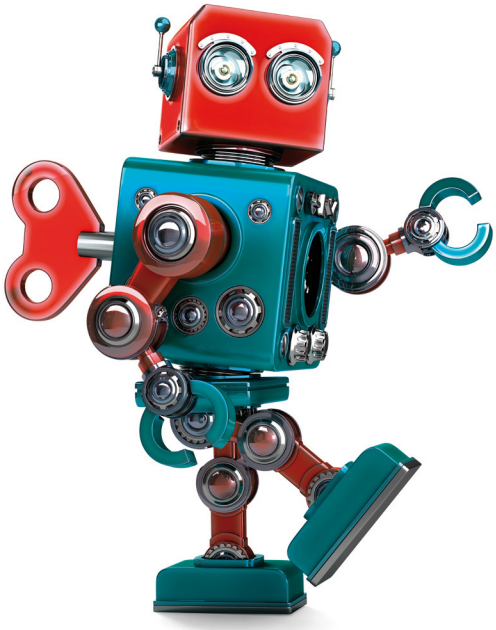
Mit vielen Beispielen und Übungen

Für alle Versionen

Windows PowerShell

Grundlagen & Scripting-Praxis für Einsteiger

Dr. Tobias Weltner



O'REILLY®

Dr. Tobias Weltner

Windows PowerShell – Grundlagen und Scripting-Praxis für Einsteiger

O'REILLY®

Dr. Tobias Weltner

Lektorat: Ariane Hesse
Korrektorat: Sibylle Feldmann, www.richtiger-text.de
Satz: Gerhard Alfes, mediaService, www.mediaservice.tv
Herstellung: Susanne Bröckelmann
Umschlaggestaltung: Michael Oreal, www.oreal.de, unter Verwendung eines Fotos von Kirillm/iStock by Getty Images
Druck und Bindung: M.P. Media-Print Informationstechnologie GmbH, 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 978-3-96009-073-1
PDF 978-3-96010-177-2
ePub 978-3-96010-178-9
mobi 978-3-96010-179-6

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.

1. Auflage 2018

Wieblinger Weg 17
69123 Heidelberg

5 4 3 2 1 0

Inhalt

Einleitung

PowerShell ist als Automationssprache entwickelt worden: IT-Administratoren und ambitionierte Computeranwender können damit Routineaufgaben automatisch erledigen. Das spart Zeit und lästige Handgriffe und ist aus der modernen IT nicht mehr wegzudenken.

Darüber hinaus ist PowerShell inzwischen aber auch eine sehr effektive Programmiersprache geworden, mit der man ohne komplexes Vorwissen und in nur wenigen Zeilen Code kleine Programme und Tools schreiben kann. Das ist für nahezu jeden ambitionierten Computeranwender interessant, nicht mehr lediglich für IT-Profis.

Und spätestens mit PowerShell 6 laufen PowerShell-Skripte nun sogar nicht mehr nur auf Windows, sondern auch auf Linux und MacOS. Das macht PowerShell-Fachwissen noch erheblich vielseitiger.

Dieses Buch beginnt bei null und führt Sie Schritt für Schritt in alle wichtigen Grundlagen der PowerShell ein. Ob Sie PowerShell beruflich nutzen oder privat, quer über das Netzwerk auf Hunderten von Computern Programme konfigurieren oder bloß auf dem eigenen Notebook Urlaubsbilder umbenennen wollen: Dieses Buch liefert zahlreiche Praxisbeispiele und erklärt die eingesetzten Techniken so, dass Sie die Beispiele leicht an eigene Bedürfnisse anpassen können.

Sieben didaktisch aufeinander aufbauende Teile führen Sie ohne notwendiges Vorwissen in die Möglichkeiten der PowerShell ein.

Im ersten Kapitel erfahren Sie, wie PowerShell auf dem Computer eingerichtet wird und welche essenziellen Einstellungen und ersten Schritte wichtig sind, um PowerShell-Befehle ausführen zu können.

In **Teil A** wird die PowerShell als praktische interaktive Konsole betrachtet – schnelle Ergebnisse mit ganz wenigen Eingaben. Sie entdecken, welche Befehle die PowerShell versteht, wie man schnell die benötigten Befehle findet und neue Befehle nachrüsten kann. In diesem Teil wird auch das universelle Laufwerkkonzept der PowerShell vorgestellt, mit dem beinahe alles – vom Dateisystem über Zertifikate und Registrierung bis hin zu Datenbanken und Benutzerverwaltung – als Laufwerk gesehen werden kann, auf dem man Informationen findet.

Teil B zeigt dann, wie aus mehreren Einzelbefehlen und der PowerShell-Pipeline komplexere und mächtigere Automationsskripte entstehen, mit denen sich vielfältige Aufgaben vollständig automatisieren lassen.

Teil C beschäftigt sich mit Operatoren, die oft unterschätzt werden. Hier erfahren Sie, wie man intelligente Entscheidungen (Bedingungen) formuliert. Einen besonderen Schwerpunkt bilden die Textoperatoren und sogenannten »regulären Ausdrücke«, mit denen PowerShell die für Sie wichtigen Informationen aus Texten, Logdateien und von Webseiten fischen kann.

PowerShell kommt bereits mit Hunderten eigener Befehle, doch bietet es vor allem eine Schnittstelle zu den unzähligen Betriebssystemfunktionalitäten, die im .NET Framework bereitgestellt werden. In **Teil D** erfahren Sie, wie PowerShell auf all diese Funktionen zugreifen kann. So erhalten Ihre PowerShell-Skripte Zugriff auf genau die gleichen Möglichkeiten, die auch Anwendungsentwicklern anderer .NET-Sprachen zur Verfügung stehen, und können von DNS-Auflösung über Sprachausgabe bis hin zu eigenen Fenstern und Oberflächen alles nachrüsten, was noch nicht über vorgefertigte PowerShell-Cmdlets erreichbar war.

Weil PowerShell eine dynamische Sprache ist, kann man den Befehlssatz sehr leicht erweitern. **Teil E** erklärt zuerst, wie Ihr PowerShell-Code zu einem PowerShell-Befehl wird, der dann in allen PowerShells zur Verfügung steht. Danach erfahren Sie, wie PowerShell-Module erzeugt werden: Mit ihnen lassen sich Befehlssammlungen an Kollegen und Kunden weitergeben. Und schließlich wird das brandneue PowerShellGet vorgestellt, über das Module automatisiert verteilt werden können.

Wenn PowerShell dabei einmal nicht genau das tut, was Sie sich vorstellen, hilft **Teil F** und zeigt, wie Fehler im Code erkannt und »behandelt« werden können. Dazu zählt auch das Mitprotokollieren von Fehlern und ungewöhnlichen Zuständen.

In **Teil G** wird zum Abschluss das »PowerShell Remoting« vorgestellt. Mit diesem extrem mächtigen Feature kann PowerShell-Code remote auf einem oder vielen anderen Computern parallel ausgeführt werden. Weil PowerShell inzwischen auch auf Linux und MacOS ausführbar ist, kann man darüber unter anderem auch Informationen in heterogenen Welten austauschen.

Dieses Buch ist eine überarbeitete und gekürzte Fassung des über 1.100 Seiten starken »Windows PowerShell 5: Windows Automation für Einsteiger & Profis«, das sich auf die für Einsteiger wichtigen Inhalte fokussiert.

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 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 viel Spaß und Erfolg 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 startklar machen

In diesem Kapitel:

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*). Darüber hinaus gibt es weitere kommerzielle und freie PowerShell-Hosts.
- **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`.

Mit PowerShell Kontakt aufnehmen

PowerShell ist eigentlich unsichtbar, denn es ist eine Automationssprache, die seit 2009 fest in das Windows-Betriebssystem eingebettet ist. Um mit ihr Kontakt aufzunehmen, wird ein Programm benötigt, das Ihre Befehle und Anweisungen an die PowerShell schicken kann und von dort die Ergebnisse zurückerhält.

Solche Programme werden »Host« genannt (engl. für »Gastgeber«). Windows enthält zwei solcher Host:

- **PowerShell-Konsole:** ein sehr einfaches Programm namens »powershell.exe«, das in einem Konsolenfenster angezeigt wird. Es ist der Standard-Host, wenn fertige PowerShell-Skripte ausgeführt werden sollen. Sie können es zwar auch dazu verwenden, einzelne PowerShell-Befehle interaktiv einzugeben, aber weil es wenig Hilfestellungen bietet und keine Skripte verfassen kann, eignet es sich nicht gut zum Erlernen der PowerShell.
- **PowerShell ISE:** eine PowerShell-Entwicklungsumgebung namens »powershell_ise.exe«. »ISE« steht für *Integrated Script Environment*, denn die ISE enthält nicht nur eine interaktive Befehlskonsole, sondern auch einen Texteditor, mit dem man PowerShell-Skripte verfassen

kann. Darüber hinaus enthält die ISE viele Hilfestellungen, wie etwa IntelliSense-Menüs, die die zur Verfügung stehenden Befehle anzeigen. Damit kann man PowerShell sehr viel bequemer erlernen als mit der simplen PowerShell-Konsole.

Beide Hosts kommunizieren mit derselben unsichtbaren PowerShell, sprechen also genau dieselbe Sprache und verstehen dieselben Befehle. Sie unterscheiden sich nur in der Bedienerfreundlichkeit.

PowerShell 6 auf Linux und MacOS

Wenn Sie kein Windows-Betriebssystem verwenden, sondern Linux oder MacOS, steht dort seit Neuestem ebenfalls die PowerShell zur Verfügung: PowerShell 6 (auch »PowerShell Core« genannt) ist ein Microsoft-Open-Source-Projekt, das hier kostenfrei heruntergeladen werden kann: *https://github.com/PowerShell/PowerShell/releases*. Scrollen Sie auf der Seite abwärts, bis Sie den Bereich der Downloads sehen. Pro Betriebssystem werden unterschiedliche Downloads angeboten.

Das PowerShell-6-Installationspaket enthält sowohl die unsichtbare PowerShell-Sprache als auch einen Host namens *pwsh.exe*. Ein komfortables Entwicklungssystem wie die ISE gibt es hier nicht.

PowerShell 6 ist eine portable Anwendung und kann auch auf Windows-Systemen parallel zur fest in Windows integrierten PowerShell genutzt werden.

Allerdings erkauft sich PowerShell 6 die Plattformunabhängigkeit mit deutlich eingeschränkter Funktionalität: PowerShell 6 basiert nämlich nicht auf dem vollen Umfang der Windows-Programmierschnittstellen (.NET), sondern auf einer portablen Minifassung (.NET Core). Obwohl es also möglich ist, PowerShell 6 auch auf Windows einzusetzen, ist das derzeit wenig sinnvoll.

Die in Windows integrierte PowerShell leistet wesentlich mehr. Aufgrund der PowerShell-6-Einschränkungen kann man darin zum Beispiel keine grafischen Benutzeroberflächen anzeigen, die die bei Windows üblichen Techniken »Forms« oder »WPF« nutzen. Deshalb fehlen in PowerShell 6 alle Befehle, die Ergebnisse auf solche Weise anzeigen. Und wegen der übrigen eingeschränkten Programmierschnittstellen lassen sich auch viele andere wichtige Windows-Technologien wie Active Directory, Microsoft Office oder Exchange nicht automatisieren.

PowerShell 6 ist nur der erste Versuch von Microsoft, die Grenzen des eigenen Betriebssystems zu sprengen, und der derzeit eingeschränkte Funktionsumfang wird künftig sicher wachsen. Einstweilen bietet Ihnen die plattformunabhängige PowerShell 6 aber die Möglichkeit, auch auf Nicht-Windows-Systemen erste Erfahrungen mit PowerShell zu sammeln und heterogene Umgebungen einheitlich zu verwalten.

In Teil G erfahren Sie mehr dazu, wie PowerShell auch remote auf anderen Computern ausgeführt werden kann. So könnte ein Skript, das in einer Windows-PowerShell läuft, per »Remoting« Kontakt zu einem PowerShell 6 auf einem Linux-System aufnehmen und dort Einstellungen vornehmen.

