

O'REILLY®

2. Auflage

Dr. Tobias Weltner

# Windows PowerShell

Grundlagen und Scripting-Praxis  
für den Einstieg

Mit  
vielen  
Beispielen  
und  
Übungen



#### Copyright und Urheberrechte:

Die durch die dpunkt.verlag GmbH vertriebenen digitalen Inhalte sind urheberrechtlich geschützt. Der Nutzer verpflichtet sich, die Urheberrechte anzuerkennen und einzuhalten. Es werden keine Urheber-, Nutzungs- und sonstigen Schutzrechte an den Inhalten auf den Nutzer übertragen. Der Nutzer ist nur berechtigt, den abgerufenen Inhalt zu eigenen Zwecken zu nutzen. Er ist nicht berechtigt, den Inhalt im Internet, in Intranets, in Extranets oder sonst wie Dritten zur Verwertung zur Verfügung zu stellen. Eine öffentliche Wiedergabe oder sonstige Weiterveröffentlichung und eine gewerbliche Vervielfältigung der Inhalte wird ausdrücklich ausgeschlossen. Der Nutzer darf Urheberrechtsvermerke, Markenzeichen und andere Rechtsvorbehalte im abgerufenen Inhalt nicht entfernen.

# **Windows PowerShell – Grundlagen und Scripting-Praxis für den Einstieg, 2. Auflage**

Dr. Tobias Weltner

**O'REILLY®**

Dr. Tobias Weltner

Lektorat: Ariane Hesse

Korrektorat: Sibylle Feldmann, [www.richtiger-text.de](http://www.richtiger-text.de)

Satz: Gerhard Alfes, [mediaService, www.mediaservice.tv](http://www.mediaservice.tv)

Herstellung: Stefanie Weidner

Umschlaggestaltung: Michael Oreal, [www.oreal.de](http://www.oreal.de), unter Verwendung eines Fotos von Kirillm/iStock by Getty Images

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 978-3-96009-192-9

PDF 978-3-96010-671-5

ePub 978-3-96010-672-2

mobi 978-3-96010-673-9

2. Auflage 2023

Copyright © 2023 dpunkt.verlag GmbH

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: [komentar@oreilly.de](mailto:komentar@oreilly.de).

Die vorliegende Publikation ist urheberrechtlich geschützt. Alle Rechte vorbehalten. Die Verwendung der Texte und Abbildungen, auch auszugsweise, ist ohne die schriftliche Zustimmung des Verlags urheberrechtswidrig und daher strafbar. Dies gilt insbesondere für die Vervielfältigung, Übersetzung oder die Verwendung in elektronischen Systemen. Es wird darauf hingewiesen, dass die im Buch verwendeten Soft- und Hardware-Bezeichnungen sowie Markennamen und Produktbezeichnungen der jeweiligen Firmen im Allgemeinen warenzeichen-, marken- oder patentrechtlichem Schutz unterliegen. Alle Angaben in diesem Buch wurden mit größter Sorgfalt kontrolliert. Weder Autor noch Verlag können jedoch für Schäden haftbar gemacht werden, die in Zusammenhang mit der Verwendung dieses Buches stehen.

5 4 3 2 1 0

# Inhalt

|   |     |
|---|-----|
| <b>Vorwort zur zweiten Auflage</b> .....            | 9   |
| <b>Einleitung</b> .....                             | 11  |
| <b>1 PowerShell startklar machen</b> .....          | 15  |
| Mit PowerShell Kontakt aufnehmen .....              | 16  |
| Die PowerShell-Konsole einrichten .....             | 18  |
| PowerShell ISE einsetzen .....                      | 22  |
| VSCode und PowerShell .....                         | 25  |
| Erste Schritte mit PowerShell .....                 | 26  |
| IntelliSense-Unterstützung im ISE-Editor .....      | 31  |
| Skriptausführung erlauben .....                     | 38  |
| Wichtige Befehle aktualisieren .....                | 39  |
| Tippfehler vermeiden und Eingaben erleichtern ..... | 40  |
| PowerShell-Hilfe aus dem Internet nachladen .....   | 42  |
| <b>Teil A Erste Befehle</b> .....                   | 45  |
| <b>2 Cmdlets – die PowerShell-Befehle</b> .....     | 47  |
| Alles, was Sie über Cmdlets wissen müssen .....     | 48  |
| Cmdlets für eine Aufgabe finden .....               | 49  |
| Mit Parametern Wünsche formulieren .....            | 64  |
| Neue Cmdlets aus Modulen nachladen .....            | 80  |
| Alias: Zweitname für Cmdlets .....                  | 84  |
| <b>3 PowerShell-Laufwerke</b> .....                 | 89  |
| Dateisystemaufgaben meistern .....                  | 91  |
| Umgebungsvariablen .....                            | 108 |
| Windows-Registrierungsdatenbank .....               | 110 |
| Virtuelle Laufwerke und Provider .....              | 115 |
| <b>4 Anwendungen und Konsolenbefehle</b> .....      | 121 |
| Programme starten .....                             | 123 |
| Argumente an Anwendungen übergeben .....            | 128 |
| Ergebnisse von Anwendungen weiterverarbeiten .....  | 130 |
| Laufende Programme steuern .....                    | 137 |

