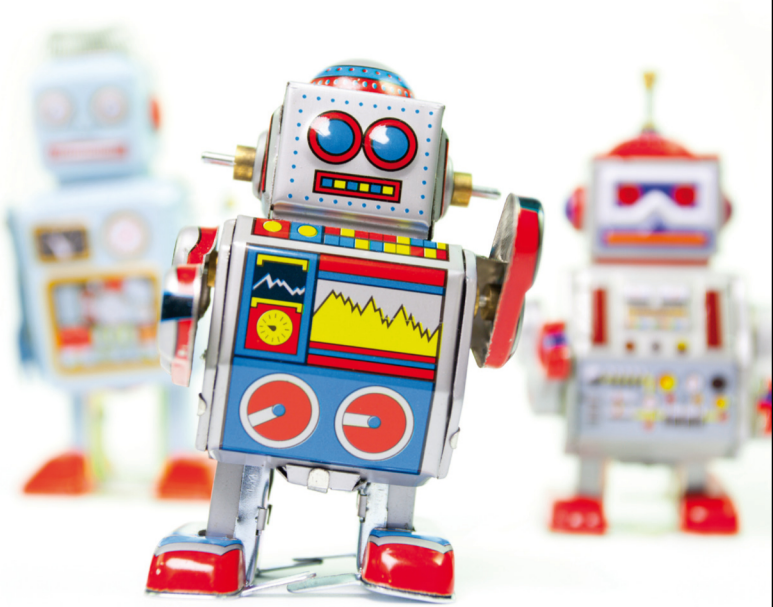
Mit vielen Beispielen aus der Praxis

PowerShell 7
und Windows PowerShell

Das komplette Praxiswissen für Administratoren und IT-Profis

Für Windows – Linux – macOS – Cloud

Dr. Tobias Weltner



O'REILLY®

Dr. Tobias Weltner

PowerShell 7 und Windows PowerShell

Das komplette Praxiswissen für Administratoren und IT-Profis

O'REILLY®

Dr. Tobias Weltner

Lektorat: Ariane Hesse
Lektoratsassistenz: Anja Weimer, Julia Griebel
Korrektorat: Sibylle Feldmann, *www.richtiger-text.de*
Satz: Gerhard Alfes, mediaService, *www.mediaservice.tv*
Herstellung: Stefanie Weidner
Umschlaggestaltung: Michael Oreal, *www.oreal.de*, unter Verwendung eines Fotos von iStock by Getty Images von querbeet, Stock-Fotografie-ID184997148
Druck und Bindung: mediaprint solutions GmbH, 33100 Paderborn

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 978-3-96009-163-9
PDF 978-3-96010-479-7
ePub 978-3-96010-480-3
mobi 978-3-96010-481-0

1. Auflage 2021

Wieblinger Weg 17
69123 Heidelberg

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.

Hinweis:
Dieses Buch wurde auf PEFC-zertifiziertem Papier aus nachhaltiger Waldwirtschaft gedruckt. Der Umwelt zuliebe verzichten wir zusätzlich auf die Einschweißfolie.

Schreiben Sie uns:
Falls Sie Anregungen, Wünsche und Kommentare haben, lassen Sie es uns wissen: *kommentar@oreilly.de*.

5 4 3 2 1 0

Inhalt

Vorwort

PowerShell begann vor 15 Jahren als Windows Powershell. Heute ist diese Shell plattformübergreifend auf Windows, Linux und macOS verfügbar und wird begeistert von einer immer größeren Anwendercommunity genutzt.

In diesem Buch werden Sie Schritt für Schritt erfahren, was PowerShell eigentlich ist und was diese Shell alles für Sie tun kann. Ob Sie Administrator »on-premise« sind und lokale Server betreuen, Ihr Unternehmen in der Cloud verwalten oder ob Sie heterogene Umgebungen »unter einen Hut« bringen wollen – PowerShell bietet die Werkzeuge dafür.

Aber auch zu Hause und in der Schule lässt es sich hervorragend einsetzen: PowerShell ist es nämlich egal, ob Sie damit eine Unternehmens-IT administrieren, Ihre Philips-Hue-Homeautomation verwalten oder Sonnenkollektoren steuern. Ob Sie den eigenen NAS-Server auf dem Dachboden sichern, automatisiert Dateien aus dem Internet laden oder vielleicht bloß Matherätsel knacken wollen.

Und genau deshalb ist PowerShell auch in Ausbildung und Schulen spannend: Kaum eine andere kostenlose Programmierumgebung und Shell unterstützt so viele Programmierkonzepte auf so vielen Plattformen – angefangen von Befehlsaufrufen über moderne Programmiertechniken wie den Einsatz von Funktionen und Objekten bis hin zu nativer Klassenunterstützung, Vererbung und universellem Netzwerk-Remoting. Kaum eine andere Umgebung bietet so unmittelbares Feedback wie PowerShell. Ideal also, um im Unterricht und in der Ausbildung anhand von nachvollziehbaren Praxisbeispielen die Funktionsweisen moderner IT auszuprobieren und zu vertiefen.

Dieses Buch bietet zahlreiche Praxisbeispiele aus den unterschiedlichsten Einsatzbereichen und demonstriert schrittweise die vielfältigen Möglichkeiten der PowerShell.

So schlüpfen Sie zu Anfang des Buches in die Rolle des einfachen PowerShell-Anwenders. Sie generieren mit Einzeilern Excel- und HTML-Reports und können mit wenigen Schritten Cloud-Systeme steuern oder Notenblätter aus dem Internet herunterladen und als PDF exportieren.

Von diesen ersten Fingerübungen inspiriert lernen Sie den Minimal-Wortschatz der PowerShell kennen und fügen mehrere Befehle zu eigenen größeren Automationslösungen zusammen.

Die Anwendungsbeispiele werden immer komplexer, und Sie lernen schrittweise alle weiteren wichtigen Konzepte der PowerShell kennen. Dazu gehören Remotezugriffe, direkte Betriebssystemaufrufe, das Erstellen grafischer Oberflächen sowie eigene PowerShell-Befehle und -Module.

Am Ende dieses Buches beherrschen Sie dann eine der modernsten Automationssprachen, die für Windows, Linux und macOS kostenfrei zur Verfügung stehen, und haben sich privat wie beruflich vom einfachen Anwender zum versierten IT-Automatisierer qualifiziert.

Damit Sie auf dieser Reise nicht allzuviel eintippen müssen, automatisiert PowerShell in diesem Buch auf Wunsch natürlich auch das Eintippen der Beispiele für Sie. Schon im ersten Kapitel lernen Sie den passenden Befehl kennen: durch Eingabe der jeweiligen Listingnummer fügt PowerShell den Quellcode aus dem Buch automatisch ein.

