Thomas Steinberger

Administrator Praxis
Microsoft Deployment Toolkit (MDT) 2013

Verlag Nicole Laue

Anhang A:
Grundlagen PowerShell

Version 2.0
August 2018

4

1 Grundlagen PowerShell

„Learn PowerShell or learn to say „would you like fries with that?" - Don Jones[1]

Mittlerweile ist jedes neue Produkt von Microsoft fest mit der PowerShell assoziiert. Manche Produkte, wie Exchange, lassen sich an manchen Stellen ausschließlich über PowerShell konfigurieren. Selbst die dafür vorhandenen graphischen Oberflächen führen letztlich im Hintergrund PowerShell Commandlets (kurz Cmdlets) aus.

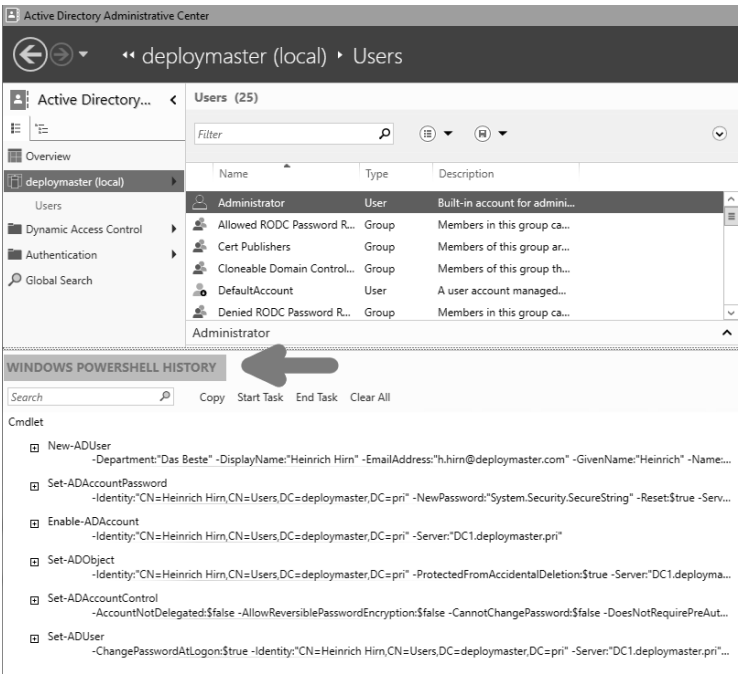


Abbildung 1.1: Anlegen eines Benutzers im ADAC durch Cmdlets

[1] Don Jones ist langjähriger „Most valuable professional (MVP) PowerShell" und beliebter Redner auf Microsoft-Konferenzen.

Über Cmdlets lassen sich alle Einstellungen auch über die Kommandozeile konfigurieren. Somit ist die Möglichkeit gegeben alle Konfigurationen zu automatisieren. Das gilt auch für die Active Directory Verwaltungskonsole (engl. Active Directory Administration Center = ADAC). Sie können sich darin die PowerShell History die letzten durchgeführten Aktionen anzeigen als PowerShell-Code lassen.

Aus diesem Grund sollte ein gewisses Basiswissen über die PowerShell Pflicht sein.

Dieser Anhang gibt Ihnen einen kleinen Überblick und fundamentales Knowhow über die Möglichkeiten der PowerShell.

Wenn Sie tiefer in das Thema einsteigen wollen, dann stehen Ihnen die Ressourcen der Microsoft Virtual Academy unentgeltlich mit entsprechenden Video-Kursen in englischer Sprache zur Verfügung.[2]

1.1 Bevor es los geht

Der Anhang A ist etwas zum mit machen. Probieren Sie während des Lesens gleich die beschriebenen Commandlets Schritt für Schritt aus.

1.2 Voraussetzungen

Zum gegenwärtigen Zeitpunkt ist das Windows Management Framework 5.1[3] die offizielle und aktuelle Version. Darin ist PowerShell 4 enthalten. Um das Windows Management Framework 5.1 installieren zu können benötigen Sie mindestens das .NET Framework 4.5. Um die Active Directory Commandlets (alles rund um das Thema Benutzer, Gruppen, OUs etc.) nutzen zu können benötigen Sie die Remote Server Administration Tools (RSAT) für Ihr gegenwärtiges Windows Betriebssystem und natürlich ein funktionierendes Active Directory.

Zusätzlich sollten Sie über etwas Kommandozeilen-Erfahrung verfügen.

1.3 Grundsätzliches

Mit dem Windows Management Framework 5.1 erhalten Sie neben der PowerShell Konsole auch die Integrated Scripting Environment Konsole (Kurz: ISE, englisch für „Integrierte Skript Umgebung“).

[2] Ein sehr gutes Buch, dass leider auch nur auf Englisch verfügbar ist, wurde von Don Jones geschrieben: Learn Windows PowerShell 3 in a Month of Lunches, ISBN-13: 978-1617291081.

[3] https://www.microsoft.com/en-us/download/details.aspx?id=54616

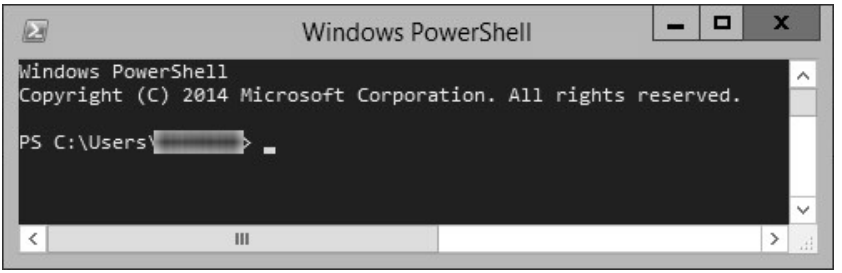


Abbildung 1.2: PowerShell Konsole

Die ISE ist eine einfache Entwicklungsumgebung für PowerShell-Skripte. Wie auch in Word werden fehlerhafte Elemente mit einer roten gekringelten Linie gekennzeichnet. Als zusätzliche Hilfe blendet Ihnen die ISE beim Schreiben der Commandlets die weiteren möglichen Parameter ein. Diese Funktion heißt IntelliSense.[4]

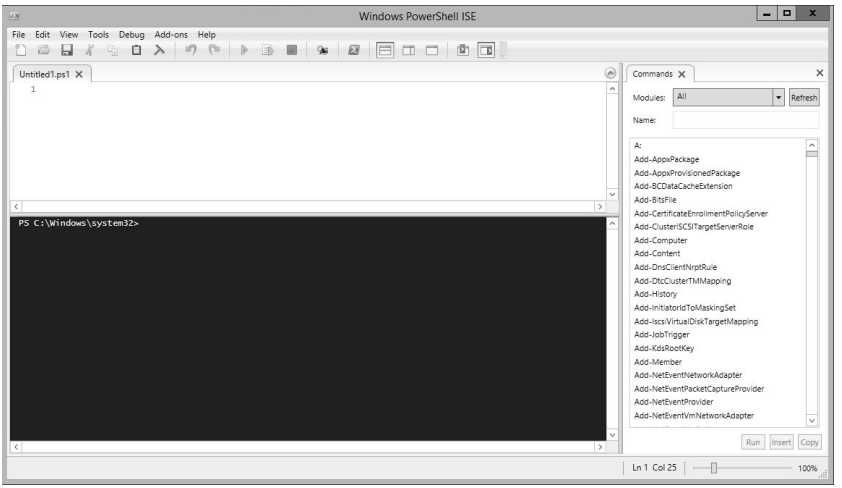


Abbildung 1.3: PowerShell ISE mit Skript- und Befehlsleiste

[4] Sie können IntelliSense jederzeit mit STRG+ Leertaste erneut starten.

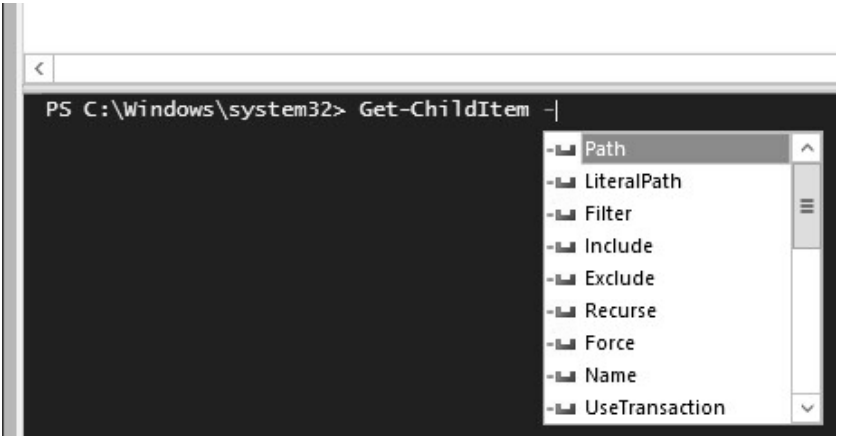


Abbildung 1.4: ISE mit Parameterauswahl durch IntelliSense

Eine weitere ISE-Funktionalität ist die farbliche Codierung der Cmdlets. Als Standard sind die Commandlets hellblau, Parameter dunkelblau und Parameterwerte lila. Variablen sind orange und Kommentare sind grün. An dieser Farbgebung können Sie sich orientieren und nach etwas Übung sehen Sie, was die ISE als nächstes von Ihnen als Eingabe erwartet. Eventuell möchte PowerShell an dieser Stelle einen Parameter oder einen Wert? Sie lesen dadurch auch fremde Skripte leichter. Oder später Ihre eigenen Skripte ;-).

Praxis-Tipp: Gewöhnen Sie sich früh die Verwendung der Tab-Taste an. Damit können Sie die bereits getippten Commandlets automatisch vervollständigen. Dadurch haben Sie zum einen weniger Tipparbeit. Zum anderen machen Sie auch deutlich weniger Tippfehler. Aus „Invo“ und dreimal Tab-Taste wird „Invoke-Command“.

Praxis-Tipp: Sollten Sie ein kleines Skript mit der ISE schreiben, dann nutzen Sie unbedingt einmal die Tasten-Kombination STRG+J. Damit können Sie sich kleine PowerShell-Code-Fragmente (engl. Snippets) einblenden lassen und Sie haben weniger Tipparbeit bei höherem Komfort. Die Snippets reichen von einem Funktionen-Grundgerüst über einen Kommentarblock zu den einzelnen Schleifenart und Abfragen.