In diesem Buch werden alle wesentlichen Funktionen von Windows PowerShell beschrieben. Viele Grundtechniken funktionieren unverändert auch in PowerShell 6, aber Sie wissen nun, warum einige Befehle dort nicht vorhanden sind oder nicht richtig funktionieren.

Weitere PowerShell-Hosts

Neben der mitgelieferten Konsole und dem ISE-Entwicklungssystem gibt es unzählige weitere PowerShell-Hosts, kommerzielle wie frei verfügbare.

Microsoft bietet zum Beispiel mit »VSCode« (*Visual Studio Code*) einen kostenlosen Texteditor, für den es ein ebenfalls kostenfreies Plug-in namens »PowerShell Editor Services« gibt. Damit wird auch aus VSCode eine leistungsfähige und hochmoderne PowerShell-Entwicklungsumgebung. Sie richtet sich allerdings eher an erfahrenere Anwender, weil sie über unzählige Tastenkombinationen bedient wird und wenig anklickbare Funktionen bietet.

VSCode ist ähnlich wie PowerShell 6 plattformunabhängig und ist deshalb besonders auf Nicht-Windows-Betriebssystemen wichtig, auf denen es keine anderen PowerShell-Entwicklungssysteme gibt.

PowerShell und Hosts einsatzbereit machen

Damit Sie gleich bequem mit PowerShell arbeiten können, schauen wir uns im folgenden Abschnitt die verfügbaren PowerShell-Hosts näher an und überprüfen direkt auch einige wichtige PowerShell-Grundeinstellungen. Diesen Teil sollten Sie nicht überspringen, denn nur so ist gewährleistet, dass bei Ihnen alle PowerShell-Funktionalitäten richtig funktionieren und Sie mit den aktuellsten Versionen arbeiten.

Anschließend wählen Sie sich unter den verfügbaren Hosts einfach denjenigen aus, der Ihnen am sympathischsten ist. Weil alle Hosts mit derselben unsichtbaren PowerShell zusammenarbeiten, stehen Ihnen in allen Hosts also grundsätzlich die gleichen Befehle zur Verfügung, und auch die Ergebnisse sind die gleichen. Die Wahl des Hosts ist also eine reine Geschmacksfrage und hängt davon ab, wie viel Unterstützung Sie sich bei der Arbeit mit der PowerShell von Ihrem Host wünschen.

Die PowerShell-Konsole einrichten

Die PowerShell-Konsole ist der Basis-Host, der am wenigsten Speicher benötigt und deshalb auch der Standard-Host ist, in dem später fertige Skriptlösungen ausgeführt werden. Zum Entdecken der PowerShell und zum Konzipieren neuer PowerShell-Skripte ist die Konsole weniger gut geeignet, weil sie so spartanisch gestaltet ist und wenig Hilfestellung liefert. Sie wird aber gern genutzt, wenn man nur wenige Befehle einzugeben hat oder die nötigen Befehle ohnehin auswendig kennt.

Um die PowerShell-Konsole erstmals zu starten, öffnen Sie mit ⊞+R das *Ausführen*-Fenster und geben darin ein: `powershell` ↵.

In Windows 10 erreichen Sie PowerShell wahlweise auch wieder über das Startmenü (Abbildung 1.1).

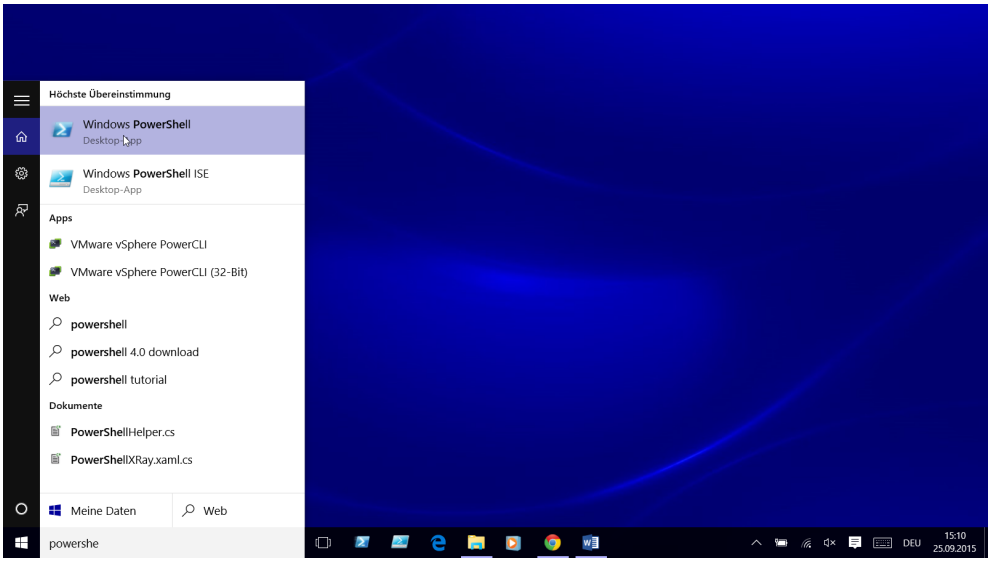


Abbildung 1.1: PowerShell über das Startmenü in Windows 10 öffnen.

Wenige Sekunden später präsentiert sich die hochmoderne objektorientierte PowerShell-Konsole. Besonders eindrucksvoll ist das Erlebnis anfangs indes nicht, denn es erscheint nur ein hässliches schwarzes oder blaues Konsolenfenster. Darin begrüßt Sie die Eingabeaufforderung, die mit »PS« beginnt und dahinter den Pfadnamen des aktuellen Ordners anzeigt. Außerdem blinkt eine Einfügemarke und ermuntert Sie mit dem Charme der 1980er-Jahre dazu, erste Befehle einzugeben (Abbildung 1.2).

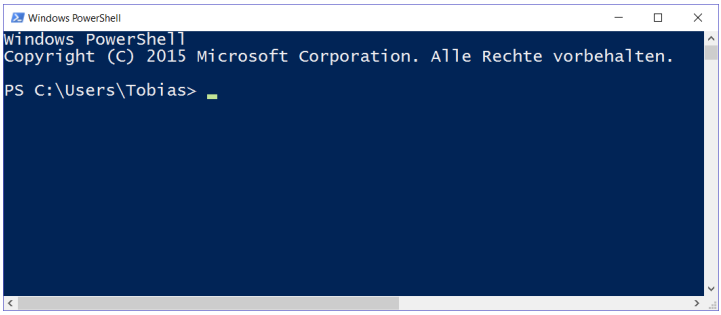


Abbildung 1.2: Die PowerShell-Konsole führt interaktive PowerShell-Befehle aus und benötigt wenig Speicher.

PowerShell-Version kontrollieren

Kontrollieren Sie zuerst, welche PowerShell-Version Sie verwenden. Es gibt fünf Versionen, die aufeinander aufbauen. Dieses Buch handelt von der aktuellsten PowerShell-Version, also 5.0, was Sie inspirieren sollte, Ihre PowerShell-Version ebenfalls auf Versionsstand 5.0 zu aktualisieren, falls Sie eine ältere Version vorfinden. Andernfalls können Sie einige in diesem Buch beschriebene Funktionalitäten nicht nutzen.

Ein besonders einfacher Weg, die Version Ihrer PowerShell zu prüfen, ist ein Blick auf das Copyright, das beim Start der Konsole erscheint:

Copyright-Jahr	PowerShell-Version
2006	Version 1.0. Diese Version ist veraltet und sollte nicht mehr eingesetzt werden.
2009	Version 2.0. Eingeführt mit Windows 7/Windows Server 2008R2. Als Update verfügbar für Windows XP, Vista sowie Windows Server 2003 und 2008.
2012	Version 3.0. Eingeführt mit Windows 8 und Windows Server 2012. Kann auf PowerShell 5.0 aktualisiert werden (außer auf Windows-7-Clients).
2013	Version 4.0. Eingeführt mit Windows 8.1 und Server 2012R2. Kann auf PowerShell 5.0 aktualisiert werden.
2015	Version 5.0. Eingeführt mit Windows 10 und Server 2016.
2016	Version 5.1. Ab dieser Version erhalten Sie künftige Updates automatisch über den Windows-Update-Mechanismus.
2017	Version 6.0. Diese Version wird auch »PowerShell Core« genannt und ist plattformübergreifend auch auf Linux und MacOS verfügbar. Sie ist aber nicht der Nachfolger von PowerShell 5.0, auch wenn die Version dies suggeriert. PowerShell 5 bleibt die in Windows fest integrierte PowerShell-Version und ist auf Windows-Systemen wesentlich leistungsfähiger als PowerShell 6 (siehe auch die Anmerkungen am Anfang des Kapitels).

Tabelle 1.1: PowerShell-Versionen identifizieren.

So finden Sie die aktuell verwendete PowerShell-Version per Befehl heraus:

```
PS C:\> $PSVersionTable

Name                           Value
----                           -----
PSVersion                      5.1.15063.726
PSEdition                      Desktop
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
BuildVersion                   10.0.15063.726
CLRVersion                     4.0.30319.42000
WSManStackVersion              3.0
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1
```

`$PSVersionTable` liefert Ihnen unter anderem auch die Information `PSEdition`, die Ihnen mit `Desktop` verrät, dass Sie die Windows PowerShell verwenden. Steht hier dagegen `Core`, verwenden Sie die in ihren Funktionen eingeschränkte, aber dafür plattformübergreifende PowerShell 6.0.

Auf Nicht-Windows-Systemen ist das eine gute Idee, weil Ihnen andernfalls überhaupt keine PowerShell zur Verfügung stünde.

Auf Windows-Systemen dagegen wissen Sie inzwischen, dass PowerShell 6 nur über einen eingeschränkten Funktionsumfang verfügt und deshalb Befehle fehlen können, die in diesem Buch beschrieben werden.

Aktuelle Version der PowerShell nutzen

Weil dieses Buch auf der neuesten Version der Windows PowerShell, PowerShell 5.1, basiert, sollten Sie, falls noch nicht geschehen, Ihre PowerShell auf den aktuellsten Stand bringen. So erhalten Sie nicht nur neue und bessere Befehle, die Aktualisierung hat auch einen Sicherheitsaspekt: Die aktuellste Version der PowerShell ist sehr viel sicherer als ältere Versionen, und

sobald Sie einmal PowerShell 5.1 installiert haben, übernimmt fortan das normale Windows Update die weitere Aktualisierung.

Ob Ihre PowerShell aktuell ist oder nicht, zeigt Ihnen `$PSVersionTable`. Hinter `PSVersion` findet sich die Version Ihrer PowerShell. Steht also hinter `PSVersion` nicht »5.1«, verwenden Sie eine veraltete Version.

Das kostenfreie Update für PowerShell 5.1 bietet ein Windows-Updatepaket (mit der Erweiterung *.msu*), das im Internet kostenfrei bereitsteht und per Doppelklick installiert wird. Da sich der Downloadlink ändern kann, öffnen Sie am besten eine Suchseite wie *google.de* und suchen kurz nach den Stichwörtern »Windows Management Framework 5.1« und »Download«, um das Updatepaket zu finden. Der letzte offizielle Downloadlink bei Drucklegung dieses Buchs war *https://www.microsoft.com/en-us/download/details.aspx?id=54616*.

Auf Ihrem eigenen Computer ist das Update auf eine neue PowerShell-Version ganz unkompliziert und erfordert höchstens einen Neustart. Bevor Sie allerdings PowerShell auf einem Produktionsserver aktualisieren, informieren Sie sich vorher über die sonstige Software, die darauf läuft. Es gibt Software, die eng mit PowerShell verzahnt ist, beispielsweise Microsoft Exchange oder SQL Server. Diese Programme sind fest an bestimmte PowerShell-Versionen gebunden und müssen gemeinsam mit der PowerShell aktualisiert werden.