|               |   |     |
|---------------|---|-----|
| <b>Teil B</b> | <b>Aufgaben automatisieren</b>                | 141 |
| <b>5</b>      | <b>PowerShell-Skripte</b>                     | 143 |
|               | PowerShell-Skripte verfassen                  | 144 |
|               | Profilskripte – die Autostartskripte          | 147 |
|               | Skripte außerhalb von PowerShell starten      | 149 |
| <b>6</b>      | <b>Die PowerShell-Pipeline</b>                | 155 |
|               | Aufbau der PowerShell-Pipeline                | 156 |
|               | Select-Object                                 | 159 |
|               | Where-Object                                  | 168 |
|               | Sort-Object                                   | 173 |
|               | ForEach-Object                                | 176 |
|               | Group-Object                                  | 177 |
|               | Measure-Object                                | 180 |
|               | Mit »berechneten« Eigenschaften arbeiten      | 181 |
| <b>Teil C</b> | <b>Operatoren nutzen</b>                      | 189 |
| <b>7</b>      | <b>Operatoren und Bedingungen</b>             | 191 |
|               | Operatoren – Aufbau und Namensgebung          | 192 |
|               | Zuweisungsoperatoren                          | 194 |
|               | Vergleichsoperatoren                          | 196 |
|               | Bedingungen                                   | 203 |
| <b>8</b>      | <b>Textoperationen und reguläre Ausdrücke</b> | 207 |
|               | Texte zusammenfügen                           | 208 |
|               | Textstellen finden und extrahieren            | 214 |
|               | Reguläre Ausdrücke: Textmustererkennung       | 218 |
|               | Textstellen ersetzen                          | 225 |
|               | Split und Join: eine mächtige Strategie       | 225 |
| <b>Teil D</b> | <b>Betriebssystem-Funktionen einsetzen</b>    | 229 |
| <b>9</b>      | <b>Mit Objekten arbeiten</b>                  | 231 |
|               | Eigenschaften und Methoden                    | 234 |
|               | Eigenschaften und Methoden anzeigen           | 240 |
|               | Ergebnisse eines Befehls untersuchen          | 242 |
|               | Eigenschaften lesen                           | 248 |
|               | Eigenschaften ändern                          | 252 |
|               | Methoden aufrufen                             | 253 |