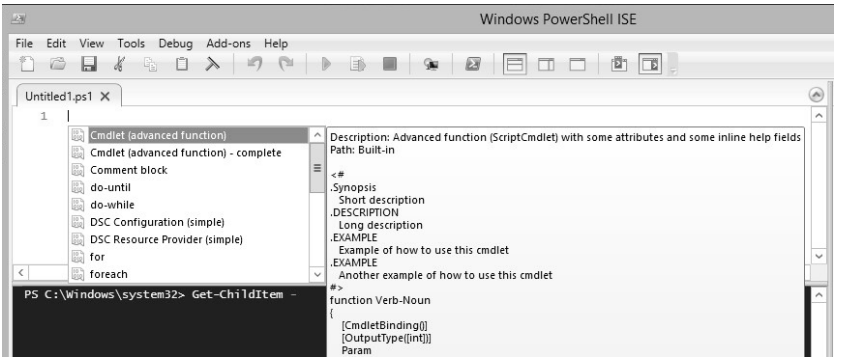


Abbildung 1.5: ISE mit Snippet-Auswahl

Eine der Neuerungen in PowerShell 5 ist jetzt, dass die PowerShell-Konsole auch in der Syntax farblich hinterlegt ist.

Praxis-Tipp:
Nutzen Sie zu Beginn Ihres Lernprozesses ausschließlich die ISE!

1.3.1 Welche Version haben Sie?

Falls Sie sich nicht sicher sind, welche PowerShell Version Sie aktuell auf Ihrem Rechner/ Server haben, dann lässt sich das leicht überprüfen. Starten Sie dazu eine PowerShell-Konsole. Ab Windows 7 drücken Sie einfach den Start-Knopf und tippen „power“ ein. Warten Sie einen Moment, bis die Suche Ihnen auch „Windows PowerShell“ als Ergebnis zurückliefert, wählen Sie dieses und drücken Sie dann die Enter-Taste.[5]

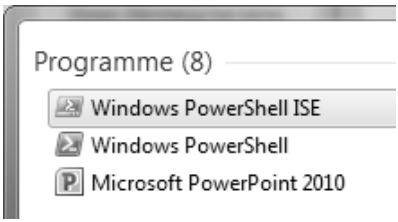


Abbildung 1.6: Suchergebnis unter Windows 7

Ansonsten finden Sie das Gewünschte über „Start – Alle Programme – Zubehör – Windows PowerShell“.

[5] Unter Windows 8.x können Sie im Start-Menü gleich mit dem Tippen beginnen.

Abbildung 1.7: PowerShell über das Startmenü

Nachdem sich die PowerShell Konsole geöffnet hat tippen Sie „`$psversiontable`“ ein und drücken die Enter-Taste. (Schneller geht es mit „$psv + Tab-Taste“.)

Als Ergebnis erhalten Sie eine kleine Tabelle, die Ihnen beim Eintrag „PSVersion“ verrät, welche PowerShell Version bei Ihnen installiert ist.

```
Name                           Value
----                           -----
PSVersion                      4.0
WSManStackVersion              3.0
SerializationVersion           1.1.0.1
CLRVersion                     4.0.30319.34209
BuildVersion                   6.3.9600.16406
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0}
PSRemotingProtocolVersion      2.2
```

Auf diesem System wäre also PSVersion 4.0 installiert. Es kann losgehen!

```
PS C:\Temp> $PSVersionTable

Name                           Value
----                           -----
PSVersion                      4.0
WSManStackVersion              3.0
SerializationVersion           1.1.0.1
CLRVersion                     4.0.30319.34209
BuildVersion                   6.3.9600.16406
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0}
PSRemotingProtocolVersion      2.2
```

Abbildung 1.8: Abfrage der PowerShell Version

Sollten Sie schon mit Windows 10 arbeiten, dann haben Sie bereits PSVersion 5.x.

1.3.2 Benutzerdefinierte Anpassung

Sowohl die Konsole als auch die ISE lassen sich auf die individuellen Vorlieben farblich und schriftarttechnisch anpassen. Bei der Konsole erreichen Sie das auf dem gleichen Weg wie beim normalen Kommandozeileninterpreter cmd.exe. Klicken Sie einfach mit der rechten Maustaste auf die Titelleiste und rufen Sie die Eigenschaften auf.

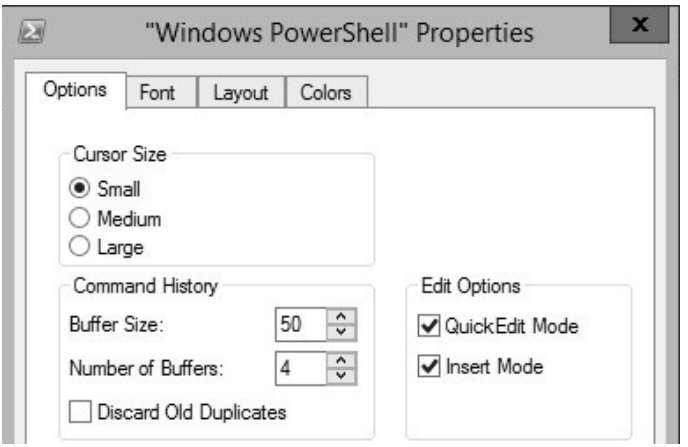


Abbildung 1.9: Konsoleneigenschaften

Danach können Sie die Fenstergröße, Schriftart und –größe sowie die farblichen Einstellungen für Hinter und Vordergrund ändern.

In der ISE haben Sie einen eigenen Menüpunkt unter Tools - Optionen. Hier können Sie sogar die einzelnen Syntaxfarben Ihren individuellen Farbwünschen anpassen.

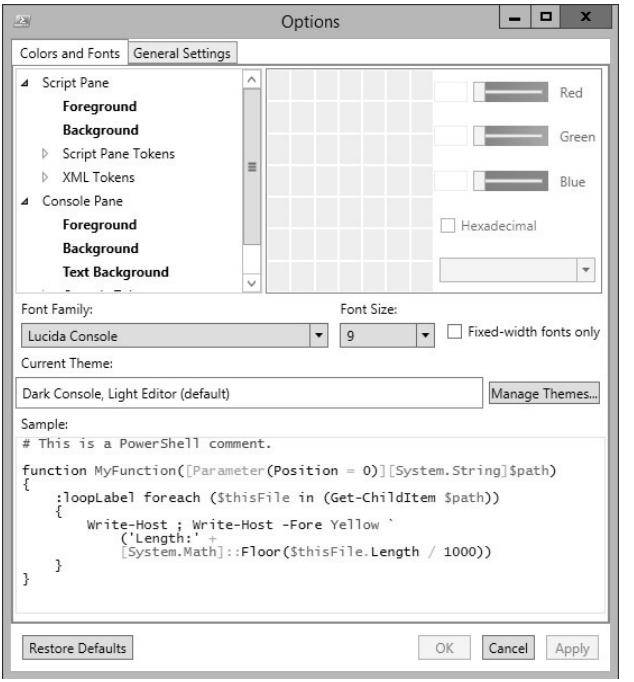


Abbildung 1.10: ISE-Eigenschaften

1.4 Commandlets direkt ausführen

Cmdlets führen Sie grundsätzlich so aus, wie Sie das von der cmd.exe gewohnt sind. Tippen Sie den Befehl ein und führen ihn mit der „Enter"-Taste aus. Idealerweise nutzen Sie zum Tippen die Tab-Taste zur automatischen Vervollständigung.

„Get-C + Tab-Taste" vervollständigt zu „`Get-ChildItem`". Jeder weiterer Tastendruck der Tab-Taste führt Sie weiter in der Liste der PowerShell-Befehle, die mit „Get-C" beginnen. Das nächste Cmdlet wäre demnach „`Get-CimAssociatedInstance`". Einen Schritt zurück zu „`Get-ChildItem`" kommen Sie mit „Shift + Tab-Taste". Zur Ausführung drücken Sie einfach die „Enter"-Taste.

Je nach ausgeführtem Commandlet müssen Sie noch weitere Parameter angeben. „`Get-Childitem`" funktioniert hier ohne weiteren Parameter. Die Details dazu sehen wir uns im dazugehörigen Unterkapitel 1.9 an. Aber so viel vorab: als notwendiger Pflichtparameter „`-Path`" ist ein Standardparameter hinterlegt, falls nichts angegeben ist. Dieser Standardparameter ist der gegenwärtige Pfad. Sollten Sie gerade in „c:\temp" sein, dann zeigt Ihnen „`Get-Childitem`" den Inhalt des Pfades c:\temp ohne Unterordner.

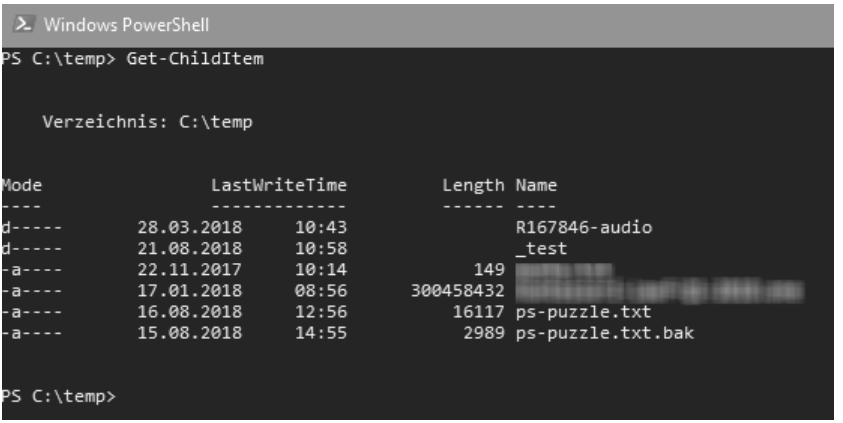


Abbildung 1.11: Inhalt von C:\Temp

1.5 Commandlets als Skript ausführen

Standardmäßig ist die Ausführungsrichtlinie von PowerShell so eingestellt, dass Sie keine PowerShell-Skripte aus Versehen ausführen können. Wenn Sie eine Datei mit der Endung „.ps1“ doppelklicken öffnet sich Notepad mit dem im Skript enthaltenen Code. Der Code wird **nicht** ausgeführt. Sie müssen aktiv die PowerShell-Konsole oder ISE starten und von dort das Skript aufrufen. Diese Vorgehensweise gehört zu den Sicherheitsmechanismen von PowerShell.

Prüfen Sie als nächstes, wie die Richtlinie zur Ausführung von Skripten auf Ihrem Rechner eingestellt ist. Standardmäßig werden Skripte nicht ausgeführt. Auch wenn Sie diese aktiv über die Konsole oder ISE ausführen wollen.

Starten Sie die Konsole als Administrator und tippen Sie „Get-Ex + Tab-Taste“. Die Vervollständigung zeigt Ihnen nun „`Get-ExecutionPolicy`“ an. Bestätigen Sie mit der „Enter“-Taste. Wenn Sie als Ergebnis „Restricted“ zurück erhalten, dann müssen Sie die Ausführungsrichtlinie ändern.

Mit „`Set-ExecutionPolicy RemoteSigned -Force`“ erlauben Sie PowerShell-Skripte für selbst erstellte Skripte und digital signierte fremde Skripte, z.B. aus dem Internet. Der „-Force“-Parameter unterdrückt an dieser Stelle die Nachfrage, ob Sie die Ausführungsrichtlinie wirklich ändern wollen. Ab diesem Moment können Sie PowerShell-Dateien über die Konsole oder ISE ausführen.[6]

[6] Sehen Sie aus Sicherheitsgründen davon ab bei der ExecutionPolicy die Option „Unrestricted“ zu wählen. Das ist ein Sicherheitsrisiko.

