ware an, das Eigengewächs Windows Phone wird hingegen 2015 aufgegeben.

Lässt man diese (R)Evolution Revue passieren, überrascht es wenig, dass die Weiterentwicklung der Windows PowerShell nach zehn Jahren eingestellt wurde. An ihre Stelle tritt 2018 mit Version 6 eine in wesentlichen Teilen neu entwickelte PowerShell (ursprünglich »PowerShell Core« genannt), die neben Windows auch Linux und macOS unterstützt. Diese neue Generation der PowerShell wird nach wie vor federführend von Microsoft-Mitarbeitern entwickelt, ist aber quelloffen auf der bereits erwähnten Plattform GitHub jedem Interessierten zugänglich und steht unter der MIT-Lizenz, einer anerkannten FOSS-Lizenz (*Free and Open Source Software*). Zahlreiche Personen und Organisationen tragen seither zur Weiterentwicklung der Shell bei.

Das lange Leben der Windows PowerShell 5

Die PowerShell basiert wesentlich auf Komponenten des .NET Framework. Diese mächtige Sammlung von Programmbibliotheken und Schnittstellen (zum Beispiel zur Anbindung von Datenbanken, dem Dateisystem oder der Netzwerkkommunikation) existiert in zwei Editionen:

- *.NET Framework für Windows-Betriebssysteme*
  Diese traditionelle .NET-Variante wird gern als »FullCLR« umschrieben, da sie alle Komponenten des Frameworks bereitstellt. Die Abkürzung CLR steht für *Common Language Runtime*, was im Kern die Ausführungsschicht des .NET Framework beschreibt.
- *.NET Framework Core für Windows, Linux und macOS*
  Im Funktionsumfang (noch) eingeschränkte Neuentwicklung, auch »CoreCLR« genannt. Konzipiert als Nachfolger des klassischen .NET Framework.

Mit dem Neustart der Entwicklung hin zu einer plattformübergreifenden Shell war der Wechsel auf das *.NET Framework Core* verbunden. Die Entwickler mussten große Teile des Codes neu schreiben

und dennoch zahlreiche funktionale Einschränkungen in Kauf nehmen, da der Funktionsumfang der CoreCLR stark eingeschränkt war. Diese neue Generation der PowerShell (Version 6+) ist eine installierbare Anwendung. Anders als die etablierte Windows PowerShell 5, wie sie in Windows 10 und in Windows Server ab Version 2016 integriert ist, ist sie kein Bestandteil des Betriebssystems und ist kein Teil der Lizenzvereinbarung zwischen dem Hersteller und dem Kunden.

Microsoft gewährt Geschäftskunden bis zu zehn Jahre (unter gewissen Umständen sogar noch länger) Unterstützung beim Betrieb von Windows Server und Windows 10. Werden Fehler gefunden, stellt der Hersteller Updates für das Betriebssystem bereit. Hierzu gehört nach wie vor die Windows PowerShell 5. Mit jedem neuen *Long Term Servicing*-Release von Windows verlängert sich die garantierte Unterstützung für die alte Generation somit um weitere zehn Jahre – bis zu dem Tag, an dem die Windows PowerShell 5 aus dem Betriebssystem entfernt wird.

Das .NET Framework Core hat einen wesentlich kürzeren Lebenszyklus als die Betriebssysteme aus dem gleichen Haus. Die Core Edition wird bestenfalls für drei Jahre mit Updates versorgt, anschließend ist man gezwungen, auf die nachfolgende Version zu wechseln. Sie ahnen sicherlich, in welchem Dilemma sich aufgrund dieser Umstände Microsofts PowerShell-Team befindet: Die PowerShell ist ab Version 6 nicht mehr Teil des Windows-Betriebssystems, aus Gründen der Kompatibilität verteilt das Windows-Team aber noch auf Jahre hinaus die Windows PowerShell 5, die aller Voraussicht nach jedoch keinerlei Funktionsupdate mehr erfahren wird. Sie verbleibt als Artefakt der Vergangenheit im Fokus der Anwender, da sie mit dem Betriebssystem ausgeliefert wird. Neue Versionen müssen dahin gehend heruntergeladen und bereitgestellt werden, bestehende Skripte müssen auf ihre Kompatibilität hin getestet werden – und das Ganze vor dem Hintergrund, dass PowerShell 6 und 7 in wesentlichen Teilen funktionsreduziert sind im Vergleich zu ihrem Urahn. Neue Versionen der PowerShell müssen also sehr überzeugende Verbesserungen bieten, um den Umstieg auf die aktuelle Generation zu beschleunigen.