Symbol an Taskleiste heften

Als Nächstes sollten Sie die PowerShell-Konsole besser erreichbar machen. Dazu klicken Sie das PowerShell-Symbol in der Taskleiste mit der rechten Maustaste an und wählen im Kontextmenü *Dieses Programm an Taskleiste anheften* (Abbildung 1.3). Ziehen Sie das Symbol danach in der Taskleiste mit der Maus an den äußersten linken Rand, sodass es das erste Symbol in der Taskleiste ist. Schließen Sie die PowerShell-Konsole und öffnen Sie sie danach erneut mit einem Klick auf das angepinnte Symbol in der Taskleiste.

Abbildung 1.3: PowerShell-Symbol an die Taskleiste anheften.

Sie können PowerShell nun auch über die Tastenkombination ⊞+1 öffnen oder in den Vordergrund holen, sofern das PowerShell-Symbol das erste in Ihrer Taskleiste ist. Andernfalls verwenden Sie eine andere Zahl, die der Position des Symbols in der Taskleiste entsprechen muss. Spätestens jetzt sollte sich ein blaues und nicht schwarzes Konsolenfenster öffnen. Windows speichert Einstellungen wie die blaue Hintergrundfarbe oder die Bildschirmpuffergröße der Konsole in Verknüpfungen.

Sprungliste: Administratorrechte und ISE

Haben Sie die PowerShell-Konsole wie oben beschrieben an die Taskleiste angeheftet und danach mindestens einmal gestartet, öffnet ein Rechtsklick auf das angeheftete Konsolensymbol nun die PowerShell-Sprungliste: ein Menü mit den wichtigsten PowerShell-Startbefehlen (Abbildung 1.4).

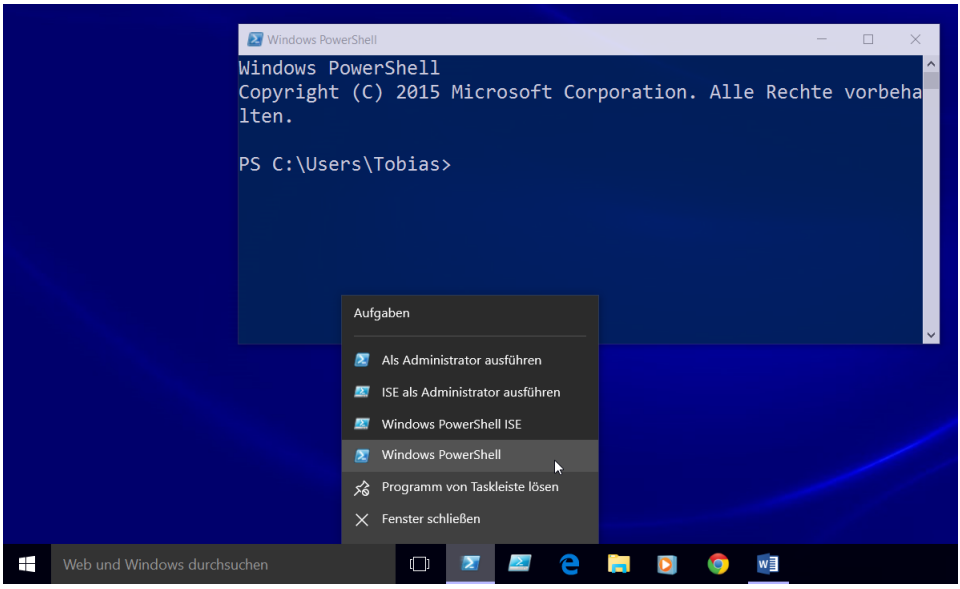


Abbildung 1.4: Sprungliste von PowerShell öffnen.

Über die enthaltenen Befehle können Sie PowerShell bei Bedarf mit vollen Administratorrechten starten (was Sie im Normalfall zum eigenen Schutz eher vermeiden und nur einsetzen sollten, wenn diese Rechte tatsächlich gebraucht werden). Auch die ISE, der integrierte Skripteditor, kann über die Sprungliste wahlweise normal oder mit Administratorrechten geöffnet werden.

Befehl	Beschreibung
Als Administrator ausführen	Öffnet die interaktive PowerShell-Konsole mit allen Rechten.
ISE als Administrator ausführen	Öffnet den PowerShell-Editor mit allen Rechten.
Windows PowerShell ISE	Öffnet den integrierten PowerShell-Skripteditor.
Windows PowerShell	Öffnet die interaktive PowerShell-Konsole ohne besondere Rechte.

Tabelle 1.2: Befehle in der PowerShell-Sprungliste.

32-Bit- und 64-Bit-Versionen

Auf 64-Bit-Versionen von Windows gibt es sowohl die PowerShell-Konsole als auch den ISE-Editor in doppelter Ausführung. Neben der 64-Bit-Version stehen zusätzlich 32-Bit-Versionen bereit. Sie sind daran zu erkennen, dass an den Namen ein »(x86)« angefügt ist.

Die 32-Bit-Versionen der PowerShell-Hosts sind nur für Ausnahmefälle gedacht, in denen ein Skript ausdrücklich im 32-Bit-Subsystem ausgeführt werden muss. Nötig ist das nur selten, zum Beispiel dann, wenn PowerShell auf Komponenten zugreifen soll, die es nur als 32-Bit-Versionen gibt. Im normalen Alltag setzen Sie immer die regulären 64-Bit-Versionen ein und achten darauf, dass hinter dem Programmnamen eben nicht der Zusatz »(x86)« steht.

PowerShell ISE einsetzen

Die PowerShell ISE ist ein modernerer PowerShell-Host als die Konsole und bietet zum Beispiel praktische IntelliSense-Menüs mit Befehlsvorschlägen sowie rote Kringellinien, die Fehler anzeigen. Sie öffnen die ISE entweder über ihren Programmnamen *powershell_ise.exe* oder noch bequemer über den Befehl `ise` aus einer bereits geöffneten PowerShell-Konsole.

```
PS> ise ↵
```

Wenn Sie die PowerShell-Konsole wie eben beschrieben an Ihre Taskleiste angeheftet haben, genügt auch ein Rechtsklick auf das angeheftete blaue PowerShell-Symbol, um die Sprungliste zu öffnen und darin mit dem Kontextmenübefehl *Windows PowerShell ISE* die ISE wahlweise mit oder ohne Adminstratorrechte zu starten.

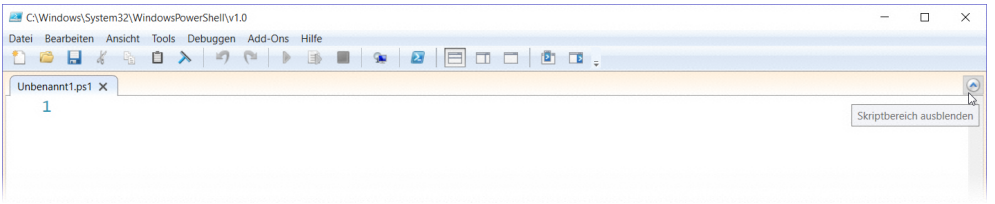


Abbildung 1.5: PowerShell ISE als Konsolenersatz.

Weil PowerShell ISE nicht nur ein moderner Ersatz für die interaktive Konsole ist, sondern auch als Skripteditor dient, sieht das Fenster möglicherweise bei Ihnen etwas anders aus als das in Abbildung 1.5. Mit der Pfeilschaltfläche in der oberen rechten Ecke blenden Sie den Skriptbereich ein und aus. Im Augenblick sollten Sie den Skriptbereich im versteckten Zustand belassen. Über Strg+R kann der Skriptbereich gänzlich mauslos sichtbar und wieder unsichtbar gemacht werden.

Hilfreich sind auch die Schaltflächen in der Symbolleiste, mit denen Sie den interaktiven Konsolenteil wahlweise unten oder an der Seite anzeigen oder bei Bedarf eben auch ganz ausblenden, um maximalen Platz zur Eingabe von Skripten zu haben (Abbildung 1.6).

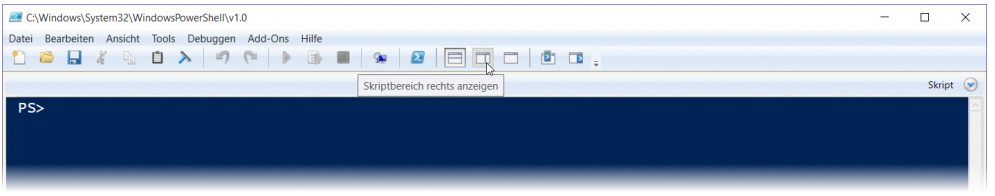


Abbildung 1.6: Über die Werkzeugleiste kann die interaktive PowerShell-Konsole ein- und ausgeblendet werden.

Mit dem Schieberegler am unteren rechten Fensterrand variieren Sie nahtlos die Schriftgröße. Ohne Maus verwenden Sie dazu Strg+[+] und Strg+[-].

Möchten Sie auch die Schriftart ändern, rufen Sie *Tools/Optionen* auf. Im Dialogfeld aktivieren Sie das Kontrollkästchen *Nur Festbreitenschriftart*, denn PowerShell ISE kommt zwar im Unterschied zur Konsole auch mit Proportionalschriftarten zurecht, aber weil hier die Schriftzeichen unterschiedlich breit sind (einem *m* wird zum Beispiel mehr Platz eingeräumt als einem *i*), führt dies zu Problemen bei der Ausgabe, wenn Tabellenspalten verrutschen und nicht mehr bündig erscheinen (Abbildung 1.7).

Im Listenfeld *Schriftfamilie* sehen Sie jetzt alle Schriftarten mit fester Zeichenbreite. Die Schriftart, die Sie auswählen, gilt sowohl für den interaktiven Konsolenbereich als auch den Skripteditor. Nicht alle Schriftarten, die die Liste anbietet, sind wirklich gut zu gebrauchen. Eine besonders gut lesbare Schriftart heißt »Consolas«. Die »Lucida Console« ist die Standardschriftart.

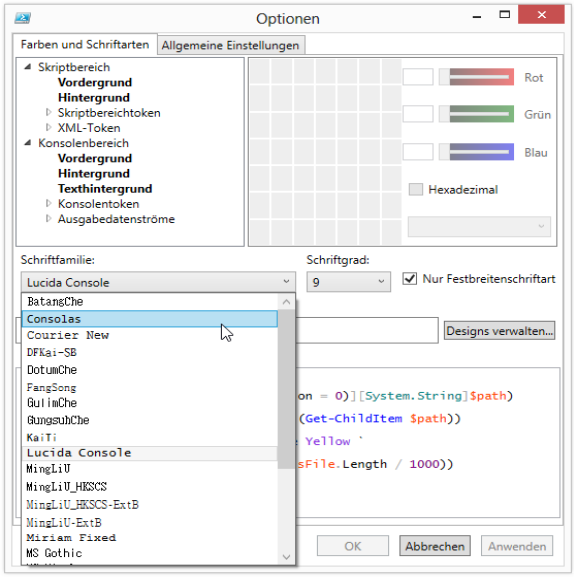


Abbildung 1.7: Andere Schriftart für ISE auswählen.

Notfalls stellt die Schaltfläche *Standard wiederherstellen* in der linken unteren Ecke des Dialogfelds die Ausgangseinstellungen wieder her.

Hinweis

Genügt Ihnen der eingebaute Funktionsumfang der ISE nicht, lässt er sich über kommerzielle und kostenfreie Erweiterungen ergänzen. Die folgenden Beispiele und Befehle müssen in der ISE ausgeführt werden.

Eine häufig genutzte kostenfreie Erweiterung ist zum Beispiel der »ISE Project Explorer«, mit dem Sie Projekte, die aus mehreren Dateien bestehen, direkt in der ISE verwalten können.