1.6 Struktur eines Commandlets

Schauen wir uns nun die Basis-Struktur eines Commandlets an.

Ein Cmdlet besteht im einfachsten Fall aus einem Verb, einem Bindestrich und einem Hauptwort.[7]

```
Get-ChildItem
```

Das Verb „Get“ gefolgt von einem Bindestrich **ohne** Leerzeichen und gefolgt vom Hauptwort „ChildItem“. Auch hier **ohne** Leerzeichen dazwischen. Das Hauptwort ist immer in der Einzahl. Es gibt z.B. „ChildItem“ und „Service“ usw. Auch wenn Sie mit „`Get-ChildItem`“ alle Datei**en** in diesem Verzeichnis angezeigt bekommen, heißt das Cmdlet nicht „Get-ChildItem**s**“. Ebenso ist für den Anfänger zunächst verwirrend, dass im Sprachgebrauch der Bindestrich schlicht weggelassen wird. „`Get-ChildItem`“ wird „Get ChildItem“ ausgesprochen. Fragen Sie also Ihren erfahrenen Kollegen nach dem Commandlet für eine neue Active Directory Gruppe, wird er Ihnen hoffentlich „New ADGroup“ antworten. Schreiben Sie jetzt auf der Konsole den Befehl genau so, dann erhalten Sie in freundlichen roten Lettern eine Fehlermeldung.

```
„new: Die Benennung "new" 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 enthalten), und wiederholen Sie den Vorgang.
In Zeile:1 Zeichen:1
+ new adgroup
+ ~~~
+ CategoryInfo : ObjectNotFound: (new:String) [],
CommandNotFoundException
+ FullyQualifiedErrorId : CommandNotFoundException”
```

[7] Es hat sich so ergeben, dass man von der „Verb-Hauptwort“-Struktur redet, auch wenn „New“ kein Verb ist und „SqlAvailabilityGroupListenerStaticIp“ ein sperriges Hauptwort (= Nomen) ist.

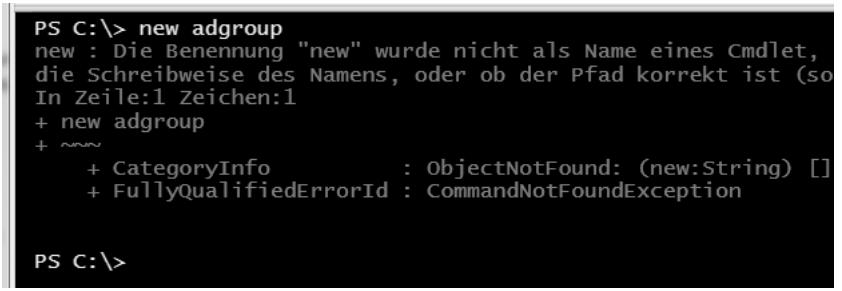


Abbildung 1.12: Fehlermeldung von „New ADGroup“ (Ausschnitt)

Aber an Hand der farblichen Codierung der Syntax in der ISE an Ihrem System sehen Sie eigentlich schon, dass “new” blau ist, aber „ADGroup” die violette Wertefarbe besitzt. Es muss etwas „faul sein im Staate Dänemark“.

Danach folgen in einer korrekten Syntax die Parameter, die stets einen Bindestrich vorangestellt haben und mit einem Leerzeichen die einzelnen Werte zwischen Parameter und Wert. Die einzelnen Werte sind durch Kommata zu trennen.

Beispiele:

```
Get-ChildItem –Path c:\temp
```

```
Get-ChildItem –Path c:\temp –force –recurse
```

```
Get-ChildItem –Path c:\temp, c:\Windows –force -recurse
```

Beachten Sie, dass Sie viele Cmdlets ohne Parameter verwenden können. Manche allerdings brauchen zwingend einen oder mehrere Parameter. Wenn Sie diese nicht angeben, dann werden Sie dazu aufgefordert diese nachträglich anzugeben. Sehen wir uns gleich ein Beispiel an. Tippen Sie „`Get-WmiObject`“ und schließen Sie mit „Enter“ ab.

```
PS C:\Temp> Get-WmiObject
Cmdlet Get-WmiObject an der Befehlspipelineposition 1
Geben Sie Werte für die folgenden Parameter an:
Class:
```

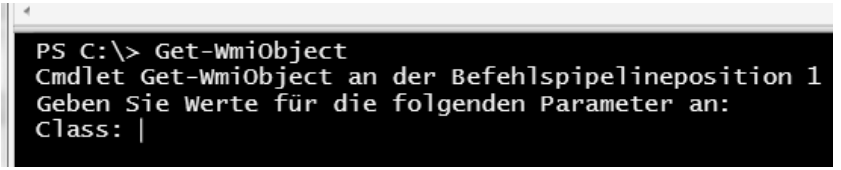


Abbildung 1.13: Get-WmiObject ohne Parameter fordert die Klasse an

Hier ist also verpflichtend ein Parameter notwendig. Geben Sie hier einmal als Klasse „win32_bios“ an. Als Ergebnis sollten Sie diverse Angaben zur Bios Version, zum Hersteller (Manufacturer), zur Seriennummer etc. erhalten.

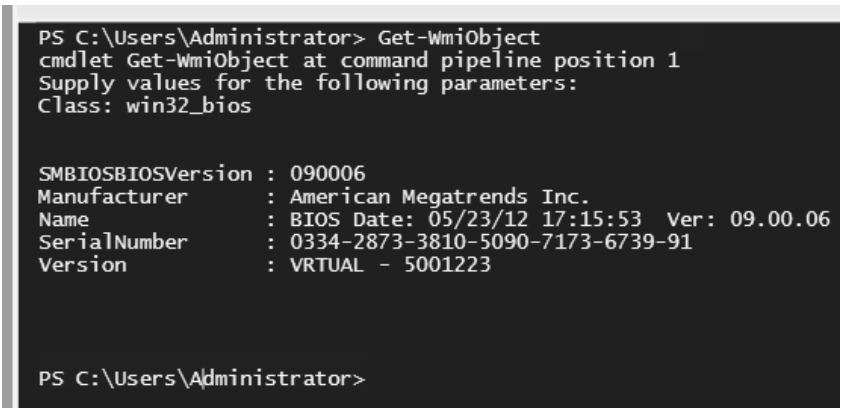


Abbildung 1.14: Get-WmiObject mit class-Parameter „win32_bios“

1.7 Hilfe benutzen

Bisher war man mit den Hilfetexten innerhalb der Kommandozeile unter Windows nicht sehr verwöhnt.