Wie Sie dieses Buch nutzen

Dieses Buch setzt keinerlei Grundkenntnisse voraus, wenn Sie von vorn zu lesen beginnen – und das ist auch empfehlenswert. Die Kapitel bauen aufeinander auf. Am Anfang jedes Kapitels finden Sie eine kurze Zusammenfassung, falls es einmal eilig ist.

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.

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

Falls trotz aller Sorgfalt einmal Fragen offenbleiben oder Sie weitere Ideen und Themenwünsche haben, besuchen Sie einfach das interaktive Leserforum zu diesem Buch: *https://github.com/TobiasPSP/OReilly/discussions*

Damit bleibt mir an dieser Stelle nur noch, Ihnen viel Spaß zu wünschen bei der Lektüre dieses Buchs und bei der Arbeit mit der faszinierenden PowerShell! Ich würde mich freuen, von Ihnen im Diskussionsforum zu hören.

Herzlichst,

Tobias Weltner

Kapitel 1
PowerShell: Erste Schritte

In diesem Kapitel:

Ausführlich werden in diesem Kapitel die folgenden Aspekte erläutert:

- **Windows PowerShell:** In Windows integrierte PowerShell, die auf dem klassischen *.NET Framework Version 4.5* oder höher basiert und auch künftig für Automationsaufgaben im Windows-Umfeld eingesetzt werden kann. Der Startbefehl lautet `powershell.exe`, und die aktuelle Version ist 5.1.
- **PowerShell:** Neuartige, plattformunabhängige PowerShell, die als portable Anwendung bei Windows parallel zur *Windows PowerShell* verwendet werden kann und auch auf Linux, macOS und weiteren Betriebssystemen zur Verfügung steht. Der Startbefehl lautet `pwsh.exe`. Diese PowerShell beruht auf dem plattformunabhängigen neuen *.NET Framework Core*, das weitgehend kompatibel zum klassischen .NET Framework ist. Die *PowerShell* ist im Gegensatz zur *Windows PowerShell* quelloffen (Open Source).

- **Autovervollständigung:** Mit einigen Tastendrücken kann man sich bei der Eingabe von Befehlen, Parametern und Argumenten Tipparbeit sparen: ⇆ vervollständigt Eingaben. Drücken Sie die Taste mehrmals, zeigt PowerShell bei jedem Drücken einen weiteren Vorschlag. ⇧+⇆ geht in der Reihenfolge wieder einen Schritt zurück. Mit Strg+Leertaste werden Eingabevorschläge als vollständige Auswahlliste präsentiert. Editoren zeigen dazu ein IntelliSense-Menü an. In der Konsole erscheint ein textbasiertes Auswahlfeld. Unabhängig von den Autovervollständigungsvarianten können Sie mit ↑ und ↓ frühere Eingaben aus Ihrer Befehlshistorie anzeigen lassen. Diese Liste ist anfangs leer und wächst mit der Verwendung der Konsole.
- **Zeilen löschen und Befehlsabbruch:** Um die gesamte aktuelle Zeile zu löschen, drücken Sie Esc. Um einen Befehl abzubrechen, drücken Sie Strg+C.
- **Groß- und Kleinschreibung:** PowerShell selbst unterscheidet bei Befehlsnamen und Parametern nicht zwischen Groß- und Kleinschreibung. Ob die Groß- und Kleinschreibung bei Befehlsargumenten (wie zum Beispiel Pfadnamen oder anderen Angaben) wichtig ist, hängt vom jeweiligen Befehl und dem verwendeten Betriebssystem ab. Bei Kennworteingaben beispielsweise kommt es natürlich immer auf die richtige Groß- und Kleinschreibung an.

Achtung

Wenn Sie es eilig haben und die Grundlagen der PowerShell schon kennen, dürfen Sie diesen Einleitungsteil selbstverständlich überspringen. Sie sollten aber in jedem Fall wenigstens den Abschnitt »Zusammenfassung« am Ende dieses Kapitels beachten. Dort werden wichtige Grundeinstellungen besprochen, die die Voraussetzung für viele Beispiele in den folgenden Kapiteln sind.

PowerShell ist eine verblüffend flexible und machtvolle plattformunabhängige Automationssprache, die mit geringem Aufwand ein enormes Spektrum von Aufgaben automatisieren kann.

Dazu zählen typische IT-Administrationsaufgaben ebenso wie völlig andere Einsatzbereiche aus Mathematik, Forschung und Lehre, in der Büroautomation und nicht zuletzt in Hobby und Tüftelei: Schon im nächsten Kapitel werden wir uns kurz der Musikkomposition und Steuerung von MIDI-Musikinstrumenten widmen, und in *Kapitel 13* erfahren Sie zum Beispiel, wie PowerShell sogar Lampen und Steckdosen in Ihrem Zuhause fernsteuert. Die Grundlagen und Strategien sind dabei indes immer dieselben.

Bei all diesen Beispielen geht es also ausschließlich um zweierlei: kurzweilig und verständlich möglichst viele Einsatzszenarien der PowerShell vorzustellen, um Ideen zu wecken und eine breite Leserschicht anzusprechen, und die stets gleichen allgemeinen Mechanismen zu verstehen, die dabei zum Einsatz kommen.

Denn mit dem Wissen, das Sie beispielsweise in der Musikkomposition oder auch bei der Fernsteuerung von Elektrogeräten benötigen, können Sie natürlich auch Server aufsetzen oder Drittanbietersoftware für Backup-Lösungen steuern – und umgekehrt. Hier wird deutlich, dass PowerShell ebenfalls eine ideale Plattform für Ausbildung und Schule ist, denn alle modernen IT-Grundlagen lassen sich damit anschaulich und unterhaltsam vermitteln.

Zunächst aber muss PowerShell vollständig eingerichtet werden, und genau darum geht es in diesem ersten Kapitel. Dabei werden auch einige wichtige Sicherheitseinstellungen besprochen, und der Unterschied zwischen *PowerShell* und *Windows PowerShell* wird Ihnen schnell klar werden.

Danach folgt ein kleiner Exkurs zu den wichtigsten kostenlosen PowerShell-Tools, mit denen Sie Ihren Computer in eine moderne PowerShell-Entwicklungsumgebung verwandeln.

Zum Abschluss des Kapitels erfahren Sie, wie die vielen Hundert Skriptbeispiele in diesem Buch ohne Tipparbeit direkt von PowerShell aus dem Internet heruntergeladen und ausgeführt werden und wie offengebliebene Fragen unmittelbar an den Autor gerichtet werden können.

Hinweis

In diesem Buch wird das Wort »PowerShell« immer dann kursiv gesetzt, wenn spezifisch die neue plattformübergreifende *PowerShell* gemeint ist. Der nicht kursiv gesetzte Begriff »PowerShell« bezieht sich allgemein auf die Sprache PowerShell und gilt sowohl für *Windows PowerShell* als auch für *PowerShell*.