|  |         |
|--|---------|
| <b>10 Typen verwenden</b> .....                                | 257     |
| Typumwandlungen .....  | 258     |
| Neue Objekte durch Typumwandlungen .....                       | 263     |
| Implizite Umwandlung und typisierte Variablen .....            | 270     |
| Verborgene Befehle in Typen .....                              | 273     |
| Statische Methoden verwenden .....                             | 274     |
| Neue Objekte herstellen .....                                  | 285     |
| COM-Objekte verwenden .....                                    | 290     |
| Webdienste ansprechen .....                                    | 299     |
| Typen nachladen .....  | 300     |
| <br><b>Teil E Neue eigene Befehle erfinden</b> .....           | <br>303 |
| <b>11 Powershell-Funktionen</b> .....                          | 305     |
| Alles Wichtige: ein Überblick .....                            | 306     |
| Eine bessere Prompt-Funktion .....                             | 317     |
| Zwingend erforderliche Parameter .....                         | 317     |
| Argumente ohne Parameter .....                                 | 319     |
| Rückgabewerte festlegen .....                                  | 320     |
| <b>12 Pipeline-fähige Funktionen</b> .....                     | 323     |
| Anonyme Pipeline-Funktion .....                                | 324     |
| Parameter und Pipeline-Kontrakt .....                          | 330     |
| »HASA«-Kontrakt: Objekteigenschaften lesen .....               | 333     |
| Modularer Code mit Pipeline-fähigen Funktionen .....           | 338     |
| <b>13 Eigene Module erstellen</b> .....                        | 349     |
| Module sind Ordner .....                                       | 350     |
| Manifestdatei für ein Modul .....                              | 354     |
| <b>14 PowerShellGet – Module verteilen und nachladen</b> ..... | 359     |
| PowerShell Gallery nutzen .....                                | 361     |
| Privates Repository einrichten .....                           | 367     |
| <br><b>Teil F Fehlerbehandlung und Debugging</b> .....         | <br>371 |
| <b>15 Fehlerhandling</b> .....                                 | 373     |
| Fehlermeldungen unterdrücken .....                             | 374     |
| Fehlerhandler einsetzen .....                                  | 378     |

|                       |   |            |
|-----------------------|---|------------|
| <b>Teil G</b>         | <b>Mit Remoting quer durchs Netzwerk</b>        | <b>387</b> |
| <b>16</b>             | <b>Fernzugriff und Netzwerk-Troubleshooting</b> | <b>389</b> |
|                       | Klassische Fernzugriffe                         | 390        |
|                       | Troubleshooting für Fernzugriffe                | 392        |
| <b>17</b>             | <b>Windows PowerShell-Remoting</b>              | <b>399</b> |
|                       | PowerShell-Remoting aktivieren                  | 400        |
|                       | Erste Schritte mit PowerShell-Remoting          | 405        |
|                       | Remotefähigen Code entwickeln                   | 407        |
| <b>Index</b>          |   | <b>417</b> |
| <b>Über den Autor</b> |   | <b>424</b> |



# Vorwort zur zweiten Auflage

»Windows PowerShell« wurde 2006 das erste Mal ausgeliefert und 2016 als Bestandteil von Windows 10 und Server 2016 fertiggestellt.

Nach der Fertigstellung der in Windows integrierten »Windows PowerShell« begann Microsoft ab 2016 mit der Entwicklung einer »neuen« PowerShell, die zuerst PowerShell Core, dann PowerShell 6 und schließlich PowerShell 7 oder nur noch »PowerShell« hieß. Hauptsächlicher Vorzug der »neuen« PowerShell: Sie läuft nicht nur unter dem Betriebssystem Windows, sondern plattformübergreifend auch unter Linux und macOS.

Warum also die Neuauflage eines Buchs nur zur »Windows PowerShell«? Weil weniger manchmal genau richtig ist.

Fast alle Unternehmen, die Windows-Betriebssysteme und Microsoft-Services einsetzen, nutzen in ihrer Unternehmens-IT weiterhin die in Windows integrierte »Windows PowerShell«. Nur sie garantiert derzeit 100% Skriptkompatibilität und wird vollständig von Microsoft »supportet«.

Und anders als die »modernere« PowerShell 7 ändert sich die in Windows integrierte »Windows PowerShell« auch nicht mehr in unvorhersehbarer Weise mit dem Einspielen von Windows-Updates. Bei der »neuen« PowerShell 7 kann es nämlich durchaus passieren, dass nach dem neuesten Update ein Befehl (»Cmdlet«) anders oder gar nicht mehr funktioniert, auch wenn das eher selten vorkommt.

Wenn Sie im Windows-Umfeld arbeiten und ein solides PowerShell-Wissen für den produktiven Alltag aufbauen möchten, wird Ihnen dieses Buch sehr helfen. Sie werden in diesem Praxisbuch nur das finden, was im Windows-Umfeld tatsächlich funktioniert. Die neue umfassende Funktionalität von PowerShell 7 wird nicht behandelt. So laufen Sie nicht Gefahr, Skripte zu entwickeln, die anschließend im Unternehmensumfeld auf »Windows PowerShell« nicht oder nur fehlerhaft laufen.