```
c:\>net user /?
Die Syntax dieses Befehls lautet:
NET USER
[Benutzername [Kennwort | *] [Optionen]] [/DOMAIN]
      Benutzername {Kennwort | *} /ADD [Optionen]
[/DOMAIN]
      Benutzername [/DELETE] [/DOMAIN]
      Benutzername [/TIMES:{Zeiten | ALL}]
```

Das hat sich unter PowerShell fast schon „dramatisch“ geändert. Die Hilfeseiten sind sehr ausführlich und erinnern an die „Man-Pages“ unter Linux/ Unix.

1.7.1 Aktualisierung der Hilfedateien

Die Hilfedateien sind seit PowerShell 4 nicht mehr standardmäßig vollständig mit dabei. Der Grund dafür ist schlicht, dass man dadurch keine fehlerhaften und veralteten Hilfedateien ausliefert sowie verhindern will, dass große Windows Update Pakete nur mit der PowerShell-Hilfe für Benutzer bereitgestellt werden, welche die PowerShell-Hilfe niemals benötigen werden. Um stets die aktuelle Version der Hilfedateien zu erhalten gibt es seit PSVersion 3 ein extra Commandlet dafür: „`Update-Help`“. Um die Hilfedateien zu aktualisieren benötigen Sie einen Internetzugang und ein paar Minuten Zeit.
Starten Sie eine PowerShell-Konsole als Administrator und geben Sie dort den Befehl „`Update-Help`“ ein. Die Hilfedateien laden sich dann bei einer bestehenden Internetverbindung innerhalb weniger Minuten herunter.

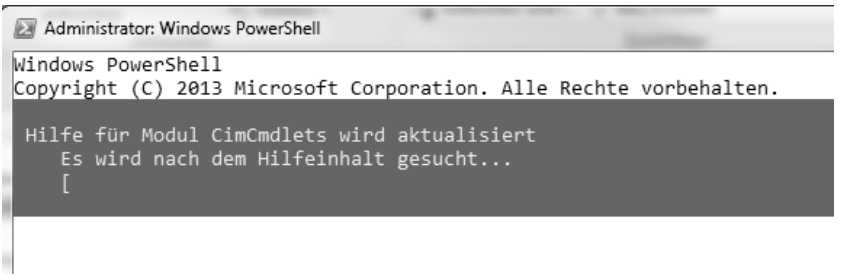


Abbildung 1.15: Aktualisierung der Hilfe als Administrator

Zeitgleich mit der Aktualisierung der Hilfedateien zum Herunterladen über den eben beschriebenen Weg, aktualisiert Microsoft zeitgleich auch die Online-Version. Wenn Sie keine Administratoren-Berechtigungen auf dem System haben, aber einen Internet-Zugang besitzen, dann nutzen Sie die Option „`-Online`“ hinter dem Hilfe-Cmdlet: „`Get-Help <Name des Commandlets> –Online`“. Sie bekommen damit die Hilfe direkt in Ihrem Standard-Browser angezeigt.

1.7.2 Export der Hilfe-Dateien

Sie können die einmal heruntergeladenen Hilfedateien exportieren und auf anderen Rechnern importieren. Zum Beispiel auf Servern, die aus Sicherheitsgründen nicht mit dem Internet verbunden sind. Das Cmdlet dazu ist „`Save-Help`“. Zum Export müssen Sie einen Speicherort und optional die Sprachversion dazu angeben. Die Parameter dazu sind „`-DestinationPath`“ und „`-UICulture`“. Der komplette Befehl lautet somit „`Save-Help -DestinationPath C:\Temp\ps_help –UICulture „en-us“`“. Die Hilfe wird nun im Pfad C:\Temp\ps_help

gespeichert.

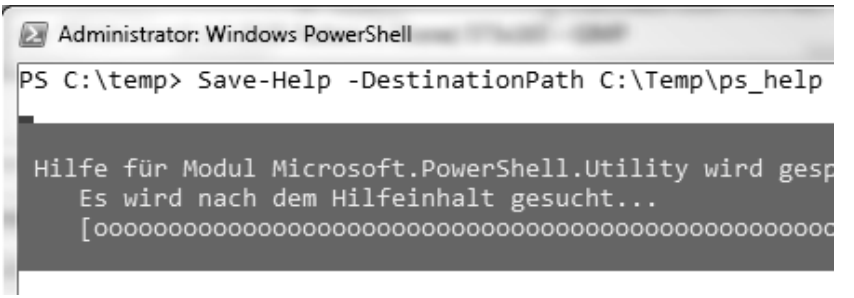


Abbildung 1.16: Export der Hilfedateien nach c:\temp\ps_help

Dieser Vorgang dauert einen Moment und Sie finden die exportieren Hilfedateien anschließend im angegebenen Ordner.

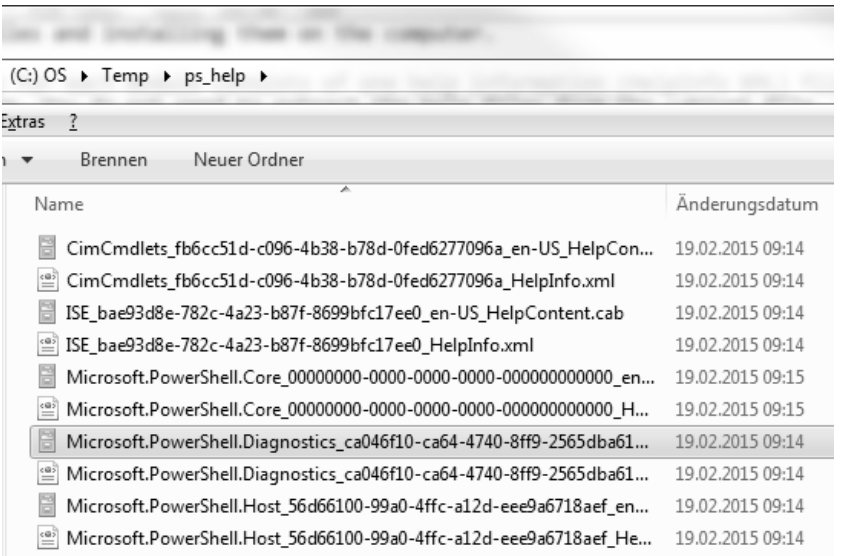


Abbildung 1.17: Exportierte PowerShell-Hilfedateien

1.7.3 Import der Hilfe-Dateien

Kopieren Sie nun das komplette Verzeichnis auf einen USB-Stick oder eine Netzwerk-Freigabe und importieren Sie mittels „`Update-Help`“ die Hilfe-Dateien auf den oder die anderen Rechner. Sie müssen dabei den Pfad zu den Hilfedateien angeben. Der notwendige Parameter ist „`-Path`“: „`Update-Help –Path <Pfad zu den Hilfedateien>`“. Auch hier dauert es einen kurzen Moment, bis die Hilfedateien alle in das lokale System importiert sind.

1.7.4 Verwendung der Hilfe

Jetzt liegen Ihnen die aktuellen Hilfe-Dateien vor. Damit können Sie zu jedem Commandlet die komplette Syntax einsehen und sich auch Beispiele dazu ansehen. Spielen wir ein Beispiel durch.

Der Befehl um sich die Hilfe-Inhalte zu einem Cmdlet anzusehen lautet „`Get-Help <Cmdlet>`“. Damit Sie sich den Inhalt eines Ordners anzeigen zu lassen können gibt es das Cmdlet „`Get-ChildItem`“. Wenn Sie das nun in der PowerShell-Konsole eintippen und mit „Enter“ bestätigen, dann erhalten Sie die Ausgabe zu diesem Ordner. Jetzt möchten Sie aber auch die untergeordneten Ordner aufgelistet haben. Deshalb müssen Sie wissen, welche Parameter es zu „`Get-ChildItem`“ gibt. Verwenden Sie nun den Befehl „`Get-Help Get-ChildItem`“.

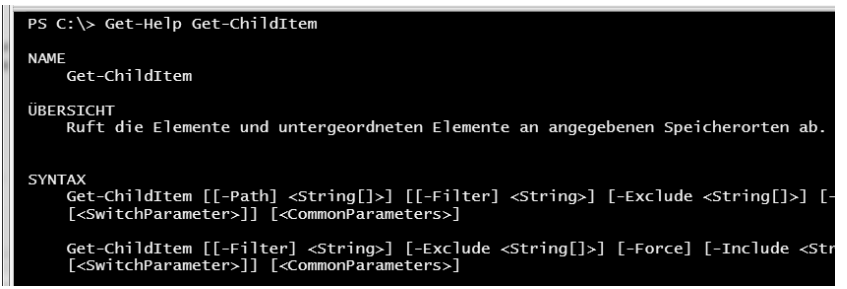


Abbildung 1.18: Get-Help Get-ChildItem (Ausschnitt)

Die Ausgabe am Bildschirm zeigt Ihnen nun das Ergebnis des Hilfetexts zu „`Get-ChildItem`“. Sie sehen noch einmal den Namen des Cmdlets und in einer kurzen Beschreibung, was das Cmdlet macht, wenn Sie es verwenden.
„`Ruft die Elemente und untergeordneten Elemente an angegebenen Speicherorten ab.`“ Damit haben Sie das PowerShell-Gegenstück zu „dir“ unter der bekannten schwarzen Kommandozeile. „`Get-ChildItem`“ zeigt Ihnen den Inhalt des gegenwärtigen Pfads. Als nächstes sehen Sie zwei Parametersätze[8]. Diese sehen auf den ersten Blick identisch aus.

Der erste Parametersatz beginnt mit `Get-ChildItem [[-Path]]` und der Zweite mit `Get-ChildItem [[-Filter]]`. Die weiteren Parameter unterscheiden sich in beiden Sätzen von Get-ChildItem kaum.

[8] „Windows PowerShell uses parameter sets to enable you to write a single cmdlet that can perform different actions for different scenarios. Parameter sets enable you to expose different parameters to the user and to return different information based on the parameters specified by the user.” Quelle: https://msdn.microsoft.com/en-us/library/dd878348(v=vs.85).aspx

Grundsätzlich gilt: wenn Sie einmal einen Parameter verwenden, der für diesen Parametersatz einzigartig ist, können Sie ausschließlich nur die für diesen Parametersatz hinterlegten Parameter bis zum Ende dieses Cmdlets verwenden. **Sie können nicht beide Parametersätze in einem PowerShell-Befehl mischen.**

Als konkretes Beispiel: Wenn Sie den Parametersatz `Get-ChildItem [[-Path]]` verwenden, dann steht Ihnen der Parameter „`-LiteralPath`“ hier nicht zur Verfügung.[9]

Sehen wir uns einen Parametersatz etwas genauer an:

„`Get-ChildItem [[-Path] <String[]>] [[-Filter] <String>] [-Exclude <String[]>] [-Force] [-Include <String[]>] [-Name] [-Recurse] [-UseTransaction [<SwitchParameter>]] [<CommonParameters>]`“

Sie sehen um die einzelnen Parameter eine oder mehrere eckige Klammern “[..]”. Diese bedeuten, dass diese Angabe optional ist. In diesem Fall müssen Sie den Parameter „`–Path`“ nicht angeben. PowerShell interpretiert den ersten Wert nach „`Get-ChildItem`“ als den Wert zu „`-Path`“. Geben Sie auch hier nichts an, dann wird automatisch der aktuelle Pfad verwendet, in dem sich die PowerShell-Konsole gerade befindet. Daher funktioniert „`Get-ChildItem`“ auch ohne explizite Parameter und Werte. Genau wie beispielsweise auch „`Get-Service`“.

Hinter dem Parameter „`-Path`“ steht als Eingabewerte „<`String[]`>“. Das bedeutet, dass mehrfache Werte als Eingabe akzeptiert werden, die jeweils durch ein Komma voneinander getrennt werden müssen.

„`Get-ChildItem –Path C:\temp, C:\Windows, c:\Windows\system32`“

Ein alternativer Weg sich die Hilfedatei außerhalb der Konsole und/ oder ISE anzeigen zu lassen ist neben der bereits erwähnten Methode über den Parameter „`-Online`“ der Parameter „`-ShowWindow`“. Es öffnet sich dadurch ein extra Fenster mit dem Inhalt der lokalen Hilfe. Dies können Sie bequem auf einen zweiten Monitor oder neben das Konsolenfenster schieben. Sie bekommen hier immer die kompletten Inhalte angezeigt. So als würden Sie die Option „`-Full`“ nutzen. (dazu kommen wir in ein paar Momenten in diesem Anhang.)

[9] Im Gegensatz dazu hat „Invoke-Command“ sieben Syntax-Sätze, die sich mitunter signifikant unterscheiden.

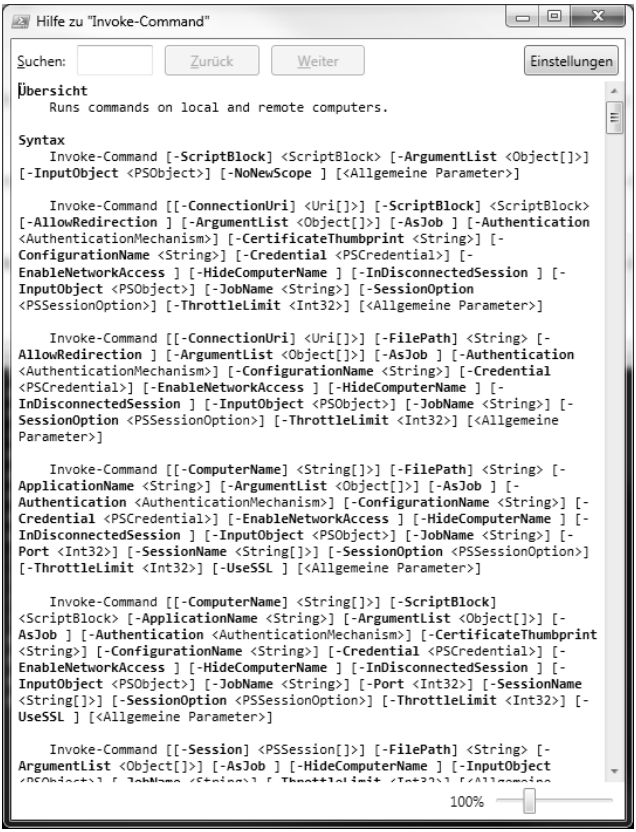


Abbildung 1.19: Get-Help Invoke-Command -ShowWindows

Praxis-Tipp: Gewöhnen Sie sich an die gewünschte Hilfe zu einem Cmdlet stets mit dem Parameter „`-Full`“ aufzurufen.

Der „`Get-Help`“-Aufruf eines Cmdlets mit dem Parameter „`–Full`“ hat mehrere Vorteile. Dadurch erhalten Sie die detaillierte Hilfeinhalte mit einer ausführlichen Beschreibung der Parameter und zwischen einem und elf (!) Beispielen. Die Ausführungen zeigen Ihnen im Einzelnen an, welche Parameter verpflichtend und positional sind. Positionale Parameter müssen an der geforderten Stelle stehen. Manchmal mit dem Parameternamen, manchmal ohne.

Schauen wir uns dazu gleich „Get-Help `Get-ChildItem –Full`“ an.