Schon in diesem Kapitel wird Ihnen bereits PowerShell-Code begegnen, der aber hier als reines »Werkzeug« für die Einrichtung Ihres Computers dient. Sie können den Code natürlich bereits neugierig mustern, aber bitte brüten Sie nicht zu lange über einzelnen Codebeispielen. Es kommt in diesem Kapitel nur auf die Ergebnisse der Codebeispiele an, nicht auf den Code selbst und seine Mechanismen. Diese stehen ab dem zweiten Kapitel im Vordergrund und werden dort dann auch systematisch besprochen.

PowerShell installieren

PowerShell ist plattformunabhängig und kann auf verschiedenen Betriebssystemen ausgeführt werden, zum Beispiel Windows, Linux oder macOS.

Historisch ist PowerShell allerdings als Teil des Windows-Betriebssystems entstanden und dort als sogenannte *Windows PowerShell* immer automatisch enthalten. Seit 2016 ist diese Fassung in Version 5.1 fertiggestellt, wird auf diesem Stand gehalten und weiter von Microsoft gewartet.

Alle Entwicklungsarbeit geht seit 2016 in die neue, plattformunabhängige *PowerShell*, bei Drucklegung dieses Buchs in Version 7.2, die bei allen Betriebssystemen zuerst nachinstalliert werden muss. Wie das geschieht, erfahren Sie jetzt, und zwar in separaten Abschnitten für die jeweiligen Betriebssysteme.

Windows-Betriebssystem

In allen Windows-Betriebssystemen ab Server 2008 und Windows 7 ist die PowerShell bereits fester Bestandteil – genauer gesagt die *Windows PowerShell*. Hier könnten Sie also sofort loslegen, sollten aber wenigstens prüfen, ob Ihre *Windows PowerShell* auf dem aktuellen Stand ist (dazu gleich mehr).

Noch empfehlenswerter ist es, zusätzlich zur integrierten *Windows PowerShell* auch die neue *PowerShell* hinzuzuinstallieren. Beide laufen friedlich parallel nebeneinander, und so können Sie sich ein eigenes Bild von den Unterschieden machen.

Immer vorhanden: Windows PowerShell

Sie starten die *Windows PowerShell* mit dem Befehl `powershell.exe` (oder kurz `powershell`) – wie jedes andere Programm auch. Drücken Sie zum Beispiel ⊞+R, um das Dialogfeld *Ausführen* zu öffnen (siehe Abbildung 1.1), und geben den Befehl `powershell` ein. Dann klicken Sie auf *OK*.

In der Taskleiste erscheint ein blaues Symbol, das Sie per Rechtsklick und *An Taskleiste anheften* am besten dauerhaft dort anpinnen. Ein Rechtsklick auf das Symbol öffnet die sogenannte Sprungliste, sozusagen das Cockpit der *Windows PowerShell* (siehe Abbildung 1.2).

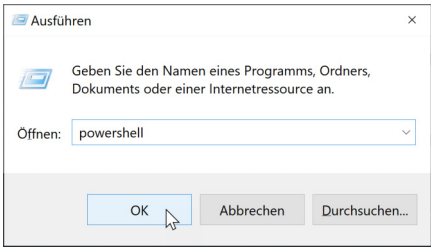


Abbildung 1.1: Windows PowerShell starten

Von hier aus können Sie die PowerShell zum Beispiel mit vollen Administratorrechten starten, und auch der in *Windows PowerShell* integrierte Editor *ISE* (*Integrated Script Environment*) steht hier bereit.

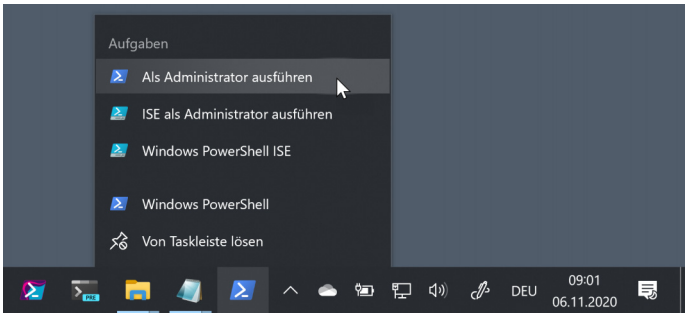


Abbildung 1.2: Sprungliste der Windows PowerShell

Die *Windows PowerShell* selbst erscheint als blaue oder schwarze Konsole, in die Sie Befehle eingeben können (siehe Abbildung 1.3). Auch wenn es Ihnen bereits in den Fingern juckt: Verschieben Sie Ihre Experimente bitte noch einen Moment – was Sie alles mit PowerShell tun können, veranschaulicht der Rundkurs im nächsten Kapitel.

Lassen Sie uns zunächst alles einsatzbereit machen.

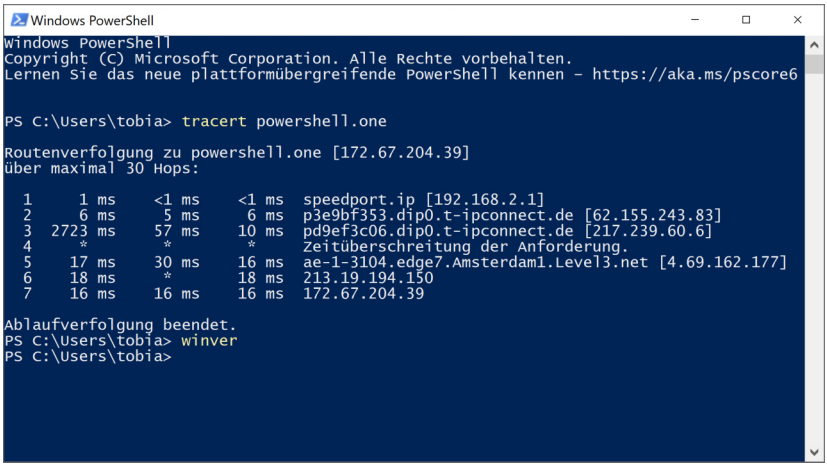


Abbildung 1.3: Die in Windows integrierte Windows PowerShell

Ist Ihre Windows PowerShell noch aktuell?

Falls Sie Windows 10 oder Server 2016 (und besser) verwenden, ist alles gut: Diese Betriebssysteme enthalten *Windows PowerShell* in Version 5.1, der letzten und ausgereiften Version. Diese Version wird über die gesamte Laufzeit von Windows 10 weitergepflegt und automatisch aktualisiert.