So wird das Modul heruntergeladen und installiert:

```
PS> Install-Module -Name PSISEProjectExplorer -Scope CurrentUser ↵
```

Wenn Sie das zum ersten Mal tun, wird nachgefragt, ob eine kostenfreie Downloadkomponente installiert werden soll. Stimmen Sie zu. Anschließend wird gefragt, ob Sie die Erweiterung laden möchten. Wenn Sie erneut zustimmen, können Sie den Project Explorer künftig mit diesem Befehl laden:

```
PS> Import-Module PsISEProjectExplorer ↵
```

Möchten Sie das nicht jedes Mal durchführen, kann die Erweiterung auch automatisch geladen werden, wenn Sie den folgenden Befehl verwenden.

```
PS> Add-PsISEProjectExplorerToIseProfile ↵
```

Achtung

Diese Erweiterung ist kostenfrei, weil sie von ambitionierten Anwendern als Hobbyprojekt betrieben wird. Sie kann ausgezeichnet funktionieren, aber auch dazu führen, dass die ISE instabil wird, weil sie nicht professionell entwickelt wurde. Ob sie für Sie nützlich ist, müssen Sie selbst prüfen.

Testen Sie die Erweiterung daher ausgiebig, bevor Sie sie automatisch laden lassen. Ohne Autostart brauchen Sie nur die ISE neu zu starten, um die Erweiterung wieder loszuwerden.

Wie Autostarts funktionieren und wie Sie den Autostart auch wieder entfernen können, lesen Sie etwas später in Kapitel 5. Hinter dem Autostart stehen nämlich sogenannte Profilskripte, die – falls vorhanden – beim Start der PowerShell automatisch ausgeführt werden.

Sobald die Erweiterung geladen ist, wird über ein weiteres Panel rechts damit begonnen, die Dateien in Ihrem Dokumente-Ordner zu analysieren. Sie können darin künftig bequem zu Skriptdateien navigieren, vor allem aber über das Textfeld am Oberrand blitzschnell Dateien finden, die bestimmte Stichwörter enthalten. Das indes funktioniert erst, nachdem die Erweiterung die Suche abgeschlossen und seinen Dateiindex erstellt hat.

Eine sehr verbreitete kommerzielle Erweiterung heißt »ISESteroids«, die zum Beispiel Echtzeit-Codeverbesserungen, einen Variablenmonitor und automatische Codegeneratoren liefert. Diese Erweiterung kann testweise ohne Einschränkungen genutzt werden und wird über diesen Befehl heruntergeladen und installiert:

```
PS> Install-Module ISESteroids -Scope CurrentUser ↵
```

Nach Download und Installation wird die ISESteroids-Erweiterung mit dem folgenden Befehl bei Bedarf geladen:

```
PS> Start-Steroids ↵
```

Tipp

PowerShell ISE erhält in der Taskleiste ein eigenes Symbol. Um ISE künftig direkt per Klick zu starten, klicken Sie mit der rechten Maustaste auf das Symbol von ISE in der Taskleiste und wählen *Dieses Programm an Taskleiste anheften*. Danach schieben Sie es nach links neben das Symbol der PowerShell-Konsole und können nun per Klick entscheiden, ob Sie die klassische Konsole oder lieber ISE öffnen möchten.

Denken Sie aber daran, dass nur das Symbol der PowerShell-Konsole per Rechtsklick die Sprungliste öffnet. Das Symbol der ISE verfügt über keine Sprungliste.

VSCode und PowerShell

Neuerdings steht mit »VSCode« ein weiterer kostenfreier Texteditor zur Verfügung, der im Gegensatz zu ISE viele weitere Skriptformate unterstützt und außerdem plattformunabhängig auch auf Linux und MacOS zur Verfügung steht.

Er richtet sich allerdings an erfahrenere Anwender, weil vieles nur über kryptische Befehlskürzel angesprochen werden kann. Hier kann VSCode heruntergeladen werden: *https://code.visualstudio.com/download*.

PowerShell-Erweiterung laden

Damit VSCode zu einem vollwertigen PowerShell-Host wird, klicken Sie nach seiner Installation und anschließendem Start in der linken Symbolleiste auf das unterste Symbol. Auf diese Weise öffnet sich eine Spalte mit Editorerweiterungen, die anfangs noch leer ist. Geben Sie ins Textfeld an seinem oberen Rand `powershell` ein.

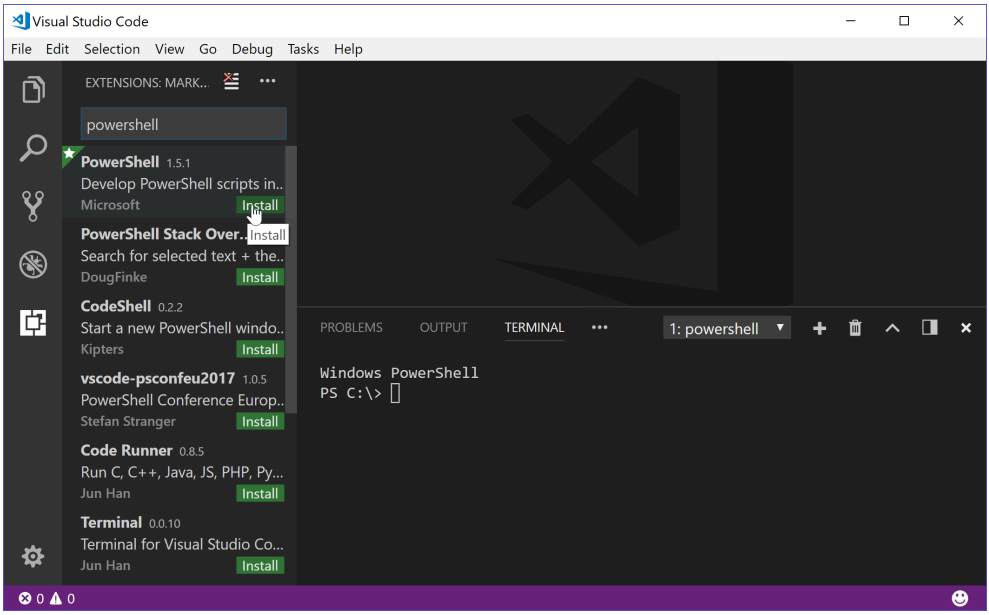


Abbildung 1.8: PowerShell-Erweiterung in VSCode installieren.

Wenig später wird die Erweiterung »PowerShell« gefunden, die mit einem Klick auf *Install* heruntergeladen und installiert wird. Nach einem Neustart ist die Erweiterung geladen.

VSCode zeigt Ihnen nun nicht nur unten eine PowerShell-Konsole an, sondern bietet jetzt auch umfangreiche IntelliSense- und andere Hilfen an, wenn Sie PowerShell-Skripte laden.

Um ein Skript auszuführen, sind indes Tastenkombinationen nötig. Mit F5 wird ein gesamtes Skript ausgeführt und mit F8 die aktuelle Selektion. Die gleichen Tastenkombinationen gelten ebenfalls in der ISE, jedoch stehen hier dafür auch Schaltflächen zur Verfügung.

Im Rahmen dieses Buchs werden die vielen weiteren Möglichkeiten von VSCode nicht weiter vertieft. Da aktuell beinahe wöchentlich Aktualisierungen bereitgestellt werden, ändern sich viele Bedienkonzepte derzeit noch.

Erste Schritte mit PowerShell

Die Ausführung von Befehlen funktioniert in allen PowerShell-Hosts gleich: Sie geben einen Befehl ein, schicken ihn mit einem entschlossenen Druck auf ↵ ab und warten dann gespannt, was als Nächstes geschieht. Wie Sie herausfinden, welche Befehle Ihnen zur Verfügung stehen, werden Sie gleich erfahren.

Wichtige Vorsichtsmaßnahmen

Damit das, was dann als Nächstes geschieht, keine unschöne Überraschung wird, sind ein paar vorausschauende Vorsichtsmaßnahmen ratsam. Mit nur zwei simplen Regeln entschärfen Sie das Potenzial karrierelimitierender Fehleingaben erheblich:

1. **Keine Administratorrechte:** Starten Sie PowerShell ohne spezielle Administratorrechte! So sind alle Einstellungen gesperrt, die das System ernstlich in Bedrängnis brächten. Ist die Windows-Benutzerkontensteuerung aktiv, passiert das automatisch (sofern Sie nicht über die Sprungliste auf vollen Administratorrechten bestehen). Ob PowerShell mit vollen Administratorrechten arbeitet, zeigt die Titelleiste des Fensters, in der dann das Wort *Administrator:* erscheint. Auf Servern ist die Windows-Benutzerkontensteuerung indes meist ausgeschaltet, sodass PowerShell hier stets mit vollen Rechten startet und Sie für erste Tests und die Einarbeitung in PowerShell besser ein eingeschränktes Benutzerkonto einrichten und verwenden sollten.
2. **Simulationsmodus:** Schalten Sie einen versteckten Simulationsmodus für noch mehr Schutz (und Einschränkungen) ein. Er bewirkt, dass PowerShell Änderungen am Computer nur simuliert, aber nicht ausführt. Dieser Schutz erstreckt sich auf die eingebauten PowerShell-Befehle, nicht aber auf klassische Konsolenbefehle wie beispielsweise *shutdown.exe*. So wird der Simulationsmodus eingeschaltet:

   ```
   PS> $WhatIfPreference = $true ↵
   ```

   Er gilt nur für die PowerShell-Instanz, in der der Befehl eingegeben wurde, und auch nur, bis diese Instanz wieder geschlossen wird.

Befehle eingeben

Im Fenster sehen Sie die Eingabeaufforderung. Sie beginnt mit `PS`, und dahinter steht der Pfadname des Ordners, in dem Sie sich gerade befinden. Eine blinkende Einfügemarke wartet auf Ihre ersten Eingaben. Sie werden gleich erfahren, welche Befehle PowerShell versteht, probieren Sie die Eingabe aber schon einmal aus. Geben Sie zum Beispiel ein:

```
PS> hallo ↵
```

Sobald Sie ↵ drücken, wird Ihre Eingabe an PowerShell geschickt und verarbeitet. Das Ergebnis folgt postwendend und ist in diesem Fall eine nüchterne rote Fehlermeldung:

```
hallo : Die Benennung "hallo" wurde nicht als Name eines Cmdlet, einer Funktion, einer Skriptdatei
oder eines ausführbaren Programms
erkannt. Überprüfen Sie die Schreibweise des Namens, oder ob der Pfad korrekt ist (sofern enthal-
ten), und wiederholen Sie den Vorgang.
In Zeile:1 Zeichen:1
+ hallo
+ ~~~~~
    + CategoryInfo          : ObjectNotFound: (hallo:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException
```

Fehlermeldungen sind zwar üblicherweise eher unerfreulich, doch sollten Sie sich schon einmal daran gewöhnen, sie nicht routinemäßig zu ignorieren. Oft verraten sie bei PowerShell tatsächlich den Grund des Problems, und auch in diesem Beispiel ist das, was die Fehlermeldung zu sagen hat, recht treffend: Die Benennung »hallo«, also das, was Sie als Befehl an PowerShell geschickt haben, war kein ausführbarer Befehl. Ausführbare Befehle sind gemäß Fehlermeldung Cmdlets, Funktionen, Skriptdateien oder ausführbare Programme.

Den kryptischen Teil nach dem Klartext dürfen Sie freundlich ignorieren. Er verrät erfahrenen PowerShell-Skriptentwicklern bei Bedarf noch mehr über die Natur des Fehlers und wo genau er aufgetreten ist. Spannend wird dieser Teil erst, wenn Sie umfangreichere PowerShell-Skripte starten.

Profitipp

Falls es Sie stört, dass PowerShell in epischer Breite den Pfadnamen des aktuellen Ordners im Prompt anzeigt, geben Sie einmal diesen Befehl ein:

```
PS> cd \ ↵
```

Damit wechseln Sie in den Stammordner des aktuellen Laufwerks, also vermutlich nach *C:*, und der Prompttext wird jetzt wesentlich kürzer und vergeudet keinen wertvollen Platz mehr in der Konsole. Später werden Sie bessere Wege kennenlernen, um den Prompt angenehmer zu formatieren, aber einstweilen hilft dieser Kniff schon mal weiter.

Ergebnisse empfangen

Geben Sie einen gültigen Befehl ein, wirft PowerShell fröhlich die erwarteten Ergebnisse aus. Möchten Sie zum Beispiel sehen, welche Dateien und Ordner sich in Ihrem aktuellen Ordner befinden, geben Sie ein: `dir` ↵.

Sie erhalten eine mehr oder weniger lange Textliste, und es drängt sich das Gefühl auf, dass der Ordnerinhalt in einem normalen Explorer-Fenster mit seinen bunten Symbolen viel einfacher zu erfassen ist. Grundsätzlich kommuniziert PowerShell mit Ihnen auf Textbasis. Dass PowerShell mehr kann als ein Explorer-Fenster, zeigt der nächste Befehl, der sämtliche laufenden Prozesse auflistet:

```
PS> Get-Process ↵
```

Die Stärke von PowerShell ist also nicht unbedingt die Darstellung der Informationen, sondern vielmehr ihre ungeheure Flexibilität. Fast alle Belange und Informationen Ihres Computers las-

sen sich von hier aus steuern und anzeigen – wenn auch »nur« als Textdarstellung und mithilfe von Textbefehlen.

Hier die wichtigsten weiteren Grundregeln:

- **Groß- und Kleinschreibung:** Diese spielt bei Befehlen keine Rolle. PowerShell ist also nicht *case sensitive*. Bei Argumenten, also Informationen, die Sie einem Befehl zusätzlich mit auf den Weg geben, kann die Groß- und Kleinschreibung im Einzelfall dagegen sehr wohl entscheidend sein, zum Beispiel bei Kennwortabfragen.
- **Abbrechen und löschen:** Möchten Sie einen Befehl vorzeitig abbrechen, drücken Sie Strg+C. Um die aktuelle Eingabe zu löschen, drücken Sie Esc. Möchten Sie den Inhalt des Konsolenfensters löschen, verwenden Sie den Befehl `cls`.

Informationen speichern oder umleiten

Alle Befehle der PowerShell liefern »körperlose«, nackte Informationen. Wie diese letzten Endes dargestellt oder verwendet werden, steht auf einem anderen Blatt. Im einfachsten Fall unternehmen Sie nichts weiter mit den Informationen. Sie oxidieren dann automatisch zu Text, den die Konsole anzeigt.

Alternativ könnten Sie die Informationen aber auch auf genau zwei Arten weiterverarbeiten:

- Variablen: Sie speichern das Ergebnis in eigenen Variablen, sodass das Ergebnis später an anderer Stelle weitergegeben werden kann, zum Beispiel an einen anderen Befehl.
- Pipeline: Sie geben das Ergebnis direkt an einen anderen Befehl weiter und nutzen dazu das Pipeline-Symbol »|«. Das ist besonders speicherschonend, weil keine Variablen und damit auch kein Speicherplatz benötigt werden.

Tun Sie nichts von beidem, werden die Ergebnisse stets in der Konsole angezeigt. Tun Sie beides, hat die Pipeline Vorrang.

Variablen verwenden

Variablen werden bei PowerShell immer mit einem `$` gekennzeichnet. Ansonsten arbeiten sie sehr unbürokratisch und müssen nicht besonders deklariert werden. Sie funktionieren wie Aufbewahrungsboxen für Informationen:

```
PS> $info = ipconfig.exe ↵
```

Die Informationen des Befehls liegen jetzt in der Variablen und werden nicht sichtbar ausgegeben. Erst wenn Sie die Variable ausgeben, tauchen die Informationen wieder auf:

```
PS> $info ↵

Windows-IP-Konfiguration

Drahtlos-LAN-Adapter Wi-Fi:

   Verbindungsspezifisches DNS-Suffix: Speedport_W_921V_1_39_000
   IPv6-Adresse. . . . . . . . . . . : 2003:40:e765:5043:7ca6:5208:b378:5c84
   Temporäre IPv6-Adresse. . . . . . : 2003:40:e765:5043:6485:6291:7855:a81
   Verbindungslokale IPv6-Adresse  . : fe80::7ca6:5208:b378:5c84%11
   IPv4-Adresse  . . . . . . . . . . : 192.168.2.119
   Subnetzmaske  . . . . . . . . . . : 255.255.255.0
   Standardgateway . . . . . . . . . : fe80::1%11
                                       192.168.2.1
```

```
Ethernet-Adapter Bluetooth Network Connection:

   Medienstatus. . . . . . . . . . . : Medium getrennt
   Verbindungsspezifisches DNS-Suffix:
(...)
```

Mit Operatoren lassen sich die Informationen in Variablen dann zum Beispiel bearbeiten. Der nächste Befehl fischt aus der Variablen nur die Zeilen heraus, die den Begriff »IPv4« enthalten:

```
PS> $info -like '*IPv4*' ↵
   IPv4-Adresse  . . . . . . . . . . : 192.168.2.119
```

Pipeline verwenden

Mit der Pipeline (|) reichen Sie die Informationen von einem Befehl direkt an den nächsten weiter. Ein sehr wichtiger Befehl ist zum Beispiel `Out-GridView`. Er zeigt die Ergebnisse in einem Extrafenster an, das beinahe so aussieht wie ein Excel-Fenster. So bleiben die Informationen nicht nur im Blick, es wird auch (anders als in der Konsole) bei Platzmangel nichts abgeschnitten. Und ein Klick auf eine Spaltenüberschrift sortiert auch gleich den Inhalt.

```
PS> ipconfig.exe | Out-GridView ↵
```

Richtig gut funktioniert das, wenn Befehle nicht reinen Text zurückliefern, sondern sogenannte »Objekte«. Objekte strukturieren Informationen in einzelnen Spalten, den sogenannten »Eigenschaften« oder »Properties«. Der nächste Befehl liefert beispielsweise alle Dienste und verrät interessante Details zu jedem Dienst:

```
PS> Get-Service ↵

Status   Name               DisplayName
------   ----               -----------
Running  AdobeARMservice    Adobe Acrobat Update Service
Stopped  AJRouter           AllJoyn-Routerdienst
Stopped  ALG                Gatewaydienst auf Anwendungsebene
Stopped  ANTS Memory Pro... ANTS Memory Profiler 8 Service
Stopped  ANTS Performanc... ANTS Performance Profiler 9 Service
Stopped  AppIDSvc           Anwendungsidentität
Running  Appinfo            Anwendungsinformationen
Running  Apple Mobile De... Apple Mobile Device Service
(...)
```

Werden solche Informationen weitergeleitet, zum Beispiel an `Out-GridView`, werden die Einzelinformationen in separaten Spalten angezeigt und lassen sich im GridView beispielsweise per Klick auf die Spaltenüberschrift sortieren (Abbildung 1.9):

```
PS> Get-Service | Out-GridView ↵
```

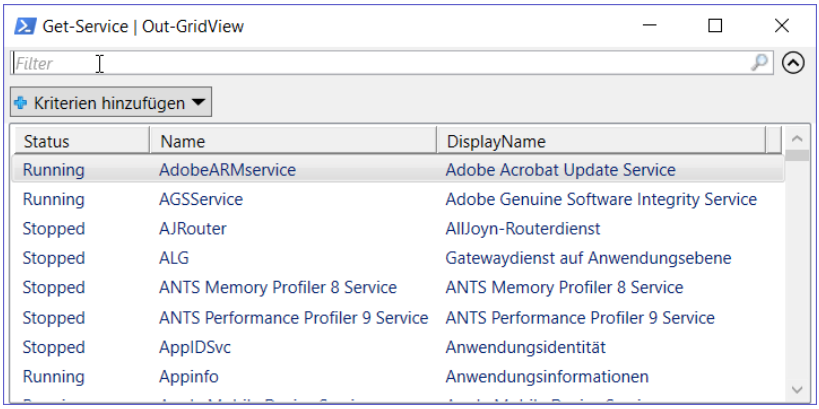


Abbildung 1.9: Befehlsergebnisse in einem Extrafenster anzeigen, dem »GridView«.

Hier erhalten Sie quasi bereits einen Vorgeschmack auf den »objektorientierten« Charakter der PowerShell, der in den folgenden Kapiteln immer wieder aufgegriffen wird. Mit `Select-Object` lassen sich so beispielsweise die Informationen bestimmen, an denen Sie interessiert sind:

```
PS> Get-Service | Select-Object -Property Status, DisplayName ↵

 Status DisplayName
 ------ -----------
Running Adobe Acrobat Update Service
Stopped AllJoyn-Routerdienst
Stopped Gatewaydienst auf Anwendungsebene
Stopped ANTS Memory Profiler 8 Service
Stopped ANTS Performance Profiler 9 Service
Stopped Anwendungsidentität
Running Anwendungsinformationen
(...)
```

Dasselbe Cmdlet macht auch deutlich, dass viele Befehle in Wahrheit sehr viel detailliertere Informationen liefern, als von der PowerShell-Konsole zunächst angezeigt werden. Fordern Sie mit dem Jokerzeichen * sämtliche Informationen an, prasseln sehr viel mehr Informationen auf Sie ein als ohne diesen Zusatz:

```
PS> Get-Service | Select-Object -Property * ↵


Name                  : AdobeARMservice
RequiredServices      : {}
CanPauseAndContinue   : False
CanShutdown           : False
CanStop               : True
DisplayName           : Adobe Acrobat Update Service
DependentServices     : {}
MachineName           : .
ServiceName           : AdobeARMservice
ServicesDependedOn    : {}
ServiceHandle         :
Status                : Running
ServiceType           : Win32OwnProcess
Site                  :
Container             :
```