```
PS C:\Temp> Get-Help Get-ChildItem -Full

NAME
      Get-ChildItem
ÜBERSICHT
      Ruft die Elemente und untergeordneten Elemente
an angegebenen Speicherorten ab.
SYNTAX
      Get-ChildItem [[-Path] <String[]>] [[-Filter]
<String>] [-Exclude <String[]>] [-Force] [-Include
<String[]>] [-Name] [-Recurse] [-UseTransaction
      [<SwitchParameter>]] [<CommonParameters>]
      Get-ChildItem [[-Filter] <String>] [-Exclude
<String[]>] [-Force] [-Include <String[]>] [-Name] [-
Recurse] -LiteralPath <String[]> [-UseTransaction
      [<SwitchParameter>]] [<CommonParameters>]
BESCHREIBUNG
      Mit dem Cmdlet Get-ChildItem werden die
Elemente an angegebenen Speicherorten abgerufen. Wenn
es sich bei dem Element um einen Container handelt,
werden die darin enthaltenen (untergeordneten)
Elemente abgerufen. Mit dem Recurse-Parameter können
Sie Elemente in allen untergeordneten Containern
abrufen.
      Bei einem Speicherort kann es sich um einen
Dateisystemspeicherort, z.B. ein Verzeichnis, oder
einen von einem anderen WindowsPowerShell-Anbieter
verfügbar gemachten Speicherort handeln, z.B. eine
Registrierungsstruktur oder einen Zertifikatspeicher.
PARAMETER
      -Exclude <String[]>
            Lässt die angegebenen Elemente aus. Der
Path-Parameter wird durch den Wert dieses Parameters
qualifiziert. Geben Sie ein Pfadelement oder ein
Muster wie „*.txt“ ein. Platzhalter sind zulässig.
            Erforderlich?
      false
            Position?
```

named

Standardwert

Pipelineeingaben akzeptieren?

false

Platzhalterzeichen akzeptieren? true

-Filter <String>

Gibt einen Filter im Format oder in der Sprache des Anbieters an. Der Path-Parameter wird durch den Wert dieses Parameters qualifiziert. Die Syntax des Filters einschließlich der Verwendung von Platzhaltern ist vom Anbieter abhängig. Filter sind effizienter als andere Parameter, da sie beim Abrufen der Objekte vom Anbieter angewendet werden und die Objekte nicht erst nach dem Abrufen von WindowsPowerShell gefiltert werden.

Erforderlich?

false

Position? 2

Standardwert

Pipelineeingaben akzeptieren?

false

Platzhalterzeichen akzeptieren? true

-Force [<SwitchParameter>]

Ermöglicht dem Cmdlet das Abrufen von Elementen, auf die der Benutzer anderweitig nicht zugreifen kann, z.B. ausgeblendete Dateien oder Systemdateien. Die Implementierung unterscheidet sich bei den einzelnen Anbietern. Weitere Informationen finden Sie unter about_Providers (http://go.microsoft.com/fwlink/?LinkID=113250). Auch bei Verwendung des Force-Parameters kann das Cmdlet keine Sicherheitseinschränkungen außer Kraft setzen.

Erforderlich?

false

Position?

named

Standardwert

False

Pipelineeingaben akzeptieren?
false
Platzhalterzeichen akzeptieren?
false
-Include <String[]>
Ruft nur die angegebenen Elemente ab. Der Path-Parameter wird durch den Wert dieses Parameters qualifiziert. Geben Sie ein Pfadelement oder ein Muster wie „*.txt“ ein. Platzhalter sind zulässig. Der Include-Parameter ist nur dann wirksam, wenn der Befehl den Recurse-Parameter enthält oder der Pfad auf den Inhalt eines Verzeichnisses zeigt, beispielsweise „C:\Windows*“, wobei das Platzhalterzeichen den Inhalt des Verzeichnisses „C:\Windows“ angibt.
Erforderlich?
false
Position?
named
Standardwert
Pipelineeingaben akzeptieren?
false
Platzhalterzeichen akzeptieren? true
-LiteralPath <String[]>
Gibt einen Pfad zu einem oder mehreren Speicherorten an. Im Gegensatz zum Path-Parameter wird der Wert des LiteralPath-Parameters genau so verwendet, wie er eingegeben wurde. Es werden keine Zeichen als Platzhalter interpretiert. Wenn der Pfad Escapezeichen enthält, müssen Sie ihn in einfache Anführungszeichen einschließen. Einfache Anführungszeichen veranlassen WindowsPowerShell, Zeichen nicht als Escapesequenzen zu interpretieren.
Erforderlich? true
Position?
named
Standardwert
Pipelineeingaben akzeptieren? true (ByValue, ByPropertyName)
Platzhalterzeichen akzeptieren?

false

-Name [<SwitchParameter>]

Ruft nur die Namen der Elemente an den Speicherorten ab. Wenn Sie die Ausgabe dieses Befehls über die Pipeline an einen anderen Befehl übergeben, werden nur die Namen der Elemente übergeben.

Erforderlich?
false

Position?
named

Standardwert

Pipelineeingaben akzeptieren?
false

Platzhalterzeichen akzeptieren?
false

-Path <String[]>

Gibt einen Pfad zu einem oder mehreren Speicherorten an. Platzhalter sind zulässig. Der Standardspeicherort ist das aktuelle Verzeichnis (.).

Erforderlich?
false

Position? 1

Standardwert
Current directory

Pipelineeingaben akzeptieren? true (ByValue, ByPropertyName)

Platzhalterzeichen akzeptieren? true

-Recurse [<SwitchParameter>]

Ruft die Elemente an den angegebenen Speicherorten und alle untergeordneten Elemente der Speicherorte ab. In Windows PowerShell 2.0 und früheren Versionen von WindowsPowerShell funktioniert der Recurse-Parameter nur, wenn der Wert des Path-Parameters ein Container mit untergeordneten Elementen ist, z.B. „C:\Windows“ oder „C:\Windows*“. Er funktioniert nicht, wenn es sich um Elemente ohne untergeordnete Elemente handelt, z.B. „C:\Windows*.exe“.

Erforderlich?
false
Position?
named
Standardwert
False
Pipelineeingaben akzeptieren?
false
Platzhalterzeichen akzeptieren?
false
-UseTransaction [<SwitchParameter>]
Schließt den Befehl in die aktive Transaktion ein. Dieser Parameter ist nur gültig, wenn gerade eine Transaktion ausgeführt wird. Weitere Informationen finden Sie unter
Erforderlich?
false
Position?
named
Standardwert
false
Pipelineeingaben akzeptieren?
false
Platzhalterzeichen akzeptieren?
false
<CommonParameters>
Dieses Cmdlet unterstützt folgende allgemeine Parameter: "Verbose", "Debug", "ErrorAction", "ErrorVariable", "WarningAction", "WarningVariable", "OutBuffer", "PipelineVariable" und "OutVariable". Weitere Informationen finden Sie unter "about_CommonParameters" (http://go.microsoft.com/fwlink/?LinkID=113216).

EINGABEN
System.String
Sie können eine Zeichenfolge mit einem Pfad über die Pipeline an „Get-ChildItem“ übergeben.

AUSGABEN

System.Object

Der von Get-ChildItem zurückgegebene Objekttyp wird von den Objekten im Anbieterlaufwerkpfad bestimmt.

System.String

Bei Verwendung des Name-Parameters werden von Get-ChildItem die Objektnamen als Zeichenfolgen zurückgegeben.

HINWEISE

Sie können auch über die integrierten Aliase „ls“, „dir“ und „gci“ auf Get-ChildItem verweisen. Weitere Informationen finden Sie unter about_Aliases.

Get-ChildItem ruft standardmäßig keine ausgeblendeten Elemente ab. Wenn Sie ausgeblendete Elemente abrufen möchten, verwenden Sie den Force-Parameter.

Das Cmdlet Get-ChildItem ist für die Verwendung mit Daten konzipiert, die von beliebigen Anbietern verfügbar gemacht werden. Um die in der Sitzung verfügbaren Anbieter aufzuführen, geben Sie „Get-PSProvider“ ein. Weitere Informationen finden Sie unter about_Providers (http://go.microsoft.com/fwlink/?LinkID=113250).

---------- BEISPIEL 1 ----------

PS C:\>Get-ChildItem

Mit diesem Befehl werden die untergeordneten Elemente am aktuellen Speicherort abgerufen. Wenn es sich bei dem Speicherort um ein Dateisystemverzeichnis handelt, werden die Dateien und Unterverzeichnisse im aktuellen Verzeichnis abgerufen. Wenn das Element über keine untergeordneten Elemente verfügt, kehrt dieser Befehl ohne Rückmeldung zur Eingabeaufforderung zurück. In den Standardanzeigen werden der Modus (Attribute), der Zeitpunkt des letzten Schreibzugriffs, die Dateigröße (Länge) und der Name der Datei angezeigt. Die gültigen Werte für den Modus sind „d“ (Verzeichnis), „a“ (Archiv), „r“ (schreibgeschützt), „h“ (ausgeblendet) und „s“

(System).

------------------- BEISPIEL 2 ----------------------

PS C:\>Get-ChildItem –Path *.txt -Recurse -Force

Mit diesem Befehl werden alle Dateien mit der Erweiterung „.txt“ im aktuellen Verzeichnis und in seinen Unterverzeichnissen abgerufen. Der Recurse-Parameter weist WindowsPowerShell an, Objekte rekursiv abzurufen, und gibt an, dass sich der Befehl auf das angegebene Verzeichnis und dessen Inhalt bezieht. Der Force-Parameter fügt der Anzeige ausgeblendete Dateien hinzu. Wenn Sie den Recurse-Parameter in Windows PowerShell 2.0 und früheren WindowsPowerShell-Versionen verwenden möchten, muss es sich bei dem vom Path-Parameter verwendeten Wert um einen Container handeln. Geben Sie den TXT-Dateityp mithilfe des Include-Parameters an. Beispiel: Get-ChildItem –Path .* -Include *.txt -Recurse

------------------- BEISPIEL 3 ----------------------

PS C:\>Get-ChildItem –Path C:\Windows\Logs* -Include *.txt -Exclude A*

Mit diesem Befehl werden die Dateien mit der Erweiterung „.txt“ im Unterverzeichnis „Logs“ aufgelistet, mit Ausnahme der Dateien, deren Namen mit dem Buchstaben „A“ beginnen. Mit dem Platzhalterzeichen (*) wird der Inhalt des Unterverzeichnisses „Logs“ und nicht der Verzeichniscontainer angegeben. Da der Befehl den Recurse-Parameter nicht enthält, wird der Inhalt des Verzeichnisses von Get-ChildItem nicht automatisch berücksichtigt und muss explizit angegeben werden.

------------------- BEISPIEL 4 ----------------------

PS C:\>Get-ChildItem –Path HKLM:\Software

Mit diesem Befehl werden alle Registrierungsschlüssel im Schlüssel „HKEY_LOCAL_MACHINE\SOFTWARE“ in der Registrierung des lokalen Computers abgerufen.

------------------- BEISPIEL 5 ----------------------

PS C:\>Get-ChildItem -Name

Mit diesem Befehl werden nur die Namen der Elemente im aktuellen Verzeichnis abgerufen.

---------- BEISPIEL 6 ----------

PS C:\>Import-Module Microsoft.PowerShell.Security

PS C:\>Get-ChildItem –Path Cert:* -Recurse -CodeSigningCert

Mit diesem Befehl werden alle Zertifikate auf dem WindowsPowerShell-Laufwerk „Cert:“ abgerufen, die über die Codesignaturberechtigung verfügen. Mit dem ersten Befehl wird das Microsoft.PowerShell.Security-Modul in die Sitzung importiert. Dieses Modul enthält den Certificate-Anbieter, der das Laufwerk „Cert:“ erstellt.

Der zweite Befehl verwendet das Cmdlet Get-ChildItem. Der Wert des Path-Parameters ist das Laufwerk „Cert:“. Der Recurse-Parameter fordert eine rekursive Suche an. Beim CodeSigningCertificate-Parameter handelt es sich um einen dynamischen Parameter, der dem Cmdlet „Get-ChildItem“ vom Zertifikatanbieter hinzugefügt wird. Mit diesem Parameter werden nur Zertifikate abgerufen, die über die Codesignaturberechtigung verfügen.

Weitere Informationen zum Zertifikatanbieter und dem Cert:-Laufwerk finden Sie unter http://go.microsoft.com/fwlink/?LinkID=113433.
Verwenden Sie alternativ das Cmdlet „Update-Help“, um die Hilfedateien für das Microsoft.PowerShell.Security-Modul herunterzuladen, und geben Sie dann „Get-Help Certificate“ ein.

---------- BEISPIEL 7 ----------

PS C:\>Get-ChildItem –Path C:\Windows –Include *mouse* -Exclude *.png

Mit diesem Befehl werden alle Elemente im Verzeichnis „C:\Windows“ und seinen Unterverzeichnissen abgerufen, deren Dateiname „mouse“ enthält, mit Ausnahme der Elemente mit der Dateinamenerweiterung „.png“.

VERWANDTE LINKS

Online Version:

```
http://go.microsoft.com/fwlink/p/?linkid=290488
      Get-Item
      Get-Location
      Get-Process
      about_Providers
```

Im Vergleich zu einem einfachen „`Get-Help Get-ChildItem`“ beginnt der erweiterte Teil der Hilfe mit der ausführlichen Beschreibung der einzelnen Parameter. Schauen wir uns exemplarisch die Option „`-Path`“ an. „`-Path <String[]>`“ zeigt Ihnen durch <String[]> an, dass Sie einen oder mehrere Parameterwerte angeben können. Diese sind durch Kommata zu trennen. Also z.B. ist „`Get-ChildItem -Path c:\, c:\windows`“ möglich. Der Parametername „`-Path`” ist nicht verpflichtend anzugeben. Allerdings müssen die Werte dann dazu an Position 1 stehen. Da der Standardwert der aktuelle Pfad ist, muss nur ein Wert angegeben werden, wenn man einen anderen Pfad abfragen möchte. Befindet sich die PowerShell-Konsole in C:\Windows\system32, dann würden Sie logischerweise eine Auflistung aller Ordner und Dateien von system32 erhalten, wenn Sie „`Get-ChildItem`“ ohne weitere Parameter aufrufen. Sie können auch Platzhalterzeichen wählen. „`Get-ChildItem -Path c:\pro*`“ wäre somit eine gültige Eingabe.

```
PS C:\> Get-ChildItem -Path c:\pro*


    Verzeichnis: C:\


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
d-r--        17.04.2015     09:08            Program Files
d-r--        26.03.2015     16:56            Program Files (x86)



PS C:\> |
```

Abbildung 1.20: Get-ChildItem mit Wildcard

Nehmen wir noch den Parameter „`-Recurse`“ hinzu. „`-Recurse`“ ist nicht erforderlich, er kann prinzipiell an jeder Stelle in der Syntax stehen, da der Name angegeben werden muss. Das sehen Sie am Eintrag „Position?“: named. Es gibt weder einen Standardwert, noch können Inhalte aus der Pipeline übergeben werden oder Platzhalterzeichen verwendet werden.

So können Sie sich Ihre Syntax Schritt für Schritt erarbeiten. Auch später, wenn Sie eine gewisse Erfahrung haben, werden Sie bei neuen Cmdlets keine Probleme haben diese anzuwenden.

Praxis-Tipp: Am Ende jedes Commandlets stehen unter „Verwandte Links“ noch wertvolle Hinweise. Sie finden dort beispielsweise den URL zur Online-Version des Commandlets sowie eine kleine Liste von Cmdlets, die für eine Weiterverarbeitung der Daten nützlich sein können.

Bei „`Get-ChildItem`“ sieht das so aus:

```
VERWANDTE LINKS
        Online Version:
http://go.microsoft.com/fwlink/p/?linkid=290488
        Get-Item
        Get-Location
        Get-Process
        about_Providers
```

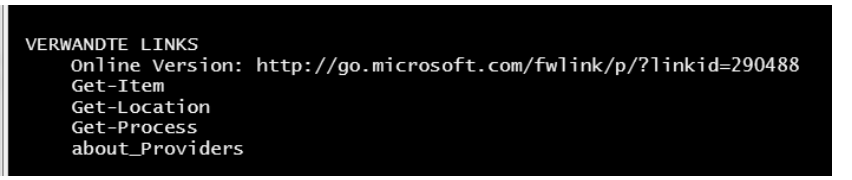


Abbildung 1.21: Verwandte Links von Get-ChildItem

1.8 Cmdlets finden

Mit „`Get-Command`“ können Sie alle gegenwärtig vorhandenen Module nach einem bestimmten Begriff durchsuchen. Nehmen wir an, Sie möchten eine Active Directory Gruppe per PowerShell anlegen. Aber Sie kennen das oder die dazugehörigen Cmdlets nicht. Laden Sie zuerst das Active Directory Modul. Das RSAT haben Sie ja schon installiert. Verwenden Sie dazu den Befehl „`Import-Module ActiveDirectory`“. Das Modul wird nun geladen. Dies dauert einen kurzen Moment. Dann können Sie mit „Get-Command“ und Wildcards danach suchen. „`Get-Command *group*`“ gibt Ihnen als Resultat alle vorhandenen Commandlets zurück, die „group“ enthalten und in den geladenen PowerShell-Modulen enthalten sind. Beginnend bei Add-ADGroupMember über New-ADGroup zu Test-SqlAvailabilityGroup. Um eine neue Active Directory Gruppe anzulegen sieht „`New-ADGroup`“ doch ganz gut aus? ;-)

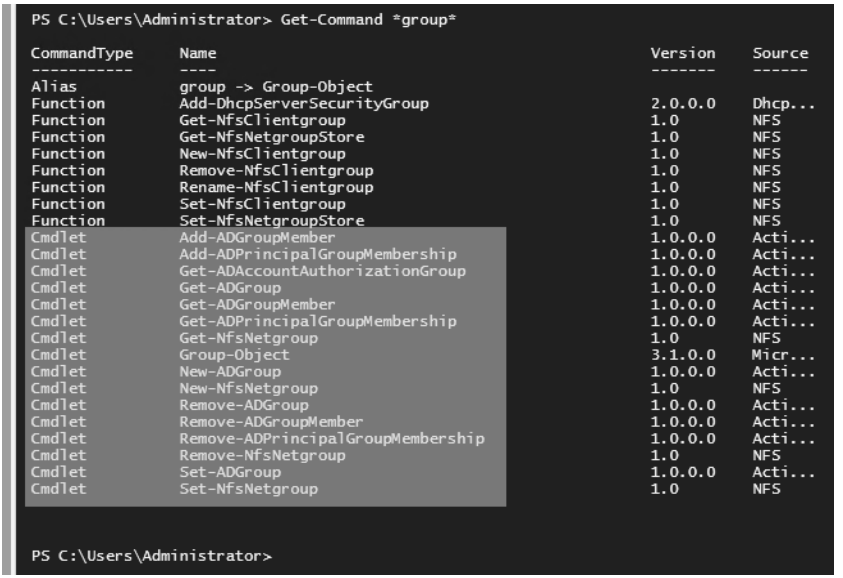


*Abbildung 1.22: Ergebnis von „Get-Command *group*“*

Schauen Sie sich jetzt mit „`Get-Help New-ADGroup -Full`” die Syntax des Cmdlets genauer an. Welche Parameter sind **erforderlich**?

„`Get-Command`“ hat den interessanten Parameter „`-Module`“. Damit können Sie direkt in einem Module nach Cmdlets suchen. Auch wenn das Module nicht geladen ist. Auf das jüngste Beispiel angewandt lautet die Befehlszeile dann „`Get-Command –Module ActiveDirectory *group*`“

Abbildung 1.23: Suche in einem speziellen Modul

„`Get-Command`“ listet Ihnen nicht nur alle Commandlets Ihres Rechners und von importierten Sitzungen entfernter Systeme auf, sondern auch die Anwendungen auf die der Suchbegriff passt! „`Get-Command *pin*`“ liefert Ihnen zu den Cmdlets auch pathping.exe, ping.exe, spinstall.exe etc.

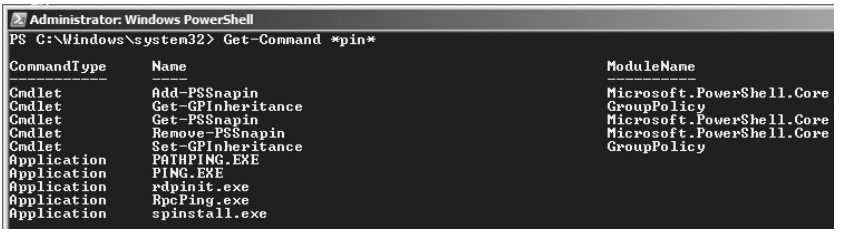


Abbildung 1.24: Get-Command liefert auch Nicht-Cmdlets zurück

1.9 Wichtige Cmdlets

Wie schon erwähnt kann man sich mit ein paar wenigen Commandlets die meisten anderen herleiten, bzw. sich ihnen Schritt für Schritt annähern.[10]

1.9.1 Get-ChildItem

„`Get-ChildItem`“ haben Sie schon im Abschnitt 5 genauer kennengelernt.

1.9.2 Get-Command

„`Get-Command`“ wurde schon im Abschnitt 8 näher beschrieben. Dieser Befehl und seine Parameter sind Dreh- und Angelpunkt zur Suche nach Commandlets.

1.9.3 Get-Help

„`Get-Help`“ wurde schon in Abschnitt 4 ausführlich behandelt. Es ist für Sie DAS Commandlet mit dem Sie an alle Informationen zu jedem anderen Cmdlets kommen. Verwenden Sie zu Beginn Ihres Lernprozesses immer den Parameter „`-Full`“. Damit haben Sie automatisch alle Erklärungen und Beispiele im Blick.

1.9.4 Get-Member

Mit „`Get-Member`“ können Sie sich die kompletten Eigenschaften eines Commandlets anzeigen lassen. PowerShell hat die (un-)angenehme Eigenschaft eine voreingestellte Vorauswahl der erzeugten Objekte zu verwenden.

Probieren Sie das an Hand eines Beispiels aus. „`Get-ChildItem`“ ist dazu ideal. Als Ergebnis erhalten Sie eine Auflistung aller Dateien im

[10] Eine Übersicht von grundlegenden Commandlets samt Aliasen finden Sie hier: http:// http://blogs.technet.com/b/heyscriptingguy/archive/2015/06/11/table-of-basic-powershell-commands.aspx

angegebenen Pfad mit „Mode, LastWriteTime, Length und Name“. Das ist aber nicht alles, was Get-ChildItem an Eigenschaften zu bieten hat. Verketten Sie „`Get-ChildItem`“ mittels der Pipeline mit „`Get-Member`“ (Details zur Pipeline im nächsten Abschnitt).

