entwickler.press
shortcuts

PowerShell

Anwendung und effektive Nutzung

Mit PowerShell-4.0-Spickzettel

Dr. Holger Schwichtenberg,
Frank Peter Schultze und
Carsten Eilers

Dr. Holger Schwichtenberg, Frank Peter Schultze und Carsten Eilers

PowerShell

Anwendung und effektive Nutzung

ISBN: 978-3-86802-528-6

Ein Imprint der Software & Support Media GmbH

1 Microsofts Windows PowerShell 4.0 – Einführung in das Kommandozeilenprogramm

Mittlerweile gibt es eine Version 4.0 der Windows PowerShell. Im Alltag findet man aber immer noch viele Administratoren und Softwareentwickler, die die Macht der PowerShell nicht kennen. Dabei ist die PowerShell inzwischen sogar in Visual Studio integriert. Grund genug, die PowerShell noch einmal von Grund auf vorzustellen und die neuesten Features hervorzuheben.

Die PowerShell 4.0 ist in Windows 8.1 oder Windows Server 2012 R2 bereits im Standard installiert. In Windows Server 2012 R2 Core ist sie ein optionales Installationsfeature. Die PowerShell 4.0 wird auf Windows 7 und Windows Server 2008 R2 sowie Windows Server 2012 als Teil des Windows Management Frameworks 4.0 (WMF) [1] installiert. In Windows 8 lässt sich die PowerShell 4.0 nur durch ein Update auf Windows 8.1 nutzen. Ältere Versionen der Power Shell laufen auch auf Windows XP (PowerShell 1.0 und 2.0). Die PowerShell 3.0 läuft ab Windows 7 bzw. Windows Server 2008.

Die PowerShell gibt es auf dem System in zwei Formen: die PowerShell-Konsole (vom Aussehen und Bedienung her dem Windows-Kommandozeilenfenster entsprechend) und das WPF-basierte „Windows PowerShell Integrated Scripting Environment“ (kurz: ISE) mit Skripteditor und interaktivem Eingabebereich.

Commandlets

Ein einzelner Befehl der PowerShell heißt Commandlet (kurz: Cmdlet). Ein Commandlet besteht typischerweise aus drei Teilen: einem Verb und einem Substantiv (getrennt durch einen

Bindestrich) sowie einer (optionalen) Parameterliste mit Name-Wert-Paaren, die durch Leerzeichen getrennt sind. Die Parameter müssen nur dann in Anführungszeichen stehen, wenn im Parameterwert selbst Leerzeichen (z. B. in einem Dateisystempfad) vorkommen. Die Groß- und Kleinschreibung ist bei den Namen nicht relevant.

Ein einfaches Beispiel ohne Parameter lautet: *Get-Process*. Durch einen Parameter kann man z. B. die Prozesse filtern: *Get-Process –name i** liefert Prozesse, deren Namen mit *i* anfangen. Alternativ kann man auch bei einigen Parametern den Namen weglassen, dann muss man aber die Position einhalten. *Get-Process i** führt zum gleichen Ergebnis. Das Filtern über die Prozess-ID funktioniert dann aber nur mit Abgabe des Parameternamens *–id: Get-Process –id 7844*.

Wenn ein Commandlet mehrere Parameter besitzt, ist die Reihenfolge der Parameter entscheidend oder der Nutzer muss die Namen der Parameter mit angeben. Alle folgenden Befehle sind gleichbedeutend, um Dateien mit dem Muster **.doc* aus dem *c:|temp*-Verzeichnis aufzulisten:

```
Get-Childitem C:\temp *.doc
Get-Childitem -Path C:\temp -Filter *.doc
Get-Childitem -Filter *.doc -Path C:\temp
```

Die PowerShell-Konsole unterstützt bei der Eingabe durch Tabulatorvervollständigung. Die ISE bietet eine IntelliSense-Eingabeunterstützung wie Visual Studio.

Durch so genannte Aliase kann der Nutzer die Eingabe von Commandlets verkürzen. So ist *dir* als Alias für *Get-Childitem* oder *help* für *Get-help* vordefiniert. Statt *Get-Childitem c:|* kann er also auch schreiben: *dir c:|*. Ebenso als Alias definiert ist der DOS-Befehl *cd*, der hier Set-Location repräsentiert. *ps* und *gps* sind Aliase für *Get-Process*. Durch *Get-Alias* (oder den entsprechenden Alias *"aliases"*) erhält man eine Liste aller vordefinierten Abkürzungen. Durch Angabe eines Namens bei *Get-Alias* erhält man die Bedeutung eines Alias, z. B. *Get-Alias gps*.

Pipelining

Das Commandlet *Get-Process* lieferte eine Liste der laufenden Prozesse in einer tabellarischen Ausgabe. Diese Ausgabe zeigt aber nicht alle Eigenschaften der Prozesse, sondern nur eine Auswahl, die durch die bei Power Shell mitgelieferte Datei *DotNetTypes.Format.ps1xml* festgelegt ist. Diese Datei kann man theoretisch ändern. Eine fallweise Ausgabe der gewünschten Eigenschaften erreicht man aber besser durch das „pipen“ der Ausgabe an ein Ausgabe-Commandlet:

```
Get-Process | Format-Table name,id,workingset.
```

Die PowerShell arbeitet aber nicht zeichen- oder zeilenorientiert, sondern objektorientiert. Dies bedeutet, dass die Commandlets typisierte .NET-Objekte in die Pipeline werfen und die folgenden Commandlets in der Pipeline diese Objekte anhand ihrer Attribute auswerten können, z. B. Filtern durch *Where-Object*. *$_* steht dabei für das jeweils aktuelle Objekt in der Pipeline. *–eq* ist der Vergleichsoperator. Deutlicher wird der objektorientierte Ansatz, wenn man als Attribut keine Zeichenkette heranzieht, sondern eine Zahl. *WorkingSet* ist ein Zahlenwert, der den aktuellen Speicherverbrauch eines Prozesses repräsentiert. Alle Prozesse, die aktuell mehr als 20 Megabyte verbrauchen, liefert der folgende Befehl:

```
Get-Process | Where-Object {$_.WorkingSet -gt 20*1024*1024 } | Sort-Object
WorkingSet -desc | Format-Table name,id,workingset
```

Anstelle von *20*1024*1024* hätte man auch das Kürzel *20MB* einsetzen können. *Workingset* kann man durch *ws* abkürzen. Außerdem kann man *Where-Object* mit einem Fragezeichen und *Sort-Object* mit *sort* und *Format-Table* mit *ft* abkürzen. Zudem kann man seit PowerShell 3.0 bei Bedingungen ohne logische Verknüpfung auch die geschweiften Klammern und das *$_* weglassen. Die ganz prägnante Variante des Befehls wäre dann also (Das Ergebnis zeigt **Abbildung 1.1**):

```
ps | ? ws -gt 20MB | sort ws -desc | ft name,id,ws
```