Und wenn Sie später (oder parallel zu Ihrem beruflichen Einsatz) doch die neue PowerShell 7 testen, werden Sie feststellen: Alles, was Sie in diesem Buch lernen, findet sich auch in der »neuen« PowerShell wieder, die schließlich die logische Weiterentwicklung der »Windows PowerShell« ist.

## **Vorwort zur zweiten Auflage**

Da der Funktionsumfang der »Windows PowerShell« unverändert geblieben ist, enthält diese Neuauflage keine neuen Themen oder Kapitel. Der Text wurde komplett durchgesehen, kleinere Fehler wurden berichtigt und einige didaktisch motivierte Ergänzungen vorgenommen.

Herzlichst Ihr

Dr. Tobias Weltner

# 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 Automationskripte entstehen, mit denen sich vielfältige Aufgaben vollständig automatisieren lassen.

## Einleitung

**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, senden Sie mir eine Nachricht an meine Mailadresse *tobias@powershell.one*.

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:

|   |    |
|---|----|
| Mit PowerShell Kontakt aufnehmen .....              | 16 |
| Die PowerShell-Konsole einrichten .....             | 18 |
| PowerShell ISE einsetzen .....                      | 22 |
| VSCode und PowerShell .....                         | 25 |
| Erste Schritte mit PowerShell .....                 | 26 |
| IntelliSense-Unterstützung im ISE-Editor .....      | 31 |
| Skriptausführung erlauben .....                     | 38 |
| Tippfehler vermeiden und Eingaben erleichtern ..... | 40 |
| PowerShell-Hilfe aus dem Internet nachladen .....   | 42 |

**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 Automationsssprache, 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 7 auf Linux und macOS

PowerShell steht inzwischen in Version 7 betriebssystemunabhängig zur Verfügung. Diese PowerShell ist eine »portable Anwendung« und kann auf Windows-Systemen auch parallel zur eingebauten »Windows PowerShell« verwendet werden.

Informationen zum Download und der Installation finden Sie unter <https://docs.microsoft.com/de-de/powershell/scripting/install/installing-powershell>, oder falls sich dieser Link inzwischen geändert haben sollte, unter dem »ewigen« Perma-Link <https://aka.ms/pscore6>, der auch beim Start in der Konsole der »Windows PowerShell« angezeigt wird.

Windows PowerShell ist keineswegs veraltet oder »abgekündigt«, und so besteht im Windows-Umfeld auch kein Druck, auf die neue PowerShell 7 umzusteigen. In erster Linie dient sie dazu, PowerShell in heterogenen Umgebungen auch auf anderen Betriebssystemen einzusetzen.

Allerdings merkt man PowerShell 7 in manchen Bereichen durchaus an, dass sie seit nunmehr sechs Jahren weiterentwickelt wurde: Manche Vorgänge werden von der neuen PowerShell 7 deutlich schneller verarbeitet als in Windows PowerShell.

Deshalb kann es durchaus Sinn machen, neben der Windows PowerShell auch auf Windows-Systemen parallel die neue PowerShell 7 zu installieren und auszuprobieren.

Die PowerShell-Funktionen, die Sie in diesem Buch kennenlernen, funktionieren in den allermeisten Fällen unverändert auch in PowerShell 7. Es kann höchstens sein, dass einzelne Befehle in PowerShell 7 weggefallen sind.

## 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 7 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 **Windows + R** das *Ausführen*-Fenster und geben darin ein: `powershell` **↵**.