```
Name                : AJRouter
RequiredServices    : {}
CanPauseAndContinue : False
(...)
```

PowerShell schaltet die Darstellung dabei automatisch vom Tabellen- in den Listenmodus, weil nun zu viele Informationen anzuzeigen sind, als in eine einzige Textzeile passen würden.

IntelliSense-Unterstützung im ISE-Editor

Es ist durchaus beeindruckend, was die PowerShell leisten kann, auch wenn die Beispiele im vorangegangenen Abschnitt wohl mehr Fragen als Antworten aufgeworfen haben. Störend ist zum Beispiel, dass die PowerShell nur dann etwas für Sie tut, wenn Sie die richtigen Befehle kennen. Es gibt keine praktischen Schaltflächen und Menüs in der textorientierten Befehlswelt der Automationssprachen.

Viel mehr Hilfestellung als in der Konsole erhalten Sie, wenn Sie zum ISE-Editor greifen. Dieser blendet IntelliSense-artige Auswahlmenüs ein, sobald Sie ein Schlüsselzeichen wie den Bindestrich (-) eingeben, und hilft Ihnen schon einmal etwas, auf intuitive Weise Befehle zu finden (Abbildung 1.10).

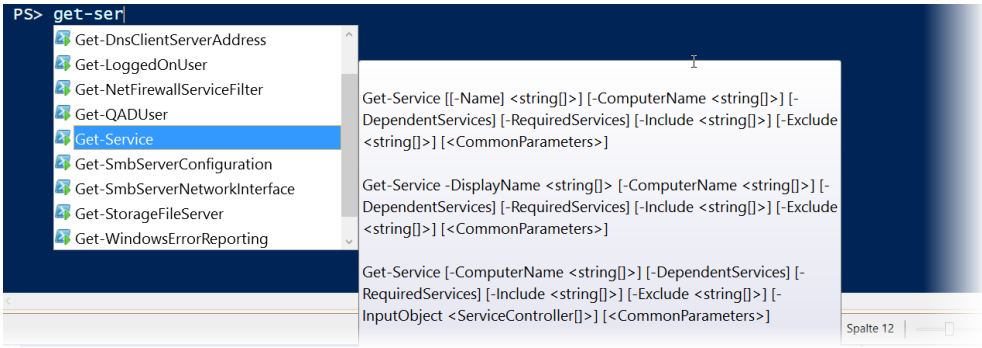


Abbildung 1.10: Moderne IntelliSense-Vervollständigung im ISE-Editor.

Das IntelliSense-Menü kann auch manuell jederzeit über Strg+Leertaste geöffnet werden.

Tipp

Ob ISE IntelliSense-Menüs anzeigen soll, bestimmen Sie über *Tools/Optionen* auf der Registerkarte *Allgemeine Einstellungen* im Bereich *IntelliSense*.

Autovervollständigung in der PowerShell-Konsole

In der PowerShell-Konsole steht – immerhin – eine Autovervollständigung zur Verfügung. Ein Druck auf ⇆ genügt, um die aktuelle Eingabe zu vervollständigen. Drücken Sie die Taste mehrmals, um weitere Vorschläge zu erhalten. ⇧+⇆ blättert einen Vorschlag zurück, falls Sie zu schnell waren.

Tipp

Bei der Autovervollständigung über ⇆ gilt die »Dreier-Regel«: Geben Sie mindestens drei Zeichen ein, bevor Sie ⇆ drücken. Bei PowerShell-Befehlen geben Sie mindestens den ersten Namensteil, den Bindestrich und dann drei Zeichen ein. Andernfalls gibt es zu viele infrage kommende Möglichkeiten, und ⇆ muss viel zu oft gedrückt werden, bis das richtige Ergebnis vorgeschlagen wird.

Die Autovervollständigung dient nicht nur der Bequemlichkeit. Sie vermeidet auch Tippfehler und macht sie deutlich. Liefert ⇆ zum Beispiel gar kein Resultat, liegt der Verdacht nahe, dass Sie sich bei Ihrer vorhandenen Eingabe bereits vertippt haben. Überprüfen Sie in diesem Fall, was Sie bisher eingegeben haben, und korrigieren Sie die Eingabe falls nötig. Danach versuchen Sie ⇆ noch einmal.

Die Autovervollständigung über ⇆ steht übrigens auch in ISE bereit und vervollständigt dann sofort, ohne dass sich ein Auswahlmenü einblendet.

Ab PowerShell 5 kann die Konsole sogar mit IntelliSense-artigen Auswahlmenüs aufwarten. Geben Sie den Beginn eines Befehls ein und drücken dann Strg+Leertaste, zeigt die Konsole die noch infrage kommenden Befehle an, die Sie mit den Pfeiltasten auswählen und mit der Leertaste übernehmen. Geben Sie zum Beispiel Folgendes ein und drücken Sie dann Strg+Leertaste:

```
PS> Get-Pr Strg+Leertaste
```

In PowerShell 5 erscheint nun das Auswahlmenü der noch infrage kommenden Befehle, aus denen Sie sich per Pfeiltasten einen aussuchen können (Abbildung 1.11).

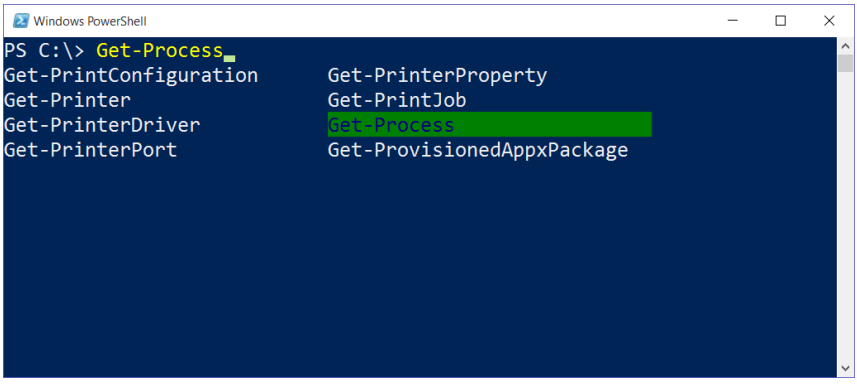


Abbildung 1.11: PowerShell 5 bietet auch in der Konsole eine Befehlsauswahl an.

Ist die Eingabe nicht eindeutig genug, fragt PowerShell gegebenenfalls nach, ob Sie wirklich alle infrage kommenden Befehle sehen wollen. Wenn Sie die Frage mit J beantworten, ergießt sich eine lange Liste möglicher Vervollständigungen – nicht so praktisch.

Zuständig dafür ist eine PowerShell-Erweiterung namens `PSReadLine`, die noch eine ganze Reihe weiterer Tricks auf Lager hat, wie Sie etwas später sehen. Möchten Sie diese Erweiterung in der PowerShell-Konsole nicht nutzen, geben Sie ein:

```
PS> Remove-Module PSReadLine ↵
```

Farbcodierungen verstehen

Eine wichtige Hilfestellung ist die Farbcodierung der aktuellen Befehlszeile. Sie zeigt sich »bunt«, solange Sie etwas darin eingeben und Ihre Eingabe noch nicht mit ↵ an PowerShell geschickt haben. Die Farben dienen nicht bloß der Unterhaltung, sie verdeutlichen, wie PowerShell Ihre Eingaben interpretiert. Viele Eingabefehler lassen sich mithilfe der Farbcodierung besser verstehen und vermeiden.

Abbildung 1.12: Farbcodierungen zeigen, wie PowerShell Ihre Eingaben versteht.

Geben Sie in ISE beispielsweise ein:

```
PS> dir C:\Windows ↵
```

Das IntelliSense-Menü unterstützt Sie bei der Eingabe des Pfadnamens, und die Befehlszeile selbst wird in mehreren Farben dargestellt. Der Befehl `dir` erscheint in Weiß. Der Pfadname dagegen wird in Pink angezeigt (Abbildung 1.12). Weiße Befehlswörter repräsentieren also stets Befehle, und pinke Elemente entsprechen Argumenten (Zusatzinformationen), die Sie einem Befehl anfügen.

Ab PowerShell 5.0 bringt auch die PowerShell-Konsole mit dem Modul `PSReadLine` die Farbcodierung für die aktuelle Eingabe mit (Abbildung 1.13):

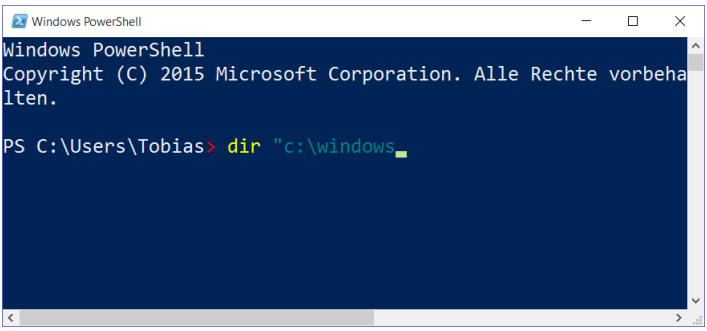


Abbildung 1.13: Farbcodierung und Syntaxfehleranzeige ab PowerShell 5.0 auch in der Konsole.

Syntaxfehler, also grammatikalische Fehler wie fehlende Klammern oder aus Sicht von PowerShell unsinnige Eingaben, werden in der ISE mit einer roten Wellenlinie unterschlängelt, und wenn Sie den Mauszeiger auf diese Linie bewegen und kurz warten, verrät ein Tooltippfenster, was mit der Eingabe noch nicht stimmt (Abbildung 1.14). In der Konsole zeigt eine rote spitze Klammer im Eingabeprompt Syntaxfehler an.

Abbildung 1.14: Syntaxfehler werden rot unterschlängelt, und PowerShell nennt die Fehlerursache.

Über *Tools/Optionen* lassen sich in der ISE die Farbzuweisungen im Zweig *Skriptbereichtoken* und *Konsolentoken* einsehen und auch ändern.

Rechnen mit PowerShell

PowerShell unterstützt alle Grundrechenarten, und ein Ausgabebefehl ist überflüssig, sodass Sie Ihre Rechenaufgaben direkt in die Konsole eingeben können:

```
PS> 100 * 58 / 5.9 ↵
983,050847457627
```

Wenn Sie genau hinschauen, werden Sie entdecken: Bei der Codeeingabe verwendet PowerShell als Dezimaltrennzeichen ausschließlich den Punkt. Das ist wichtig, damit Code länderübergreifend ausgeführt werden kann. Bei der Textausgabe der Rechenergebnisse wird als Dezimaltrennzeichen dagegen das in den regionalen Einstellungen Ihres Landes vorgesehene Zeichen benutzt – in Deutschland also ein Komma.

Hinweis

Das Komma hat bei der Codeeingabe ebenfalls eine feste Bedeutung: Es bildet Listen (oder in Programmierdeutsch: Arrays beziehungsweise Felder):

```
PS> 1,2,3 ↵
1
2
3

PS> 1,2,3 * 3 ↵
1
2
3
1
2
3
1
2
3
```

Auch die Farbcodierung Ihrer Eingaben sollten Sie im Blick behalten. Operatoren wie das Pluszeichen (+), das Minuszeichen (-) oder das Komma erscheinen stets in Grau. Zahlen haben eine eigene Farbe, und auch Texte in Anführungszeichen werden mit separater Farbe markiert:

```
PS> 'Hallo' * 10
```

Runde Klammern funktionieren genau wie in der Mathematik, es werden zuerst die Anweisungen in den Klammern ausgeführt:

```
PS> 3+5*10 ↵
53

PS> (3+5)*10 ↵
80
```

Profitipp

Runde Klammern sind bei PowerShell nicht auf Rechenoperationen beschränkt. Man kann sie überall dort einsetzen, wo es Missverständnisse bezüglich der Ausführungsreihenfolge geben könnte. Alles, was in runden Klammern steht, wird zuerst ausgeführt, und dann wird an dieser Stelle mit dem Ergebnis weitergearbeitet.

Vielleicht erinnern Sie sich an eines der ersten Beispiele in diesem Kapitel – darin wurde das Ergebnis von *ipconfig.exe* in einer Variablen gespeichert, und diese wurde dann mit dem Operator `-like` gefiltert, um nur Zeilen auszugeben, in denen »IPv4-Adresse« steht.

Mithilfe der runden Klammern kann man dies auch ohne Variablen in einem Ausdruck formulieren. Allerdings sind solche Ausdrücke nicht besonders leserlich:

```
PS C:\> (ipconfig.exe) -like '*IPv4*' ↵
   IPv4-Adresse  . . . . . . . . . . . : 192.168.2.119
```

Umwandlungen

Mit dem Präfix `0x` lassen sich hexadezimale Notationen kennzeichnen und auf diese Weise automatisch in ihren Dezimalwert umwandeln:

```
PS> 0xff ↵
255
```

Die in der IT üblichen Größenordnungen wie KB, MB, GB, TB und PB dürfen ebenfalls eingesetzt werden, wenn sie einer Zahl ohne Leerzeichen folgen:

```
PS> 1MB ↵
1048576

PS> 8.9TB ↵
9785653487206,4

PS> 0x8eKB ↵
145408
```

Hat sich dennoch ein Leerzeichen zwischen Zahl und Einheit geschmuggelt, ändern sich sofort die Farben. In der Konsole erscheint der folgende Ausdruck in Weiß, repräsentiert also eine Zahl:

```
PS> 1MB
```

Mit Leerzeichen wird die Zahl `1` weiterhin in Weiß angezeigt, aber `MB` ist nun gelb, also in der Farbe für die Befehle. Für PowerShell sieht die Eingabe nun so aus, als würden Sie eine Zahl und dann einen Befehl eingeben wollen:

```
PS> 1 MB
```

Drücken Sie ↵ und schicken diese Eingabe zu PowerShell, kommt es jetzt zu einem Fehler:

```
PS C:\> 1 MB ↵
In Zeile:1 Zeichen:3
+ 1 MB
```

```
+   ~~
Unerwartetes Token "MB" in Ausdruck oder Anweisung
    + CategoryInfo          : ParserError: (:)
   RecordException
    + FullyQualifiedErrorId : UnexpectedToken
```

Der Fehler war sogar farblich vorhersehbar, denn die Konsole färbt das >-Zeichen der Eingabeaufforderung rot ein, solange die Eingabe Syntaxfehler enthält, also Verstöße gegen die PowerShell-Grammatik.

Die Fehlermeldung beklagt sich völlig zu Recht darüber, dass das »Token« (der Sprachbestandteil also) namens `MB` an dieser Stelle keinen Sinn ergibt. Es ist schlichtweg sprachlich nicht zulässig, einer Zahl direkt einen Befehl folgen zu lassen, und so wäre die gleiche Fehlermeldung erschienen, wenn Sie die folgende etwas offensichtlicher inkorrekte Eingabe abschickten:

```
PS> 12 notepad ↵
```

Zahlenreihen

Zahlenreihen werden über den Operator `..` (bestehend aus zwei Punkten) erzeugt und können – jedenfalls von fortgeschrittenen Anwendern – über Typkonvertierungen auch in andere Datentypen verwandelt werden. Im folgenden Beispiel wird eine Zahlenfolge von 65 bis 90 generiert, was den ASCII-Codes der Buchstaben »A« bis »Z« entspricht. Durch die Umwandlung in ein Array vom Typ `Char` (einzelnes Zeichen) entsteht daraus eine Buchstabenliste:

```
PS> 65..90 ↵
PS> [Char[]](65..90) ↵
```

Sogar mit Texten – die stets in einfachen Anführungszeichen stehen – kann »gerechnet« werden:

```
PS> 'Hallo' + 'Welt' ↵
HalloWelt

PS> 'Hallo' * 10 ↵
HalloHalloHalloHalloHalloHalloHalloHalloHalloHallo
```

Unvollständige und mehrzeilige Eingaben

Mitunter kann die PowerShell-Konsole in einen Lähmungszustand fallen. Dieser tritt nur in der klassischen Konsole auf. Die moderne ISE ist dagegen immun.

Ist der Lähmungszustand eingetreten, führt die PowerShell-Konsole einfach keinen Befehl mehr aus. Dieser Zustand ist an einem klaren Symptom erkennbar: Die Eingabeaufforderung ändert sich und zeigt nun statt eines Pfadnamens an: >>. Ursache ist der primitive Mehrzeilenmodus der Konsole, der in diesem Fall aktiviert worden ist. PowerShell versteht Ihre Eingaben darin nicht mehr als einzelne Zeilen, sondern als Teil eines mehrzeiligen Texts – was die Frage aufwirft, wie (und warum) dieser Mehrzeilenmodus überhaupt aktiviert wurde und wie man wieder heil aus ihm herauskommt.

Aktiviert wird der Modus immer dann, wenn das, was Sie eingeben, noch nicht vollständig ist. Geben Sie beispielsweise einen Text in Anführungszeichen ein, ohne die abschließenden

Anführungszeichen anzufügen, geht die PowerShell-Konsole davon aus, dass der Text in der nächsten Zeile fortgesetzt werden soll – und aktiviert von ganz allein den Mehrzeilenmodus.

```
"Hallo ↵
>> Dies ist mein kleines Tagebuch. ↵
>> Ich schreibe jetzt einen mehrseitigen Text ↵
>> Das geht so lange, bis ich die Lust verliere." ↵
>>↵

Hallo
Dies ist mein kleines Tagebuch.
Ich schreibe jetzt einen mehrseitigen Text
Das geht so lange, bis ich die Lust verliere.
```

Erst wenn Sie das Abschlusszeichen eingeben (und dann noch zweimal ↵ drücken), akzeptiert PowerShell die mehrzeilige Eingabe. Sinnvoll ist das Ganze im Alltag nur in wenigen Ausnahmefällen und führt viel häufiger zu Irritationen. Falls PowerShell also plötzlich nicht mehr auf Ihre Befehle zu reagieren scheint und der verräterische >>-Prompt auftaucht, drücken Sie am besten Strg+C und brechen ab. Drücken Sie danach auf ↑, um die letzte Eingabe zurückzubekommen, und vervollständigen Sie Ihre Eingabe.

In der ISE kann der Mehrzeilenmodus nicht auftreten, weil es ihn gar nicht gibt. Die ISE ist ja bereits ein vollwertiger Skripteditor, und Sie bräuchten lediglich den Skriptbereich (zum Beispiel über Strg+R) einzublenden, um komfortabel mehrzeiligen Text und Code zu erfassen. Deshalb quittiert die Konsole in ISE fehlende Anführungszeichen und andere paarweise vorkommende Sonderzeichen mit einem aussagekräftigen Fehler:

```
PS> "Hallo ↵

Die Zeichenfolge hat kein Abschlusszeichen: ".
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : TerminatorExpectedAtEndOfString
```

Profitipp

Tatsächlich enthält die Konsole von ISE einen (versteckten) Mehrzeilenmodus. Möchten Sie eine weitere Zeile erfassen, drücken Sie ⇧+↵. Eine neue leere Zeile erscheint. Damit sind auch in ISE Eingaben wie diese möglich:

```
PS> "Hallo ⇧+↵
dies ist ein mehrzeiliger Text ⇧+↵
mit UMSCHALT+¢ lassen sich weitere Zeilen hinzufügen ⇧+↵
Auch hier muss der Text am Ende ordentlich mit einem Anführungszeichen abgeschlossen werden" ↵

Hallo
dies ist ein mehrzeiliger Text
mit UMSCHALT+¢ lassen sich weitere Zeilen hinzufügen
Auch hier muss der Text am Ende ordentlich mit einem Anführungszeichen abgeschlossen werden
```

Skriptausführung erlauben

Anfangs erlaubt die PowerShell ausschließlich die Ausführung interaktiver Befehle. Um auch PowerShell-Skripte ausführen zu können, müssen Sie dem einmalig zustimmen.

Dies sollten Sie schon jetzt tun, auch wenn Sie vielleicht noch gar keine eigenen PowerShell-Skripte schreiben möchten. PowerShell ist nämlich erweiterbar und lädt zusätzliche Befehle – sofern vorhanden – automatisch nach. Viele dieser Befehle werden von Skripten bereitgestellt, und solange die Skriptausführung verboten ist, funktionieren solche Befehle nicht. Mit `Set-ExecutionPolicy` wird die Skriptausführung erlaubt:

```
PS> Set-ExecutionPolicy –Scope CurrentUser –ExecutionPolicy RemoteSigned –Force ↵
```

Diese Änderung gilt nur für Ihr eigenes Benutzerkonto und bleibt so lange wirksam, bis Sie die Einstellung erneut ändern. Besondere Administratorrechte sind dafür nicht erforderlich. Allerdings lohnt sich ein Blick auf die übrigen Ausführungsrichtlinien, die es sonst noch gibt:

```
PS> Get-ExecutionPolicy –List ↵

        Scope ExecutionPolicy
        ----- ---------------
MachinePolicy       Undefined
   UserPolicy       Undefined
      Process       Undefined
  CurrentUser    RemoteSigned
 LocalMachine       Undefined
```

PowerShell bestimmt die effektive Einstellung, indem es die fünf Richtlinien von oben nach unten prüft. Die erste Einstellung, die nicht `Undefined` lautet, wird wirksam. Sind alle Einstellungen auf `Undefined` gesetzt, wird die Skriptausführung verboten. Das ist der Ausgangszustand der PowerShell, und die aktuell effektive Einstellung liefert `Get-ExecutionPolicy`, wenn Sie den Parameter `-List` nicht angeben.

Besonderes Augenmerk verdienen die ersten beiden Richtlinien: `MachinePolicy` und `UserPolicy` werden zentral über Gruppenrichtlinien festgelegt. Sie können diese Einstellungen nicht manuell ändern. Da es die obersten beiden Einstellungen sind, haben sie Vorrang vor allen übrigen. Wenn an diesen obersten beiden Stellen also Vorgaben zu sehen sind, können Sie zwar darunter eigene Einstellungen treffen, doch werden diese niemals wirksam.

Grundsätzlich ist es eine schlechte Idee, die Ausführungsrichtlinie über zentrale Gruppenrichtlinien zwingend vorzuschreiben, denn die Ausführungsrichtlinie ist nicht dazu gedacht, Sie vor Angreifern zu schützen. Sie ist also auch kein Teil einer Unternehmenssicherheitsstrategie. Die Ausführungsrichtlinie ist ein *persönlicher* Sicherheitsgurt, der Sie selbst vor Ihren *eigenen* Fehlern schützt. Die Einstellung `RemoteSigned` besagt zum Beispiel, dass Skripte, die aus »feindlichem« Territorium stammen und nicht mit einer gültigen digitalen Signatur versehen sind, nicht ausgeführt werden können.

Unter »feindlichem Territorium« versteht man Skripte, die aus dem Internet heruntergeladen oder als E-Mail-Anhang empfangen wurden, und solche, die auf Netzlaufwerken lagern, die nicht zur eigenen Domäne (beziehungsweise zur Internetzone der vertrauenswürdigen Sites) zählen.

So schützt Sie `RemoteSigned` also davor, potenziell gefährliche Skripte aus unbekannten Quellen auszuführen, während Sie eigene Skripte von der lokalen Festplatte aus starten können.

Wählen Sie stattdessen die Einstellung `Unrestricted`, erhielten Sie bei Skripten aus zweifelhaften Quellen eine Warnung, könnten sich aber darüber hinwegsetzen und das Skript trotzdem ausführen.

Die Einstellung `Bypass` schließlich würde alle Skripte unabhängig von ihrer Herkunft sofort und ohne Rückfragen ausführen. Sie kann zum Beispiel für Unternehmensadministratoren sinnvoller sein als die zuvor genannten, weil man damit Skripte beliebiger Herkunft auch unbeaufsichtigt ausführen kann.

Tippfehler vermeiden und Eingaben erleichtern

PowerShell ist textbasiert, und jedes Zeichen eines Befehls ist gleich wichtig. Fehlt eines, funktioniert gar nichts. Deshalb sind Tippfehler in PowerShell so wie in jeder anderen textbasierten Skriptsprache die häufigste Ursache für Frustration. Dagegen hilft nur konsequente Tippfaulheit, denn wer weniger tippt, tippt auch weniger verkehrt – und schneller geht es außerdem. Lassen Sie daher PowerShell so viel wie möglich von der Tipparbeit übernehmen.

Autovervollständigung

Die Autovervollständigung kann begonnene Eingaben vervollständigen und ist auch eine wichtige Kontrollinstanz: Falls die Autovervollständigung keine Resultate liefert, liegt vielleicht bereits ein Tippfehler in dem Text vor, den Sie bis dahin eingegeben haben. Die Autovervollständigung kann etwa Pfadnamen vervollständigen wie beispielsweise hier:

```
PS> C:\p⇥
```

Bei jedem Druck auf ⇥ schlägt PowerShell jetzt einen neuen Ordner oder eine neue Datei vor, der beziehungsweise die mit `C:\p` beginnt. Je mehr Zeichen Sie selbst eingeben, desto weniger Auswahlmöglichkeiten werden angeboten, und in der Praxis sollten Sie wenigstens drei Zeichen eintippen, um nicht lästig viele Vorschläge zu erhalten.

Pfadnamen vervollständigen

Enthält ein Pfadname Leerzeichen, stellt die Autovervollständigung den Pfad automatisch in Anführungszeichen. Wollen Sie sich in einen Unterordner vortasten, genügt es, hinter dem abschließenden Anführungszeichen einen weiteren `\` anzufügen und ⇥ zu drücken – schon geht die Autovervollständigung weiter:

```
PS> & 'C:\Program Files\Common Files'\ ⇥
```

Hat die Autovervollständigung einen Pfadnamen in Anführungszeichen gesetzt, fügt sie außerdem am Zeilenanfang ein `&` ein. Dieser spezielle sogenannte »Call«-Operator sorgt dafür, dass sich der Text in Anführungszeichen wie ein Befehl verhält, also genau so, als wäre ein Pfad ohne Anführungszeichen geschrieben worden.

Und warum? Weil Text in Anführungszeichen andernfalls eben nichts weiter ist als genau das: Text. Er würde ohne `&` einfach nur kommentarlos wieder ausgegeben. Sogar Platzhalterzeichen sind in Pfadnamen erlaubt. Geben Sie zum Beispiel `C:\pr*m` ⇥ ein, schlägt PowerShell den Ordner *C:\Program Files* vor.

Grundsätzlich kann der Call-Operator auch dafür verwendet werden, beliebige Befehle auszuführen, die als Text vorliegen. Dies ist ein ungewöhnliches Beispiel:

```
PS> $a = 'not' ↵
PS> $b = 'AD' ↵
PS> $c = 'eP' ↵
PS> & "$a$c$b" ↵
```

Hier wird der Windows-Editor Notepad gestartet, denn die Variablen ergeben in richtiger Reihenfolge einfach nur diesen Text, den der Call-Operator dann genau so ausführt, als hätten Sie ihn direkt eingegeben:

```
PS> "$a$c$b" ↵
notePAD
```

Dieses Beispiel verdeutlicht nebenbei eine andere PowerShell-Funktionalität: Variablen, die in normalen Anführungszeichen stehen, werden automatisch durch ihren Inhalt ersetzt. Achten Sie auf die Farben, wenn Sie das obige Beispiel eingeben: Die Variablen innerhalb des Texts erscheinen in einer anderen Farbe als die Anführungszeichen.

Verwenden Sie dagegen einzelne Anführungszeichen, wird der Text von PowerShell nicht verändert und erscheint immer genau so wie angegeben. Auch hier weisen die Farben den Weg: Die Variablen innerhalb der Anführungszeichen haben die gleiche Farbe wie die Anführungszeichen.