Wer ein älteres Windows-Betriebssystem nutzt, muss *Windows PowerShell* manuell aktualisieren. Das ist sinnvoll, weil die Codebeispiele dieses Buchs die aktuelle *Windows PowerShell 5.1* voraussetzen. Es ist aber vor allem wichtig, weil Updates stets auch Programmfehler und Sicherheitslücken berichtigen. Schauen Sie also in Ihrer *Windows PowerShell* kurz nach, welche Version Sie verwenden (siehe Abbildung 1.4). Geben Sie ein:

```
PS> $PSVersionTable.PSVersion ↵
```

Zurückgeliefert wird die Version der PowerShell. Bei Versionen kleiner als 5.1 (`Major: 5, Minor: 1`) ist es Zeit für ein Update.

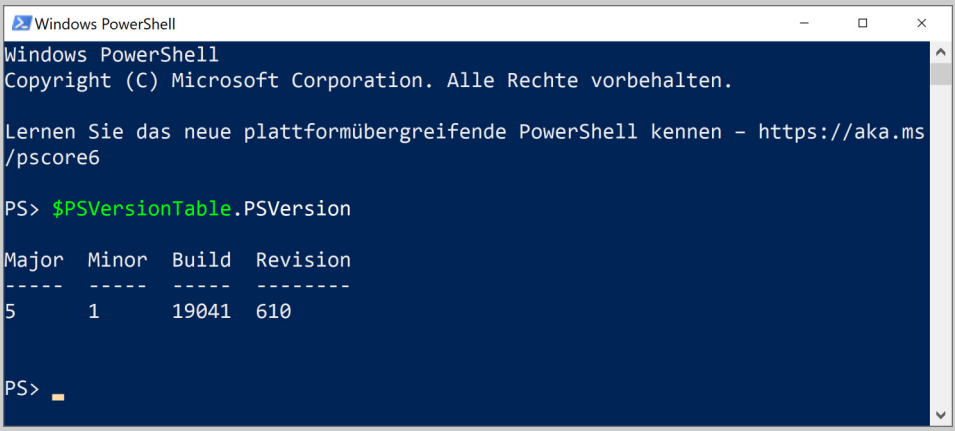


Abbildung 1.4: Version der PowerShell bestimmen

Weil *Windows PowerShell* ein fester Teil von Windows ist, kann man sie nur über spezielle Update-Pakete aktualisieren. Diese stehen kostenlos bei Microsoft als sogenannte *msu*-Pakete zum Download bereit. Bei Drucklegung dieses Buchs fand man das sogenannte *Windows Management Framework 5.1* hier: *https://www.microsoft.com/en-us/download/details.aspx?id=54616* (siehe Abbildung 1.5).

Tipp

Links ändern sich, und sollte der genannte Link nicht mehr funktionieren, googeln Sie nach »Windows Management Framework 5.1 Download«.

Bitte beachten Sie auf der Downloadseite die Installationshinweise am Ende der Seite, insbesondere wenn Sie noch das alte Windows 7 verwenden. *Windows PowerShell* erfordert übrigens das .NET Framework 4.5 oder besser, das bei sehr alten Windows-Systemen gegebenenfalls vor dem Update nachinstalliert werden muss.

Bei Clients ist ein Update der *Windows PowerShell* schnell erledigt. Bei Servern sollten Sie sich dagegen vor dem Update unbedingt genauer informieren: Serversoftware wie *Microsoft Exchange* kann auf eine ganz bestimmte *Windows PowerShell*-Version festgelegt sein. Entweder aktualisieren Sie in so einem Fall *Windows PowerShell* gemeinsam mit den übrigen Serverkomponenten, die davon abhängen, oder Sie installieren, wenn Ihnen das zu heikel oder zu aufwendig ist, parallel zu *Windows PowerShell* kurzerhand die neue moderne *PowerShell* und arbeiten künftig einfach mit dieser, während Sie *Windows PowerShell* unverändert lassen.

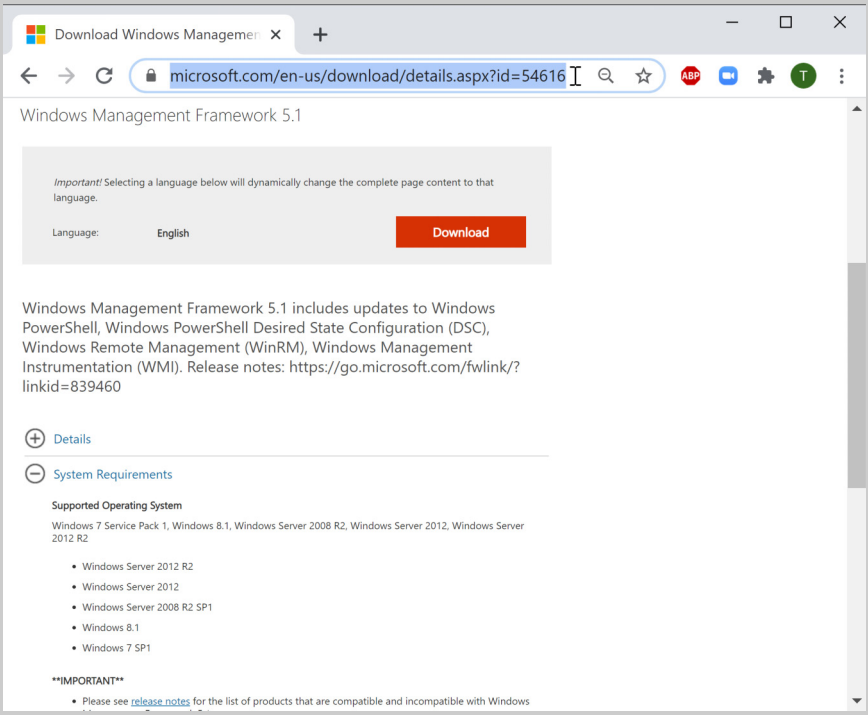


Abbildung 1.5: Windows PowerShell ist Teil des Windows Management Framework.

PowerShell 7: die »neue« PowerShell

Weil *Windows PowerShell* in das Windows-Betriebssystem fest integriert ist, kann sie – wie Sie gerade erlebt haben – nur relativ umständlich aktualisiert werden. Erst recht steht sie nicht auf anderen Betriebssystemen zur Verfügung. Deshalb hat das Microsoft-Entwicklungsteam um Steve Lee im Jahr 2016 die Entwicklung der *Windows PowerShell* abgeschlossen.

Alle folgenden Versionen der PowerShell wurden fundamental anders entwickelt: Sie sind nicht mehr Bestandteil von Windows, sondern eigenständige portable Anwendungen, die fortan auf dem neuen plattformunabhängigen *.NET Framework Core* basieren. So kann *PowerShell* nun auch auf Linux, macOS und anderen Betriebssystemen eingesetzt werden.

Die neue PowerShell heißt auch nicht länger *Windows PowerShell*, sondern nur noch *PowerShell*, die Anwendung `pwsh.exe` anstelle von `powershell.exe`.