Weiterführende Informationen zum .NET Core Lifecycle:

https://docs.microsoft.com/powershell/scripting/powershell-support-lifecycle

https://dotnet.microsoft.com/platform/support/policy/dotnet-core

Dieser Zwiespalt zwischen den Generationen spielt für den Einstieg in die PowerShell jedoch eine untergeordnete Rolle. Dieses Buch stellt die grundlegenden Ideen und Konzepte der PowerShell in den Mittelpunkt. Wo es sinnvoll oder erforderlich ist, erläutere ich versionsabhängige Unterschiede im Detail. Konzentrieren Sie sich auf die für Sie bedeutsamen Anwendungsgebiete und erweitern Sie Ihre Fähigkeiten im Umgang mit der Shell. Ihr Know-how ist versionsunabhängig.

KAPITEL 2

Hallo PowerShell!

Die PowerShell ist ein ursprünglich von Microsoft entwickeltes plattformübergreifendes Werkzeug zur Automation und Konfiguration der IT-Infrastruktur, bestehend aus einer Skriptsprache und einem Kommandozeileninterpreter, der *Shell*. PowerShell ist eine objektorientierte Skriptsprache, die die Klassen und Datentypen des .NET Framework mit einer benutzerfreundlichen Befehlssyntax zugänglich macht.

Die PowerShell steht sowohl im Quelltext als auch kompiliert für zahlreiche Betriebssysteme auf den Projektseiten zum Download bereit: *https://github.com/PowerShell/PowerShell*.

Seit Version 6 steht der Code unter der *MIT-License*. Details zur Installation und den betriebssystemspezifischen Unterschieden finden Sie in Kapitel 3, *Installieren und Aktualisieren der PowerShell*.

```
Get-Uptime
Get-FileHash -Path 'helloworld.ps1'
Test-Connection -TargetName 'www.oreilly.de' -TcpPort 80
```

PowerShell-Befehle sind idealerweise auch für Einsteiger gut les- und erinnerbar. Sogenannte Cmdlets, wie `Get-Uptime` im Beispiel oben, bestehen prinzipiell aus einem Verb (`Get`) und einem Nomen (`Uptime`) und sind exklusiv in der PowerShell ausführbar. Die Shell unterstützt den Anwender darüber hinaus mithilfe der Autovervollständigung beim (Wieder-)Finden der gewünschten Befehle, Parameter und Argumente.

```
Get-Uptime -since
Get-Process -Name 'pwsh'
Compress-Archive -Path '~/*' -DestinationPath 'backup.zip'
```

Einem Cmdlet folgen gegebenenfalls Parameter, gut erkennbar an dem vorangestellten Bindestrich, und Argumente. Im Beispiel oben verwenden wir die Tilde ~, die in der PowerShell (und vielen anderen Shells) das Heimatverzeichnis repräsentiert. In vielen Fällen unterstützt Sie die Shell mithilfe der Tabulatortaste nicht nur beim Vervollständigen des Cmdlet-Namens und dessen Parametern, auch Argumente können in einigen Fällen automatisch vervollständigt werden.

Hinweis

Um Ihnen Werte anbieten zu können, die Sie mithilfe der Autovervollständigung als Argument übergeben, muss der PowerShell-Interpret gültige Werte bereits ermitteln, bevor Sie diese Werte eintippen. Im Beispiel `Get-Process -Name 'pwsh'` kann die PowerShell Ihnen `'pwsh'` anbieten, da Sie im Hintergrund die Liste aktiver Prozesse bereits abgerufen hat. Das Gleiche wird mit `Compress-Archive -Path '~/*' -DestinationPath 'backup.zip'` nicht funktionieren – woher sollte das System wissen, wie Sie Ihre Zip-Datei nennen möchten?

Cmdlets sind das Herzstück der PowerShell. Die Ausführung von Programmcode ist aber nicht auf Cmdlets beschränkt. Sie können ebenfalls die althergebrachten Kommandozeilenprogramme (englisch *Native Commands*) verwenden.

```
nslookup www.oreilly.de
ping www.oreilly.de
```

Ebenso sind einfache (Rechen-)Operationen unmittelbar ausführbar.

```
42 + 23
60 * 60
7 % 2
'Power' + 'Shell'
```

Betrachtet man die PowerShell ein wenig technischer, stößt man schnell auf das bereits eingangs erwähnte .NET Framework.

```
[System.Net.Dns]::GetHostByName('www.oreilly.de')
[System.Net.Sockets.TcpClient]::new('188.40.159.226',80)
```