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

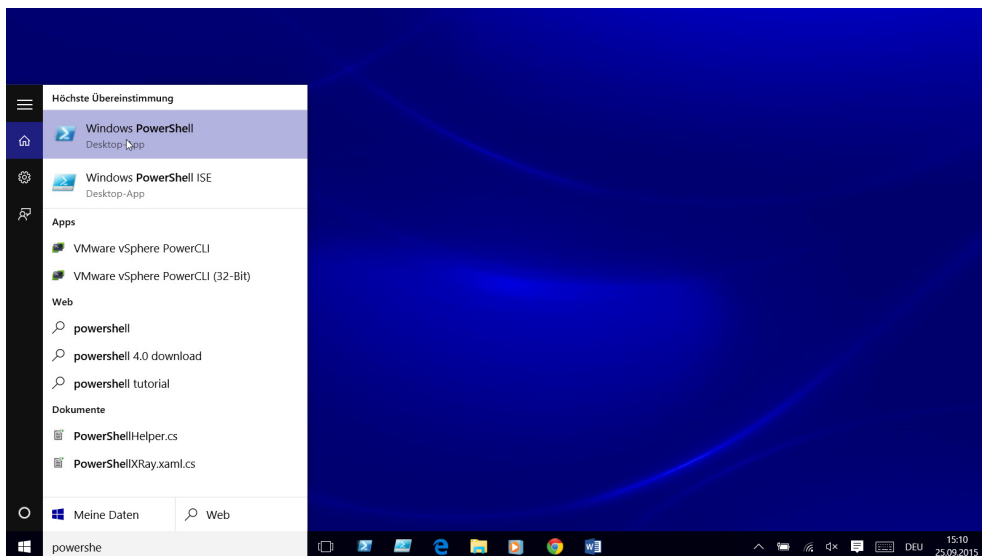


Abbildung 1.1: PowerShell über das Startmenü in Windows 10 öffnen.

Wenige Sekunden später präsentiert sich die 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).

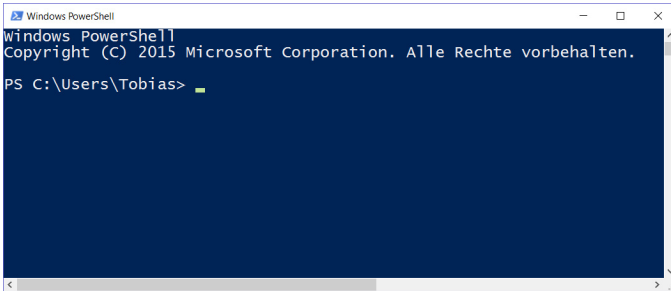


Abbildung 1.2: Die PowerShell-Konsole führt interaktive PowerShell-Befehle aus und benötigt wenig Speicher.

## 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 5.1, was Sie inspirieren sollte, Ihre PowerShell-Version ebenfalls auf Versionsstand 5.1 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.                               |