Darüber hinaus ist *PowerShell* quellcodeoffen (Open Source) und portabel. Sie kann also beliebig oft und in beliebigen Versionen parallel existieren, und wenn Sie möchten, könnten Sie sie sogar wie ein persönliches Werkzeug auf einem USB-Stick bei sich tragen und vom Stick aus starten.

So ist die neue *PowerShell* sogar für Windows-Anwender interessant. Installieren Sie sie parallel zur vorhandenen *Windows PowerShell*, können Sie wahlweise *Windows PowerShell* über den Befehl `powershell` starten oder eben *PowerShell* über den Befehl `pwsh`.

Profitipp

Das wirft in Unternehmen häufig die Frage auf, ob (und, wenn ja, wie) man unternehmensweit von *Windows PowerShell* auf *PowerShell* umsteigen sollte. Schließlich ist die *Windows PowerShell* ja inzwischen »veraltet« und wird nicht länger weiterentwickelt. Stimmt das?

Nein. *Windows PowerShell* ist keineswegs abgekündigt und wird bis ans Ende der Laufzeit von Windows 10 weiter voll von Microsoft unterstützt. Auch etwaige Sicherheitslücken, sollten sie denn auftreten, werden weiterhin behoben, und *Windows PowerShell* ab Version 5.1 wird über das automatische Windows Update automatisch gepflegt.

Deshalb spricht nichts dagegen, *Windows PowerShell* auch weiterhin einzusetzen. *Windows PowerShell* hat gegenüber *PowerShell* sogar zwei unternehmenskritische Vorteile:

- **Automatische Verfügbarkeit:** Weil *Windows PowerShell* fester Bestandteil von Windows ist, steht sie immer automatisch zur Verfügung und wird ebenso automatisch aktualisiert. Wollen Sie stattdessen die neuere *PowerShell* einsetzen, müssten Sie sich eine eigene Deployment-Strategie überlegen, also *PowerShell* in eigener Verantwortung auf alle Computer im Unternehmen verteilen und natürlich auch zeitnah aktuell halten.
- **Vollständiges .NET Framework:** *Windows PowerShell* basiert auf dem klassischen vollständigen .NET Framework 4.5. Dies ist eine Softwarebibliothek, aus der sich die meisten Windows-Anwendungen bedienen. Die plattformübergreifende *PowerShell* dagegen basiert auf dem abgespeckten .NET Framework Core, derzeit in Version 3.1. Obwohl dieses Framework rund 99% der notwendigen Funktionen enthält, besteht doch die Möglichkeit, dass bestimmte Erweiterungen oder ältere Skripte darauf nicht lauffähig sind.

Wenn Sie also nicht unbedingt auf bestimmte neue Funktionalitäten in *PowerShell* angewiesen sind, spricht überhaupt nichts dagegen, im Unternehmen auch weiterhin flächendeckend *Windows PowerShell* einzusetzen. Spezialisten und PowerShell-Entwickler können davon unbenommen auf ihren eigenen Entwicklungssystemen und in eigener Regie natürlich dennoch bereits die neue *PowerShell* einsetzen, wenn sie ihnen Vorteile bietet.

Einfache Installation über den Microsoft Store

Am einfachsten funktioniert die Installation der *PowerShell* über den *Microsoft Store*. Öffnen Sie ihn, indem Sie im Windows-Startmenü nach »Store« suchen.

Achtung

In vielen Unternehmensumfeldern ist der Zugang zum Microsoft Store abgeschaltet. Hier installieren Sie die neue *PowerShell* manuell (siehe unten). Besondere Administratorrechte sind für die Installation nicht notwendig.

Natürlich kann auch die *Windows PowerShell* den Store für Sie öffnen (denn PowerShell kann alles, was auch per Maus zu bewerkstelligen ist – sofern Sie die passenden Befehle kennen):

```
Start-Process shell:appsFolder\Microsoft.WindowsStore_8wekyb3d8bbwe!App
```

Listing 1.1: Microsoft Store mit PowerShell öffnen

Wichtig

Wenn Sie die Codebeispiele aus dem Buch nicht von Hand eingeben wollen, rüsten Sie den Befehl `L` mit dieser Zeile nach:

```
PS> Invoke-RestMethod -Uri https://tinyurl.com/codeAusBuch -UseBasicParsing | New-Item -Path
function: -Name L -Force | Out-Null
```

Geben Sie hinter dem Befehl die Listing-Nummer an, beispielsweise:

```
PS> L 1.1
Listing 1.1 liegt in der Zwischenablage. Fügen Sie es mit STRG+V in Konsole oder Editor ein.
```

Mehr zum Befehl `L`, welche sonstigen Tricks er auf Lager hat und wie Sie den Befehl dauerhaft verfügbar machen, erfahren Sie ab Seite 63.

Klicken Sie im Microsoft-Store-Fenster auf *Suchen* und geben Sie als Suchwort »PowerShell« ein. Klicken Sie auf *PowerShell Preview* und dann auf *Installieren* (siehe Abbildung 1.6).

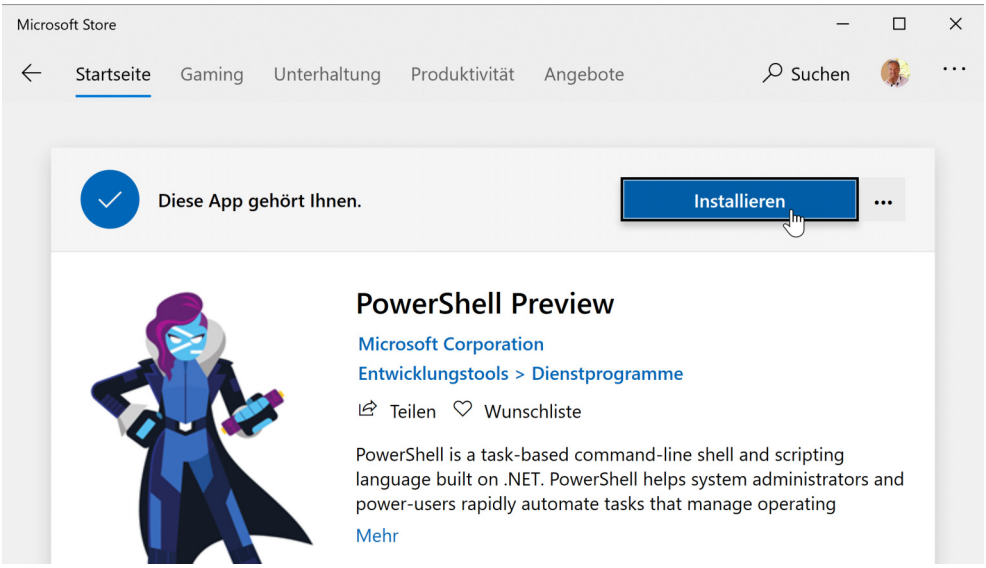


Abbildung 1.6: PowerShell 7 als Preview über den Microsoft Store installieren

Hinweis

Die Preview-Fassung der PowerShell enthält jeweils die allerneuesten Funktionen und eignet sich besonders gut zum Experimentieren und Kennenlernen. Darüber hinaus steht im Store neuerdings auch die Produktionsversion ohne den Zusatz »Preview« zur Verfügung.

