-Hosts ermöglicht. Im Kontext dieser Entwicklung veröffentlichte Microsoft auch DSC-Erweiterungen für einige Linux-Derivate.

(Mehr zum Thema DSC erfahren Sie im Abschnitt »Desired State Configuration (DSC)« auf Seite 101.)

Weiterführende Informationen erhalten Sie auf diesen Webseiten:

https://collaboration.opengroup.org/omi
https://blogs.technet.microsoft.com/windowsserver/2012/06/28/open-management-infrastructure/

Mit der vereinfachten Syntax kann ab PowerShell 3 in einigen Fällen auf geschweifte Klammern und die »Special Pipeline Variable« *$_* verzichtet werden:

Beispiel 1: klassische Schreibweise (Arraysyntax)

```
Get-Process | Where-Object { $_.Handles -gt 500 }
```

Beispiel 2: vereinfachte Schreibweise (Simplified Syntax)

```
Get-Process | Where Handles -gt 500.
```

PowerShell 4

Mit der Veröffentlichung der vierten PowerShell-Version in Windows 8.1 und Windows Server 2012 R2 standen die Systempflege und Fehlerbereinigung im Mittelpunkt.

Die bedeutsamste Neuerung repräsentiert die »Desired State Configuration« (DSC), mit deren deskriptivem Ansatz Rechenzentren orchestriert werden sollen. Mit dem Schlüsselwort *Configuration* wird in einem PowerShell-Skript ein Sollzustand beschrieben, bei dessen Aufruf eine MOF-Datei erzeugt wird, die mittels *Start-*

DscConfiguration auf einem oder mehreren Computern angewendet wird.

Da DSC im Kern keinerlei Fähigkeiten zur Konfiguration eines Betriebssystems oder einer Applikation bietet, sind sogenannte DSC-Ressourcen erforderlich, die die gewünschte Konfiguration ausführen. Die DSC-Ressource kann über ein Modul zur Verfügung gestellt werden und muss selbst keinen PowerShell-Code enthalten. DSC-Ressourcen stehen auch für Nicht-Windows-Systeme zur Verfügung, wobei der Programmcode zur Konfiguration eines Linux-Hosts beispielsweise in Python geschrieben sein kann.

PowerShell 5

Mit Windows 10 und Windows Server 2016 liefert Microsoft die fünfte Version der PowerShell aus, die zahlreiche spannende Neuerungen enthält.

Die Schlüsselworte *Class* und *Enum* erlauben erstmalig das Erstellen von eigenen Klassen (»Class«) unter Verwendung eigener Datentypen (»Enum«), was das Erstellen eigener DSC-Ressourcen vereinfacht und die PowerShell syntaktisch näher an objektorientiere Hochsprachen wie C# rücken lässt.

Wie üblich enthält die aktualisierte Version eine Reihe neuer Cmdlets, unter denen *ConvertFrom-String* und *Convert-String* hervorstechen. Die Cmdlets konvertieren Zeichenketten auf Grundlage von Beispielen, die der Anwender vorgibt. Die Befehle basieren auf regulären Ausdrücken, befreien den Administrator jedoch davon, diese oftmals schwer zu definierenden Suchmuster selbst zu erstellen.

Die neuen Module *PowerShellGet* und *PackageManagement* ermöglichen das Nachladen und Installieren von Software. So erhält man über Find-Module und Install-Module (Modul *PowerShellGet*) auf einfache Weise Zugriff auf die PowerShell Gallery und weitere, selbst zu konfigurierende, Repositories. Mit den Cmdlets *Find-Package* und *Install-Package* installiert man analog Softwarepakete. Beide Module erfordern einen NuGet-Provider (basierend auf *nuget.exe*), der bei der ersten Verwendung heruntergeladen werden kann.

TIPP

Die PowerShell Gallery ist eine von Microsoft betriebene Plattform zum Bereitstellen von Erweiterungen für die Windows PowerShell (*https://www.powershellgallery.com*). Die dort angebotenen Lösungen werden auf Schadsoftware geprüft, bevor sie öffentlich zugänglich gemacht werden. Nach einer längeren Preview-Phase ist sie seit Februar 2016 offiziell freigegeben.

Mit »Just Enough Admin« (JEA) erweitert Microsoft in PowerShell 5 die Technologie der »Constrained Endpoints« um ein Rollenmodell zur (Fern-)Administration mittels virtueller Administratorenkonten und spezifischer Endpunkte. Startet ein nicht privilegierter Anwender eine (Remote-)Session, können ihm spezifische, weiterführende Rechte zugewiesen werden.

Die vielleicht auffälligste Neuerung ist im Kern keine Neuerung der PowerShell 5, sondern des zugrunde liegenden Betriebssystems: Mit Windows 10 und Windows Server 2016 stellt Microsoft erstmals seit Jahren ein signifikantes Update der Konsolen-Applikation zur Verfügung. Sowohl in *cmd.exe* als auch in *powershell.exe* lässt sich die Größe der Konsole mit einfachem Ziehen verändern, wobei der angezeigte Text automatisch umgebrochen wird. Der Konsolenhintergrund lässt sich granular transparent schalten, und die systemweit üblichen Tastenkombinationen zum Kopieren und Einfügen (Strg + C, Strg + V) funktionieren nun endlich auch in der Konsole.