```
PS> '$a$c$b' ↵
$a$c$b
```

Befehlszeilen erneut verwenden

Auch die Befehlshistorie spart Tipparbeit. Oft sitzt der erste eingegebene Befehl nicht auf Anhieb richtig, und Sie erhalten eine Fehlermeldung, oder der Befehl macht (noch) nicht das, was Sie sich eigentlich vorgestellt haben. Wenn Sie an Ihrem Befehl ein wenig feilen und ihn verbessern wollen, brauchen Sie ihn nicht komplett neu einzugeben.

Drücken Sie stattdessen ↑, um den zuletzt eingegebenen Befehl zurückzuholen. Danach können Sie diesen Befehl verändern oder verbessern, bevor Sie ihn mit ↵ erneut an PowerShell senden. Drücken Sie ↑ mehrmals, wenn Sie vorvorherige oder noch ältere Eingaben erneut verwenden wollen. Mit ↓ wandern Sie in der Liste wieder zurück.

Befehlsnamen autovervollständigen

Geben Sie den Anfang eines Befehls ein und drücken ⇆, wird sein Name vervollständigt. Bei Cmdlets, die stets aus einem Doppelnamen bestehen, funktioniert das besonders dann schnell und zielgerichtet, wenn Sie zunächst den ersten Namensteil angeben, dann den Bindestrich hinzufügen und danach vom zweiten Namensteil zumindest drei Buchstaben eingeben.

Jedes Mal, wenn Sie danach ⇆ drücken, wird Ihnen ein neuer infrage kommender Befehl vorgeschlagen. Im ISE-Editor funktioniert dies dank der IntelliSense-Menüs sehr viel intuitiver, und ISE öffnet nach dem Bindestrich sofort das IntelliSense-Menü, das mit jedem weiteren eingegebenen Zeichen seine Auswahl weiter einschränkt.

Parameter-Autovervollständigung

Die meisten Cmdlets erwarten Zusatzinformationen von Ihnen, die Sie über Parameter eingeben. Jeder Parameter beginnt mit einem Bindestrich. Sobald Sie also hinter einen Cmdlet-Namen einen Bindestrich setzen, würde ⇆ Ihnen die verfügbaren Parameter vorschlagen. Der ISE-Editor verrichtet dies mit seinem IntelliSense-Menü vollautomatisch (Abbildung 1.15).

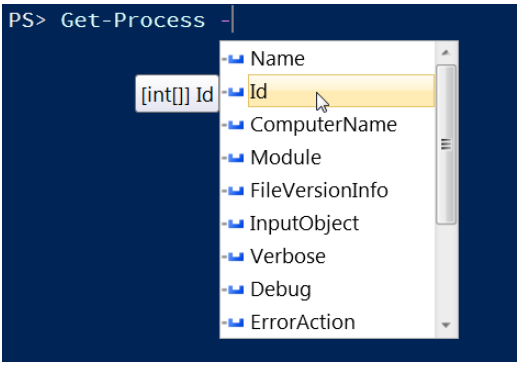


Abbildung 1.15: Parameternamen werden im ISE-Editor vorgeschlagen.

Argument-Autovervollständigung

Häufig kann PowerShell sogar Vorschläge dazu machen, was ein Parameter von Ihnen verlangt. Wieder werden die Vorschläge per ⇆ angefordert oder in ISE automatisch als IntelliSense-Menü vorgeschlagen (Abbildung 1.16).

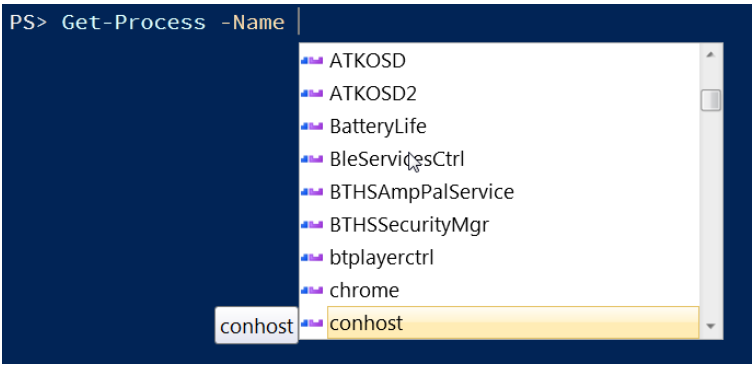


Abbildung 1.16: Sogar Argumente für einzelne Parameter werden von PowerShell häufig vervollständigt.

Die Argument-Autovervollständigung ist neu ab PowerShell 3.0. Sie hat natürliche Grenzen: Falls ein Parameter die erwartete Eingabe nicht eingegrenzt hat, sondern beliebigen Text akzeptiert, schaltet PowerShell als Notlösung um in die Pfadnamen-Autovervollständigung. Ob Pfadnamen für den Parameter wirklich sinnvoll sind, hängt vom Parameter ab und steht auf einem anderen Blatt. Mit Esc können Sie das IntelliSense-Menü jederzeit zuklappen, sollte es Ihnen keine hilfreichen Vorschläge liefern. Und falls Sie es einmal zu schnell zugeklappt haben, öffnet Strg + Leertaste es jederzeit wieder.

PowerShell-Hilfe aus dem Internet nachladen

PowerShell und seine Cmdlets sind gut dokumentiert, allerdings müssen die Hilfetexte für PowerShell (einmalig) mit `Update-Help` aus dem Internet nachgeladen werden. Leider speichert PowerShell seine Hilfedateien im Windows-Ordner, und der darf nur von Administratoren geändert werden. Deshalb kann die PowerShell-Hilfe nur aktualisiert werden, wenn Sie den folgenden Befehl aus einer PowerShell mit vollen Administratorrechten heraus ausführen (und über eine Internetverbindung verfügen):

```
PS> Update-Help -UICulture En-US -Force ↵
```

Der Parameter `-Force` sorgt dafür, dass die Hilfe auch tatsächlich heruntergeladen wird, denn ohne ihn würde der Befehl nur einmal in 24 Stunden die Hilfe abrufen. Hatten Sie dann den Befehl schon einmal innerhalb der letzten 24 Stunden gegeben, tut sich nichts. Der Parameter `-UICulture En-US` fordert ausdrücklich die englischsprachige Hilfe an. Ohne ihn würden Sie auf deutschen Systemen nur die deutsche Hilfe erhalten, und was verlockend klingt, wäre nicht allzu hilfreich: Für die meisten Befehle gibt es nur englischsprachige Hilfe.

Sobald Sie die ↵-Taste gedrückt haben, sehen Sie, wie PowerShell versucht, den Updateserver zu erreichen, und von ihm dann die nötigen Hilfetexte empfängt (Abbildung 1.17). Nach wenigen Sekunden sollte die Hilfe auf Ihrem Computer einsatzbereit sein. Ob das der Fall ist, überprüfen Sie anschließend am besten mit folgendem Befehl:

```
PS> Get-Help Get-Help -ShowWindow ↵
```

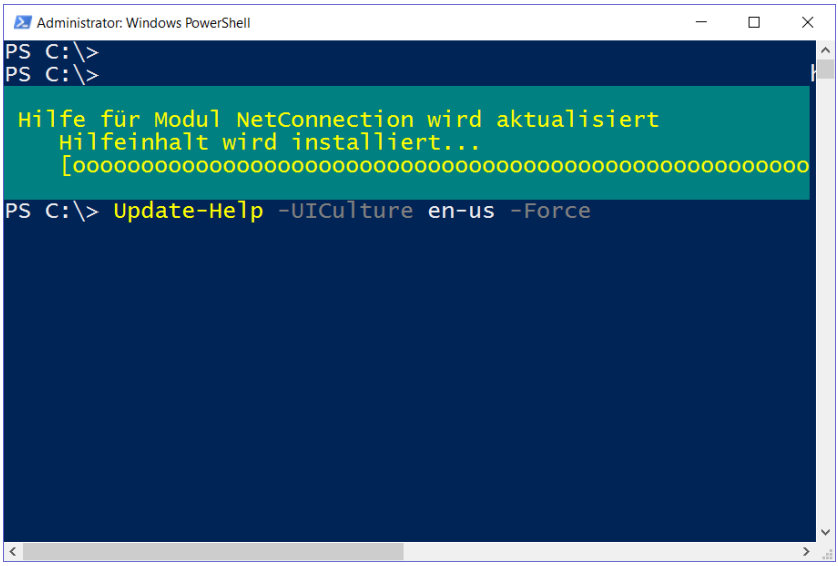


Abbildung 1.17: PowerShell-Hilfe aus dem Internet herunterladen und installieren.

Ist alles in Ordnung, sollte sich ein Hilfefenster öffnen und Ihnen erklären, was der Befehl `Get-Help` für Sie tun kann. Sie rufen quasi die Hilfe für die Hilfe ab, könnten nun aber natürlich auch jeden anderen Befehl nachschlagen, zu dem Sie Fragen haben:

```
PS> Get-Help Stop-Service -ShowWindow ↵
```

Tipp

Im ISE-Editor genügt es, auf einen fraglichen Befehl zu klicken und dann F1 zu drücken. Die ISE schreibt daraufhin den entsprechenden Aufruf von `Get-Help` automatisch ins Konsolenfenster und führt ihn aus.

Achten Sie aber darauf, wirklich nur auf den fraglichen Befehl zu klicken und keinen Text zu markieren.

Hilfe ohne Internetzugang installieren

Wie Sie sehen, liegen die Hilfedateien eigentlich in Dateiform im Windows-Ordner. Jetzt wird auch klar, wie man Systeme ohne eigenen Internetzugang mit der PowerShell-Hilfe ausstatten kann: Laden Sie die Hilfedateien auf einen anderen Computer herunter und kopieren Sie sie dann auf den Computer ohne Internetzugang.

PowerShell bietet hierzu das Cmdlet `Save-Help` an: Es lädt die Hilfedateien genau wie `Update-Help` herunter, speichert sie aber in den Ordner, den Sie angeben. Dieser Ordnerinhalt kann dann auf einen Computer kopiert werden, der über keinen eigenen Internetzugang verfügt. Dort ruft man `Update-Help` mit dem Parameter `-Source` auf und gibt den Pfad zu den bereits vorhandenen Hilfedateien an.

Falls Sie im Unternehmen selbst Betriebssysteme verteilen und installieren, haben Sie außerdem natürlich die Möglichkeit, die Hilfedateien bereits ins Image des Betriebssystems mit aufzunehmen.