Falls der Store Sie übrigens um Anmeldedaten bittet oder dazu drängen will, ein Benutzerkonto anzulegen, lehnen Sie dankend ab. Eine Anmeldung ist zum Herunterladen und Installieren nicht nötig.

Nach der Installation wird die neue *PowerShell* über den Befehl `pwsh` gestartet. Drücken Sie also zum Beispiel ⊞+R und geben Sie den Befehl `pwsh` ins Dialogfeld *Ausführen* ein. Dann klicken Sie auf *OK*.

Die neue *PowerShell* öffnet ein Konsolenfenster mit schwarzem Hintergrund (bei *Windows PowerShell* ist der Hintergrund blau). Auch das Symbol der neuen *PowerShell* ist schwarz und nicht blau. In der Titelleiste des Konsolenfensters lesen Sie außerdem den Pfad, unter dem die neue *PowerShell* installiert wurde.

Klicken Sie das Symbol der neuen *PowerShell* in der Taskleiste mit der rechten Maustaste an und wählen Sie *An Taskleiste anheften*. So bleibt es dauerhaft sichtbar, und Sie können künftig bequem per Mausklick die passende PowerShell öffnen (siehe Abbildung 1.7).

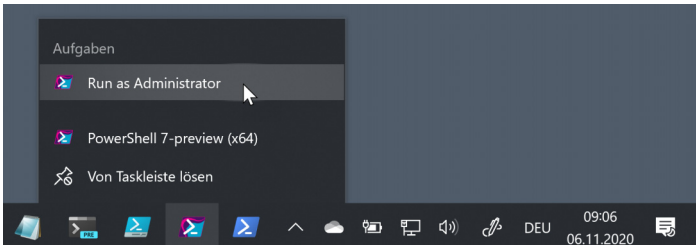


Abbildung 1.7: Windows PowerShell und PowerShell lassen sich in Windows parallel betreiben.

Tipp

Die angehefteten Symbole der Taskleiste befinden sich übrigens in einem separaten versteckten Ordner, den Sie über den folgenden Befehl öffnen können:

```
PS> explorer "$env:appdata\Microsoft\Internet Explorer\Quick Launch\User Pinned\TaskBar"
```

Wenn Sie die Version der neuen *PowerShell* durch Eingabe von `$PSVersion` überprüfen, offenbaren sich erste Unterschiede (siehe Abbildung 1.8):

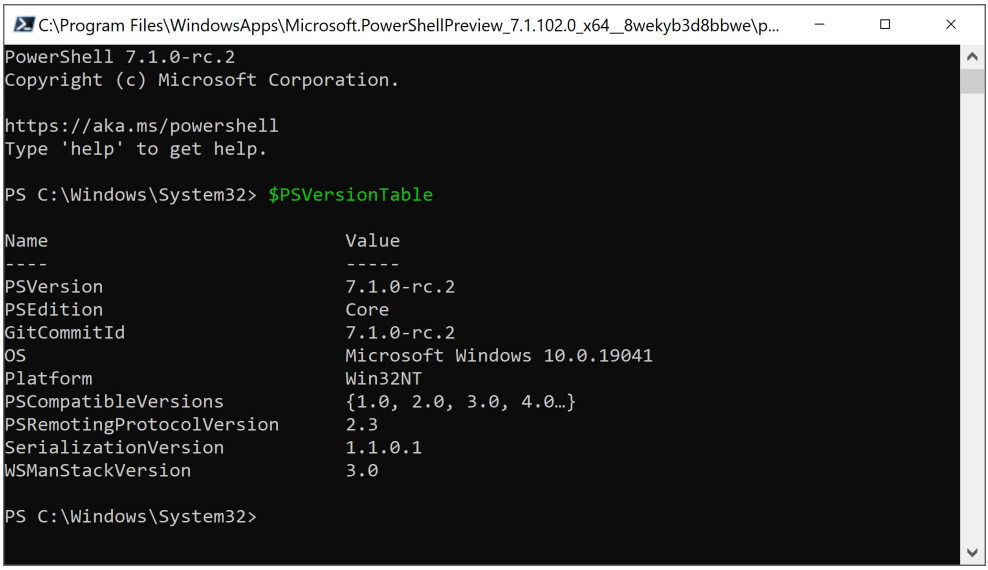


Abbildung 1.8: Eine als »App« über den Microsoft Store installierte PowerShell 7.1

`PSVersion` meldet die neue Version, und `PSEdition` zeigt `Core`. Bei *Windows PowerShell* steht hier `Desktop`.

Hinweis

Die Bezeichnungen in `PSEdition` haben ausschließlich historische Gründe:

Ursprünglich hieß die neue *PowerShell* nämlich *PowerShell Core* und war als minimalistische Version zur reinen Remote-Verwaltung der »Nano-Server« gedacht, denen man die Benutzeroberfläche wegrationalisiert hatte.

Weil die Entwicklung der Nano-Server aber schnell wieder eingestellt wurde, hat man PowerShell Core danach zu einer plattformübergreifenden Shell weiterentwickelt. Das Wort »Core« passt nun natürlich nicht mehr, und so heißt diese plattformübergreifende PowerShell heute einfach nur *PowerShell*. In der Eigenschaft `PSEdition` wird sie indes weiterhin `Core` genannt.

PowerShell manuell installieren

Künftig ist zwar geplant, in die *Windows PowerShell* einen Befehl zu integrieren, mit dem man die neue *PowerShell* automatisch herunterladen und installieren kann, bisher gibt es diesen bequemen Befehl jedoch nicht.

Man kann ihn allerdings bereits nachrüsten. Dafür sind keine besonderen Berechtigungen nötig. Mit diesem neuen Befehl lässt sich die plattformübergreifende *PowerShell* besonders flexibel manuell installieren, zum Beispiel auch dann, wenn der Zugang zum Microsoft Store nicht zur Verfügung steht.

Öffnen Sie dazu zunächst eine *Windows PowerShell*, zum Beispiel über ⊞+R und dann `powershell` ↵.