Die Zukunft: Powershell Core vs. Desktop Edition

Im Juni 2016 veröffentlichte Microsoft .NET Core 1.0 für Windows, Linux und macOS eine modulare Neuentwicklung, die eine Teilmenge des weiterhin verfügbaren »vollständigen« .NET Frameworks bereitstellt. Im August 2016 folgte die Veröffentlichung des Quelltextes der PowerShell unter der MIT Lizenz. Zeitgleich gab man bekannt, dass die PowerShell zukünftig auch für alternative Betriebssysteme (Linux, macOS) verfügbar sein wird. Basis dieser Implementierung ist die oben genannte plattformübergreifende .NET Core CLR (*Common Language Runtime*).

Windows Server 2016 erlaubt eine miniaturisierte Installation namens *Nano-Server*, die ebenfalls jene .NET Core CLR unterstützt, nicht aber das vollständige .NET Framework, das in allen anderen Installationsoptionen unter Windows integriert ist. Diese funktional beschränkten Spezialversionen der PowerShell bezeichnet Microsoft als *Core Edition*.

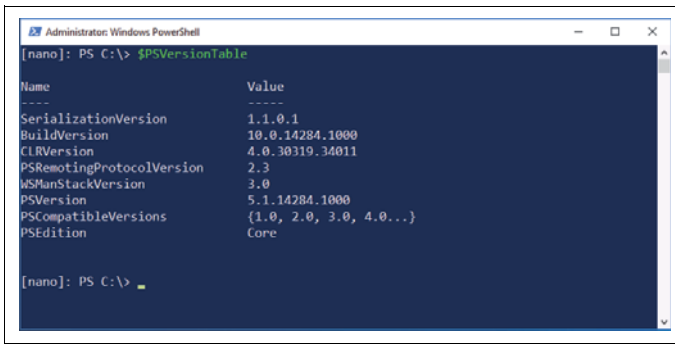

Abbildung 4: PowerShell 5.1 »Core Edition«

Mit dem Erscheinen von Windows 10 Version 1607, dem »Anniversary Update«, und Windows Server 2016 erhält die PowerShell erstmals eine Unterversion, die neben Fehlerkorrekturen auch eine Unterscheidung zwischen eben jener »Core Edition« und der vertrauten Variante, zukünftig »Desktop Edition« genannt, einführt: die PowerShell 5.1.

Das Update auf WMF 5.1 unterstützt die gleichen Betriebssysteme wie auch WMF 5.0.

Weiterführende Informationen über Microsofts OpenSource-Strategie und »PowerShell on Nano Server« finden Sie unter den nachfolgenden Links:

https://technet.microsoft.com/en-us/library/mt671124(v=ws.12).aspx
https://azure.microsoft.com/en-us/blog/powershell-is-open-sourced-and-is-available-on-linux/

Die PowerShell als Sprache

Die PowerShell kennt eine Reihe von ausführbaren Befehlen: Cmdlets (*Get-Process*), Aliasse (*ps*), Funktionen (*mkdir*), Ausdrücke (*2+2*), externe Programme (*ipconfig.exe*) sowie Dateinamen (*notiz.txt, skript.ps1*), wobei im letztgenannten Fall die verknüpfte Anwendung gestartet und das Dokument geladen wird, analog zu dem Verhalten in der klassischen Eingabeaufforderung.

HINWEIS

Die Dateinamenserweiterung **.ps1* lässt keinen Rückschluss auf die zugrunde liegende PowerShell-Version zu. Mit der Anweisung `requires` im Kopf einer Skriptdatei lassen sich Versionsanforderungen definieren.
Mehr dazu verrät *Get-Help about_Requires*.

Cmdlets

Das Herz der Skriptsprache PowerShell sind die sogenannten *Cmdlets* (gesprochen: »Commandlets«). Die Implementierung der Cmdlets erfolgt als spezialisierte .NET-Klassen oder als erweiterte Funktionen (»Advanced Functions«, auch »Script Cmdlets« genannt), die zur Laufzeit innerhalb der PowerShell instanziiert und aufgerufen werden. Eine Übersicht über die Ihnen zur Verfügung stehenden Cmdlets erhalten Sie mit *Get-Command*.

Cmdlets bestehen aus zwei Teilen, einem Verb und einem Substantiv, die durch einen Bindestrich miteinander verbunden sind. Diese Zusammensetzung der Cmdlets aus einer Verb-Substantiv-Syntax mag am Anfang etwas ungewohnt sein, aber Sie werden gerade zu Beginn Ihrer Arbeit schnell feststellen, dass diese strikte Syntax Ihnen dabei hilft, sich in die PowerShell und ihre Befehle einzuarbeiten. Eine Übersicht über die in der PowerShell zugelassenen Cmdlet-Verben finden Sie im Referenzteil dieses Buchs ab Seite 129.

```
Get-Process
Get-Process -name powershell
Get-ChildItem
Get-ChildItem -Path C:\Windows -filter *.xml -Recurse
```