```
Get-ChildItem | Get-Member
```

Sie sehen in der Ergebnisliste eine Aufzählung der MemberTypes. Alles was mit „Property“ gekennzeichnet ist, können Sie sich mit „`Get-ChildItem`“ anzeigen lassen. Das ist quasi alles schon dabei. „CreationTime, Exists, LastAccessTime“ usw.

```
PS C:\Users\Administrator> Get-ChildItem | Get-Member


   TypeName: System.IO.DirectoryInfo

Name                      MemberType     Definition
----                      ----------     ----------
Mode                      CodeProperty   System.String Mode{get=Mode;}
Create                    Method         void Create(), void Create(System.Securit...
CreateObjRef              Method         System.Runtime.Remoting.ObjRef CreateObjR...
CreateSubdirectory        Method         System.IO.DirectoryInfo CreateSubdirector...
Delete                    Method         void Delete(), void Delete(bool recursive)
EnumerateDirectories      Method         System.Collections.Generic.IEnumerable[Sy...
EnumerateFiles            Method         System.Collections.Generic.IEnumerable[Sy...
EnumerateFileSystemInfos  Method         System.Collections.Generic.IEnumerable[Sy...
Equals                    Method         bool Equals(System.Object obj)
GetAccessControl          Method         System.Security.AccessControl.DirectorySe...
GetDirectories            Method         System.IO.DirectoryInfo[] GetDirectories(...
GetFiles                  Method         System.IO.FileInfo[] GetFiles(string sear...
GetFileSystemInfos        Method         System.IO.FileSystemInfo[] GetFileSystemI...
GetHashCode               Method         int GetHashCode()
GetLifetimeService        Method         System.Object GetLifetimeService()
GetObjectData             Method         void GetObjectData(System.Runtime.Seriali...
GetType                   Method         type GetType()
InitializeLifetimeService Method         System.Object InitializeLifetimeService()
MoveTo                    Method         void MoveTo(string destDirName)
Refresh                   Method         void Refresh()
SetAccessControl          Method         void SetAccessControl(System.Security.Acc...
ToString                  Method         string ToString()
PSChildName               NoteProperty   string PSChildName=Contacts
PSDrive                   NoteProperty   PSDriveInfo PSDrive=C
PSIsContainer             NoteProperty   bool PSIsContainer=True
PSParentPath              NoteProperty   string PSParentPath=Microsoft.PowerShell....
PSPath                    NoteProperty   string PSPath=Microsoft.PowerShell.Core\F...
PSProvider                NoteProperty   ProviderInfo PSProvider=Microsoft.PowerSh...
Attributes                Property       System.IO.FileAttributes Attributes {get;...
CreationTime              Property       datetime CreationTime {get;set;}
CreationTimeUtc           Property       datetime CreationTimeUtc {get;set;}
Exists                    Property       bool Exists {get;}
Extension                 Property       string Extension {get;}
FullName                  Property       string FullName {get;}
LastAccessTime            Property       datetime LastAccessTime {get;set;}
LastAccessTimeUtc         Property       datetime LastAccessTimeUtc {get;set;}
LastWriteTime             Property       datetime LastWriteTime {get;set;}
LastWriteTimeUtc          Property       datetime LastWriteTimeUtc {get;set;}
```

Abbildung 1.25: Ergebnis von „Get-ChildItem | Get-member“ (Ausschnitt)

Praxis-Tipp: Eine interessante Variante zu „`Get-Member`“ ist in einigen Fällen die Verwendung von „`Format-List *`“. Der Unterschied liegt darin, dass Sie zu den Eigenschaften zusätzlich die tatsächlichen Werte dazu erhalten. Damit haben Sie die Möglichkeit zu vergleichen, ob Ihre Eingaben auch dem entsprechen, was der Parameter auch verarbeiten kann.

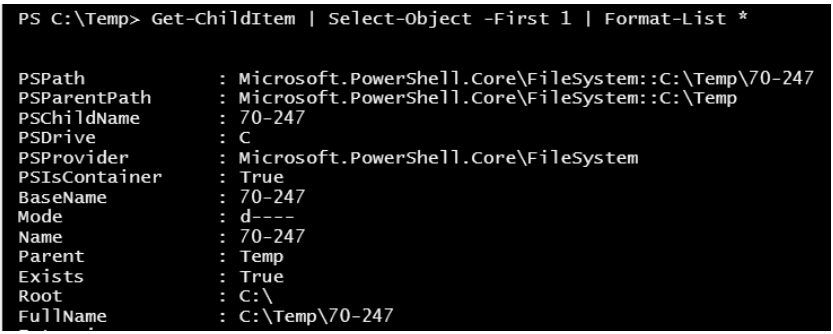


*Abbildung 1.26: Get-ChildItem | Format-List * (Ausschnitt)*

Ein passendes Beispiel für die Verwendung von Format-List ist die WMI-Abfrage über die logischen Platten in einem System. Das Commandlet dazu wäre „`Get-WmiObject -Class win32_logicaldisk | Format-List *`“. Interessant ist die Unterscheidung des DriveTypes. Eine (USB-)Festplatte hat als DriveType den Wert “3”. Das DVD-Laufwerk hat den Wert „5“.

1.9.5 Start-Transcript

Eine prima Methode sich eine Aufzeichnung einer PowerShell-Sitzung zu erstellen ist das Cmdlet „`Start-Transcript`“. Damit können Sie alle Befehle und deren Rückgabe in eine Text-Datei schreiben lassen. Starten Sie die Aufzeichnung mit „Start-Transcript –Path C:\temp\ausgabe.txt“ und alle weiteren Aktionen innerhalb dieser geöffneten Konsole werden zusätzlich in die angegebene Datei herausgeschrieben.

Abbildung 1.27: Start der Aufzeichnung

Mit „`Stop-Transcript`“ beenden Sie die Aufzeichnung. Danach können Sie in der Datei unter dem angegebenen Pfad Ihre gesamte PowerShell-Historie dieser PowerShell-Sitzung nachlesen.

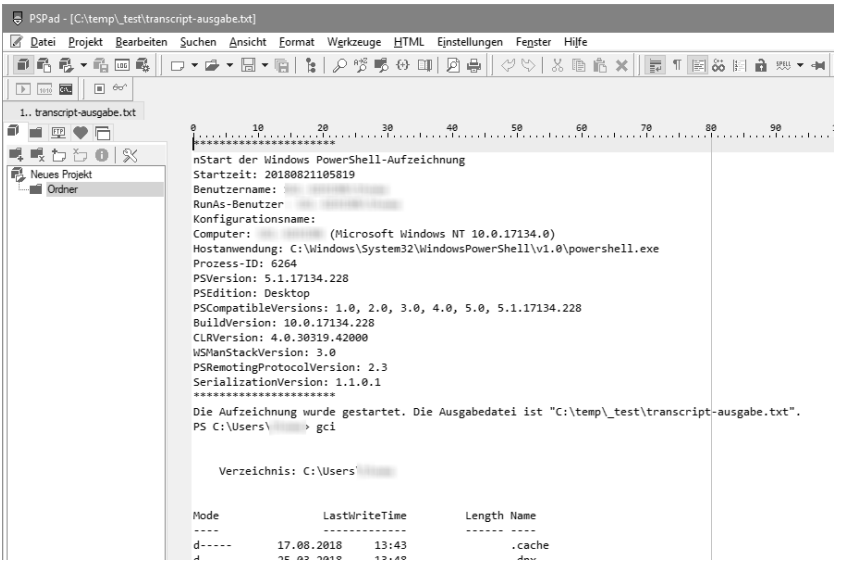


Abbildung 1.28: Ausgabe in der Text-Datei

1.10 Die Pipeline (vereinfacht)

Die Pipeline ermöglicht es Ihnen zwei oder mehrere PowerShell-Befehle miteinander und aufeinanderfolgend zu verketten. Sie können hiermit das Ergebnis des einen Befehls direkt an den nächsten übergeben, damit dieser weitere Aktionen mit den zuvor erzeugten Daten durchführt. Sie haben dadurch eine Art Schweizer Taschenmesser des Windows-Skripting zur Verfügung!

Bauen wir ein simples Beispiel schrittweise auf:

„`Get-ChildItem`“ gibt bekanntermaßen den Inhalt des gegenwärtigen Verzeichnisses zurück. Die Eigenschaften, die standardmäßig zurückgegeben werden sind „Mode, LastWriteTime, Length und Name“[11].

[11] Die weiteren Eigenschaften von „`Get-ChildItem`“ erhalten Sie über „`Get-ChildItem | Get-Member -MemberType Property`“.