Danach führen Sie den folgenden Befehl aus (siehe Listing 1.2), um das neue Cmdlet `Install-PowerShell` nachzurüsten:

```
Invoke-RestMethod -Uri https://aka.ms/install-powershell.ps1 -UseBasicParsing | New-Item -Path
function: -Name Install-PowerShell | Out-Null
```

Listing 1.2: Fehlenden Installationsbefehl nachrüsten

`Invoke-RestMethod` lädt aus dem Internet den Quellcode für den neuen Befehl herunter. Diesen Quellcode könnten Sie auch sichtbar machen, indem Sie die Internetadresse in einen Browser eingeben:

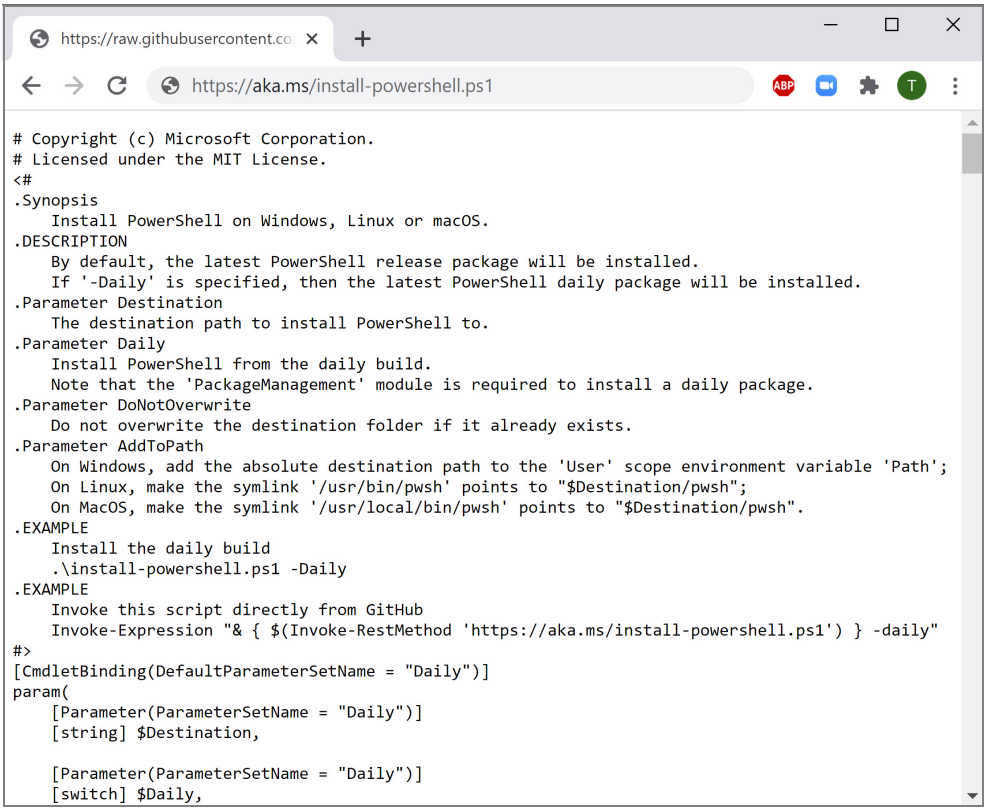


Abbildung 1.9: Microsoft stellt ein Installationsskript für PowerShell 7 zur Verfügung.

`New-Item` verwandelt den heruntergeladenen Quellcode dann in einen neuen Befehl namens `Install-PowerShell`. Er steht danach sofort zur Verfügung, und mit ihm können Sie nun ganz bequem PowerShell 7 nachinstallieren.

Hinweis

Mehr zu selbst definierten Cmdlets erfahren Sie in Kapitel 14. Was es mit `New-Item` und dem Laufwerk *function:* auf sich hat, ist Thema von Kapitel 7.

Der folgende Befehlsaufruf würde beispielsweise die aktuelle Produktivversion von *PowerShell 7* im Ordner *c:\ps7test* installieren und diesen Ordner zur `Path`-Umgebungsvariablen hinzufügen, sodass der Befehl `pwsh` von überall aus aufrufbar würde:

```
PS> Install-PowerShell -Destination c:\ps7test -AddToPath
```

In manchen Unternehmensszenarien verbieten die Sicherheitseinstellungen die Installation auf diese Weise, und Sie erhalten stattdessen rote Fehlermeldungen. Installieren Sie *PowerShell 7* dann stattdessen als verwaltetes MSI-Paket:

```
PS> Install-PowerShell -UseMSI
```

Wird *PowerShell 7* als MSI-Paket installiert, erscheint ein Dialogfeld mit Installationsoptionen. Übernehmen Sie die vorgegebenen Optionen. Wenn Sie später auch remote auf *PowerShell 7* zugreifen möchten, aktivieren Sie zusätzlich im Dialogfeld die Option für das sogenannte *Remoting*.

Auch eine unbeaufsichtigte Installation ist dabei mit dem Parameter `-Quiet` möglich. Ob PowerShell indes tatsächlich mit `-Quiet` unbeaufsichtigt installiert werden kann, hängt davon ab, ob Sie die Installation mit vollen Administratorrechten durchführen. Ohne sie erscheint trotz `-Quiet` ein Dialogfeld.

Profitipp

Da PowerShell eine »portable App« ist, können Sie potenziell so viele Versionen parallel zueinander in unterschiedlichen Ordnern installieren, wie Sie mögen, oder die portable PowerShell auf einem USB-Stick mit sich führen. Allerdings führt das zwangsläufig zu einem Dilemma: Welche davon würde nun der Befehl `pwsh` starten?

Wenn Sie den absoluten Pfadnamen verwenden, also den kompletten Pfadnamen einschließlich der Ordner, ist die Sache klar. Wird dagegen nur `pwsh` eingegeben, durchsucht Windows die Ordner in der Umgebungsvariablen `Path` und startet den ersten Treffer. Hier ist dann also die Reihenfolge maßgeblich, in der Ihre verschiedenen Installationen in der Umgebungsvariablen `Path` aufgeführt sind.

Sie können den Inhalt der Umgebungsvariable mit diesem Befehl auflisten:

```
PS> $env:path -split ';'
C:\ps7test
C:\WINDOWS\system32
C:\WINDOWS
C:\WINDOWS\System32\Wbem
...
```

Falls Sie *PowerShell* gestartet haben und nachträglich herausfinden möchten, wo diese *PowerShell* eigentlich gespeichert ist, wählen Sie folgenden Befehl:

```
PS> [Environment]::CommandLine
C:\ps7test\pwsh.dll
```

In diesem Beispiel sehen Sie, dass *PowerShell* aus dem Ordner *c:\ps7test* gestartet wurde. Sie sehen übrigens auch, dass die neue *PowerShell* eigentlich eine Bibliothek mit der Dateierweiterung **.dll* ist. Der Startbefehl `pwsh.exe` leitet den Aufruf also bloß an diese Bibliothek weiter. Nerds wie ich finden so etwas spannend, aber Sie können dieses Detail auch höflich ignorieren.