Tabelle 1.1: Windows 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
```

## Kapitel 1: PowerShell startklar machen

|                           |                         |
|---------------------------|-------------------------|
| 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 plattformübergreifende PowerShell 7.

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

## Aktuelle Version der PowerShell nutzen

Weil dieses Buch auf der aktuellen Version der Windows PowerShell, PowerShell 5.1, basiert, sollten Sie gegebenenfalls 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 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.

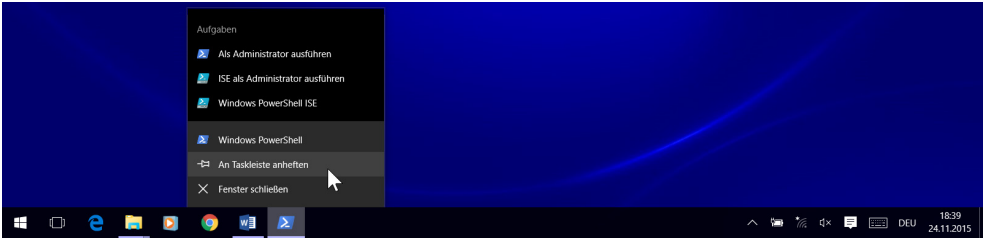


Abbildung 1.3: PowerShell-Symbol an die Taskleiste anheften.

Sie können PowerShell nun auch über die Tastenkombination **Windows + 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 Bildschirmgröße 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 14).

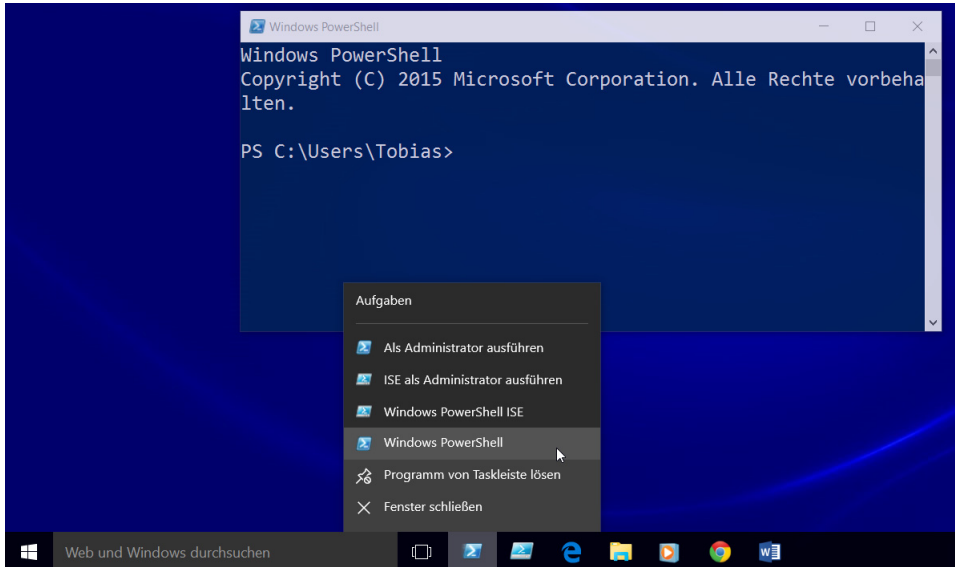


Abbildung 14: Sprungliste von PowerShell öffnen.

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.

## Kapitel 1: PowerShell startklar machen

| Befehl                                 | Beschreibung   |
|--|--|
| <i>Als Administrator ausführen</i>     | Öffnet die interaktive PowerShell-Konsole mit allen Rechten.     |
| <i>ISE als Administrator ausführen</i> | Öffnet den PowerShell-Editor mit allen Rechten.                  |
| <i>Windows PowerShell ISE</i>          | Öffnet den integrierten PowerShell-Skripteditor.                 |
| <i>Windows PowerShell</i>              | 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 Administratorrechte zu starten.

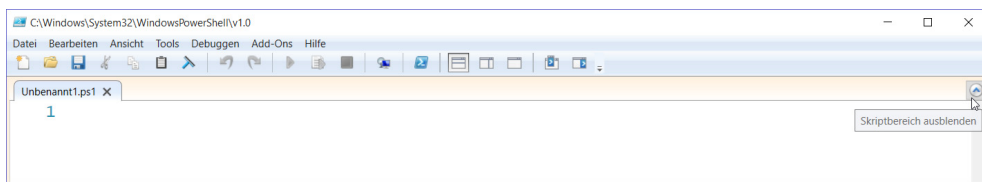


Abbildung 1.5: PowerShell ISE als Konsolenersatz.

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).

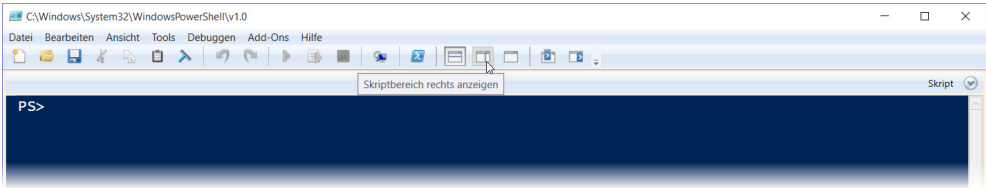


Abbildung 1.6: Über die Werkzeugleiste kann die interaktive PowerShell-Konsole ein- und ausgeblendet werden.

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 Standard-schriftart.

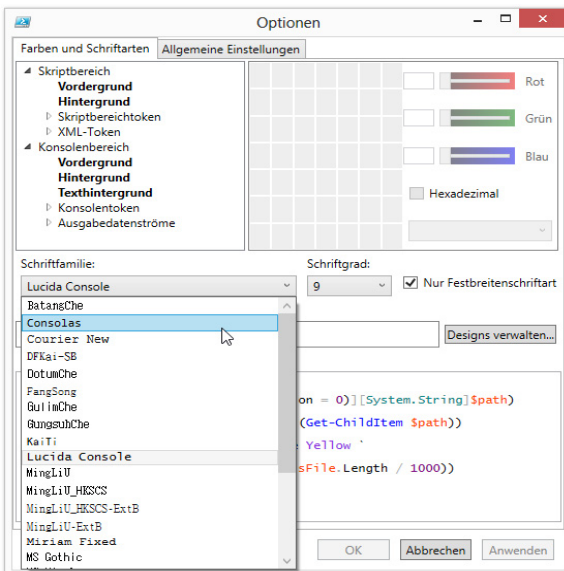


Abbildung 1.7: Andere Schriftart für ISE auswählen.

## Kapitel 1: PowerShell startklar machen

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 
```