Den Pfad zur jeweils gerade genutzten *PowerShell* finden Sie hiermit heraus:

```
PS> (Get-Process -Id $pid).Path
C:\ps7test\pwsh.exe
```

Die Variable `$pid` enthält bei allen Versionen der PowerShell stets die Prozess-ID der gerade laufenden PowerShell-Instanz-Sitzung.

Kostenlose Softwareverteilungssysteme: Paketmanager

In Linux-Betriebssystemen sind Paketmanager wie `apt`, `yum`, `dnf`, `pacman` oder `brew` seit Langem im Einsatz. Sie laden zusätzliche Software automatisch aus dem Internet und installieren sie. Mit diesen Paketmanagern wird übrigens auch PowerShell auf Linux- und macOS-Systemen installiert.

Bei Windows sind solche Paketmanager noch relativ neu, setzen sich aber auch hier inzwischen durch: Am verbreitetsten sind *Chocolatey* (*https://chocolatey.org/install*) und *Scoop* (*https://scoop.sh/*).

Auch PowerShell ist auf diesen Zug aufgesprungen und enthält einen eigenen plattformunabhängigen Paketmanager namens *PowerShellGet* (siehe Kapitel 16): Mit ihm kann man kinderleicht PowerShell-spezifische Ressourcen wie neue Cmdlets oder Skripte nachinstallieren, und schon im nächsten Kapitel begegnen Ihnen zahlreiche Beispiele dazu.

Chocolatey und Scoop

Weil Paketmanager für viele Windows-Anwender noch relativ neu sind, sollen die beiden Windows-Paketmanager Chocolatey und Scoop kurz mit einigen Beispielen zeigen, was sie können und wie Paketmanager funktionieren.

Der grundlegende Unterschied zwischen den beiden ist die Art, wie Software installiert wird und welche Rechte dazu nötig sind: Während Chocolatey Installationspakete auf konventionelle Weise stets für *alle Anwender* installiert und dazu Administratorrechte benötigt, installiert Scoop Anwendungen ohne Administratorrechte und nur für den *aktuellen Anwender* als portable Apps.

Chocolatey eignet sich daher eher für die Unternehmens-IT, während sich Scoop an Endanwender richtet, die trotz eingeschränkter Rechte schnell und einfach zusätzliche Software installieren möchten.

Voraussetzungen

In beiden Fällen müssen Sie bei älteren Windows-Systemen zuerst den neueren Sicherheitsprotokollstandard *TLS 1.2* erlauben, der bei aktuellen Windows-Versionen automatisch unterstützt wird. Andernfalls können Sie über geschützte *https:*-Verbindungen keine Daten herunterladen:

```
[System.Net.ServicePointManager]::SecurityProtocol =
[System.Net.ServicePointManager]::SecurityProtocol -bor
[System.Net.SecurityProtocolType]::Tls12
```

Listing 1.3: Internetsicherheitsprotokoll TLS 1.2 für PowerShell erlauben

Da die Installation über PowerShell-Skripte erfolgt, müssen Sie außerdem mindestens vorübergehend für die aktuelle Sitzung die Ausführung von Skripten erlauben. Sie erfahren in Kapitel 3 mehr darüber. Einstweilen genügt dieser Befehl:

```
Set-ExecutionPolicy -ExecutionPolicy Bypass -Scope Process -Force
```

Listing 1.4: PowerShell-Skripte in der aktuellen PowerShell-Sitzung erlauben

---

Achtung

Führen Sie die folgenden Befehle unbedingt in einer PowerShell-Konsole aus und nicht in einem Editor wie zum Beispiel dem ISE-Editor.

Die Befehle der hier vorgestellten Softwareverteilungen können im laufenden Betrieb Rückfragen an Sie stellen, zum Beispiel ob ein fehlendes Softwarepaket mitinstalliert werden soll. Solche Rückfragen werden von Editoren wie ISE nicht unterstützt und unterdrückt, sodass das Skript bei Nachfragen in Editoren zu »hängen« scheint.

Chocolatey als Softwareverteilung

Wenn Sie über Administratorrechte verfügen und Software klassisch für alle Anwender des Computers installieren möchten, installieren Sie sich die Paketverwaltung Chocolatey. Dazu benötigen Sie aktivierte Administratorrechte.

Achtung

Chocolatey erfordert grundsätzlich zur Installation volle Administratorrechte und ist schlecht geeignet, wenn Sie Software nur für den aktuellen Anwender installieren möchten. Verwenden Sie für diesen Fall besser Scoop (siehe nächsten Abschnitt).

Listing 1.5 lädt mit `Invoke-RestMethod` zuerst das Installationsskript von Chocolatey herunter und führt es dann mit `Invoke-Expression` aus. Das dürfen Sie natürlich nur tun, wenn Sie den Quellen solcher Skripte vertrauen. Geben Sie die URL im Zweifelsfall in die Adressleiste Ihres Browsers ein, um den Code zuerst zu untersuchen oder im Unternehmensumfeld von jemandem freigeben zu lassen.

```
Invoke-RestMethod -UseBasicParsing -Uri 'https://chocolatey.org/install.ps1' | Invoke-Expression
```

Listing 1.5: Chocolatey installieren

Alles Weitere übernimmt das Installationsskript und berichtet über alle Schritte, die es ausführt:

```
Getting latest version of the Chocolatey package for download.
Getting Chocolatey from https://chocolatey.org/api/v2/package/chocolatey/0.10.15.
Downloading 7-Zip commandline tool prior to extraction.
Extracting C:\Users\tobia\AppData\Local\Temp\chocolatey\chocInstall\chocolatey.zip to
C:\Users\tobia\AppData\Local\Temp\chocolatey\chocInstall...
Installing chocolatey on this machine
Creating ChocolateyInstall as an environment variable (targeting 'Machine')
  Setting ChocolateyInstall to 'C:\ProgramData\chocolatey'
WARNING: It's very likely you will need to close and reopen your shell before you can use choco.
Restricting write permissions to Administrators
We are setting up the Chocolatey package repository.
The packages themselves go to 'C:\ProgramData\chocolatey\lib'
  (i.e. C:\ProgramData\chocolatey\lib\yourPackageName).
A shim file for the command line goes to 'C:\ProgramData\chocolatey\bin' and points to an executable
in 'C:\ProgramData\chocolatey\lib\yourPackageName'.
Creating Chocolatey folders if they do not already exist.
WARNING: You can safely ignore errors related to missing log files when
  upgrading from a version of Chocolatey less than 0.9.9.
  'Batch file could not be found' is also safe to ignore.
  'The system cannot find the file specified' - also safe.
Adding Chocolatey to the profile. This will provide tab completion, refreshenv, etc.
WARNING: Chocolatey profile installed. Reload your profile - type . $profile
Chocolatey (choco.exe) is now ready.
You can call choco from anywhere, command line or powershell by typing choco.
```