Achtung: Dieser Befehl muss in der ISE eingegeben werden, denn die Erweiterung funktioniert nur dort. Geben Sie den Befehl versehentlich in einer PowerShell-Konsole ein, erhalten Sie eine Fehlermeldung.

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 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.

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.

## Kapitel 1: PowerShell startklar machen

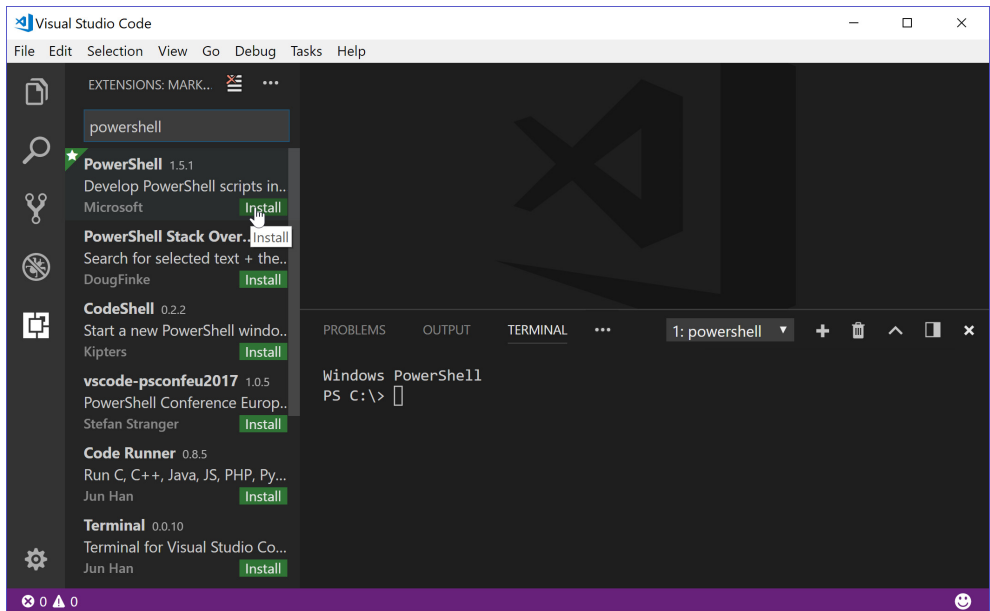


Abbildung 1.8: PowerShell-Erweiterung in VSCode installieren.

## 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 Power-

Shell-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 vergedet 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 lassen 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         |

(...)

## Kapitel 1: PowerShell startklar machen

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
```

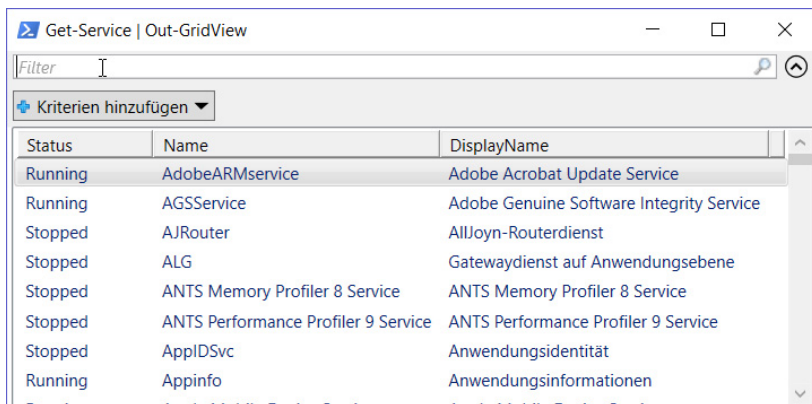


Abbildung 1.9: Befehlsergebnisse in einem Extrafenster anzeigen, dem »GridView«.

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
```