```
PS C:\Temp\test tage\jre1.7.0_65> Get-ChildItem


    Verzeichnis: C:\Temp\test tage\jre1.7.0_65


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---        25.07.2014     10:47   25682589 Data1.cab
-a---        25.07.2014     10:47     919552 jre1.7.0_65.msi
-a---        25.07.2014     10:47      40448 jre1031.MST
```

Abbildung 1.29: Ergebnis von Get-ChildItem

Sie wollen jetzt aus dem Inhalt des Verzeichnisses nur alle Dateien angezeigt bekommen, die größer sind als 1 GB (= Datei-Größe = length), weil Sie alle ISO-Dateien des Ordners haben möchten. Dazu benötigen Sie das Commandlet „`Where-Object`“. Ein kurzer Blick in die Hilfe eröffnet Ihnen zwei Möglichkeiten. Wenn Sie nur eine Filterbedingung angeben wollen, dann genügt Ihnen der Parameter „`-Property`“.

„`Get-ChildItem | Where-Object -Property Length -gt -Value '1000000000'`“

Möchten Sie jedoch mehr als eine Filterbedingung verwenden, dann ist der Parameter „`-Filterscript`“ für Sie genau richtig. Die Syntax ist anfangs etwas sperrig, aber Sie werden sich schnell daran gewöhnen. Sie müssen die Filterbedingungen als Skriptblock komplett in geschweifte Klammern fassen.[12] Die einzelnen Eigenschaften werden mit vorangestelltem „`$_`“ angegeben. Für die Länge also: „`$_.Length`“. (Für Name dann analog „`$_.Name`“ etc.)

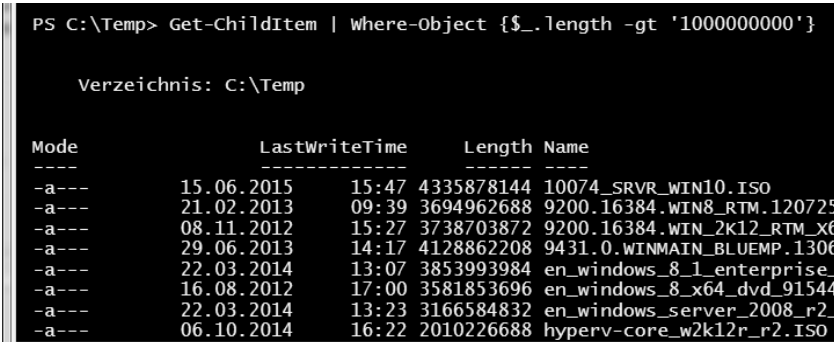
„`Get-ChildItem | Where-Object -Filterscript {$_.length -gt '1000000000'}`“[13]

Konkret werden zuerst mit „`Get-ChildItem`“ alle Objekte des gegenwärtigen Verzeichnisses eingelesen. Die Daten werden aber gleich nach Abschluss von „`Get-ChildItem`“, und bevor Sie diese am Bildschirm zu sehen bekommen, nacheinander über die Pipeline an den Befehl „`Where-Object`“ übergeben. „`Where-Object`“ prüft nun, ob die einzelnen Objekte größer (= „`-gt`“) als 1 GB sind. Nur diese werden

[12] Das Konstrukt mit den geschweiften Klammern läßt sich noch beliebig erweitern. Beispielsweise mit {$_.Length -gt '1000000000' –and $_.Name –match 'en_windows'}. Damit würden nur alle Dateien angezeigt werden, die sowohl größer als 1 GB sind UND im Namen „en_windows“ stehen haben.

[13] Der Parameter „-Filterscript“ ist positional an erster Stelle und eine Pflichtangabe. Daher kann er weggelassen werden.

am Bildschirm angezeigt. Alle Objekte kleiner als 1 GB fallen aus der Pipeline raus.



```
PS C:\Temp> Get-ChildItem | Where-Object {$_.length -gt '1000000000'}

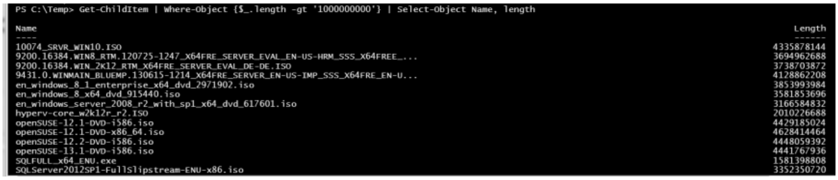

    Verzeichnis: C:\Temp


Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---        15.06.2015     15:47 4335878144 10074_SRVR_WIN10.ISO
-a---        21.02.2013     09:39 3694962688 9200.16384.WIN8_RTM.120725
-a---        08.11.2012     15:27 3738703872 9200.16384.WIN_2K12_RTM_X6
-a---        29.06.2013     14:17 4128862208 9431.0.WINMAIN_BLUEMP.1306
-a---        22.03.2014     13:07 3853993984 en_windows_8_1_enterprise_
-a---        16.08.2012     17:00 3581853696 en_windows_8_x64_dvd_91544
-a---        22.03.2014     13:23 3166584832 en_windows_server_2008_r2_
-a---        06.10.2014     16:22 2010226688 hyperv-core_w2k12r_r2.ISO
```

Abbildung 1.30: Ergebnis Get-ChildItem ab bestimmter Größe

Ein weiterer Schritt könnte nun sein, dass Sie statt der Standardausgabe von „mode, LastTimeWrite, Length, Name“ nur „Name“ vor „Length“ haben möchten. Dies erreichen Sie über eine weitere Pipeline und das Cmdlet „`Select-Object`“.

```
Get-ChildItem | Where-Object -Filterscript {$_.length -gt '1000000000'} | Select-Object name, length
```



```
PS C:\Temp> Get-ChildItem | Where-Object {$_.length -gt '1000000000'} | Select-Object Name, length

Name                                                                          Length
----                                                                          ------
10074_SRVR_WIN10.ISO                                                      4335878144
9200.16384.WIN8_RTM.120725-1247_X64FRE_SERVER_EVAL_EN-US-HRM_SSS_X64FREE_... 3694962688
9200.16384.WIN_2K12_RTM_X64FRE_SERVER_EVAL_DE-DE.ISO                      3738703872
9431.0.WINMAIN_BLUEMP.130615-1214_X64FRE_SERVER_EN-US-IMP_SSS_X64FRE_EN-U... 4128862208
en_windows_8_1_enterprise_x64_dvd_2971902.iso                             3853993984
en_windows_8_x64_dvd_915440.iso                                           3581853696
en_windows_server_2008_r2_with_sp1_x64_dvd_617601.iso                     3166584832
hyperv-core_w2k12r_r2.ISO                                                 2010226688
openSUSE-12.1-DVD-i586.iso                                                4429185024
openSUSE-12.1-DVD-x86_64.iso                                              4628414464
openSUSE-12.2-DVD-i586.iso                                                4448059392
openSUSE-13.1-DVD-i586.iso                                                4441767936
SQLFULL_x64_ENU.exe                                                       1581398808
SQLServer2012SP1-FullSlipstream-ENU-x86.iso                               3352350720
```

Abbildung 1.31: Ergebnis mit Anzeige des Namens und Länge

Als finalen Schritt hätten Sie gern noch die umgekehrte Sortierreihenfolge? Kein Problem. Einfach die Objekte der bisherigen Commandlets in ein weiteres schicken. „`Sort-Object`“ ist der richtige Befehl dazu.

```
Get-ChildItem | Where-Object -Filterscript {$_.length -gt '1000000000'} | Select-Object name, length | Sort-Object name -Descending
```

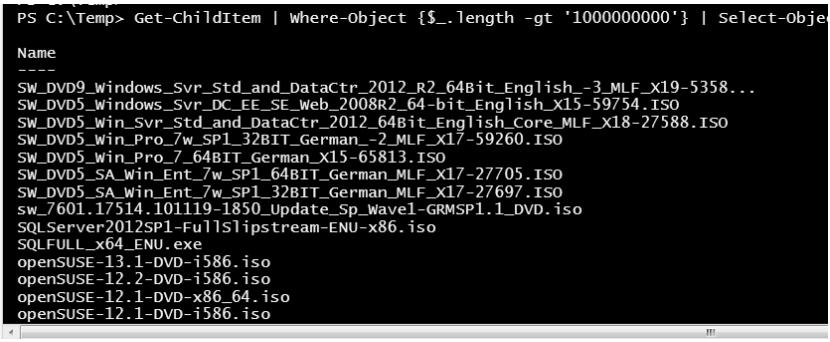


Abbildung 1.32: Ergebnis in umgekehrter Namen-Reihenfolge sortiert (Ausschnitt)

Sie benötigen für die einzelnen Schritte keine Zwischenspeicherung in Dateien.

PowerShell ist so simpel konzipiert: Hole mir das, prüfe auf das, verwerfe die folgenden Eigenschaften und sortiere nach Namen in umgekehrter Reihenfolge und zeige mir dann alles am Bildschirm an.

Praxis-Tipp: wenn Sie einmal die Objekte Ihrer Pipelinekette in eine Datei herausschreiben oder über einen der Format-Befehle geordnet am Bildschirm anzeigen lassen oder in eine Datei rausschreiben lassen, dann können Sie danach keine weiteren Commandlets verketten. Die Daten sind dann keine Objekte mehr. Die Ausgabe kann Ihnen unter Umständen sehr, sehr seltsame Ergebnisse liefern.

Sehen Sie sich den Unterschied zwischen

“`Get-ChildItem | Where-Object -Filterscript {$_.length -gt '1000000000'} | Select-Object name, length | Sort-Object name -Descending | Out-Gridview`”

und

“`Get-ChildItem | Where-Object -Filterscript {$_.length -gt '1000000000'} | Select-Object name, length | Sort-Object name -Descending | Format-Table –Autosize | Out-Gridview`”

an.

Im letzten Fall kann “Out-Gridview” nichts mehr mit dem ankommenden Format anfangen.

„`out-gridview : **Das Datenformat wird von "Out-GridView" nicht unterstützt.**`

```
In Zeile:1 Zeichen:146
+ ... le -AutoSize | out-gridview
+                    ~~~~~~~~~~~~
```

```
    + CategoryInfo              : InvalidType: (:)
[Out-GridView], FormatException
    + FullyQualifiedErrorId     :
DataNotQualifiedForGridView,Microsoft.PowerShell.Comma
nds.OutGridViewCommand".
```

```
PS C:\Temp> Get-ChildItem | Where-Object {$_.length -gt '1000000000'} | Select-
Out-GridView : Das Datenformat wird von "Out-GridView" nicht unterstützt.
In Zeile:1 Zeichen:146
+ ... le -AutoSize | Out-GridView
+                    ~~~~~~~~~~~
    + CategoryInfo          : InvalidType: (:) [Out-GridView], FormatException
    + FullyQualifiedErrorId : DataNotQualifiedForGridView,Microsoft.PowerShell.
```

Abbildung 1.33: Ergebnis nach einem Format-Table (Ausschnitt)

1.11 Filterung

Im vorangegangenen Kapitel haben wir bereits, ohne es groß zu erwähnen, Daten gefiltert. Ausgehenden von allen Dateien im Ordner C:\Temp wurden mittels Filterung nur Dateien angezeigt, die größer als 1 GB sind.

`Get-ChildItem | Where-Object -Property Length -gt -Value '1000000000'`

Betrachten wir eine weitere Filterung. Im Verzeichnis C:\Temp_test liegen mehrere csv-Dateien, einige png-Dateien und eine stattliche Zahl an txt-Dateien.

„`Get-ChildItem -Path C:\temp\_test\`“ listet alle Dateien auf der ersten Ebene auf.

Um nur die png-Dateien angezeigt zu bekommen werden die Inhalte Get-ChildItem gefiltert. Dies setzen wir mittels einer Pipeline und dem Cmdlet „Where-Object“ um.

```
Get-ChildItem | Where-Object {$_.extension -eq '.png'}
```

```
PS C:\temp\_test> Get-ChildItem | Where-Object {$_.extension -eq '.png'}


    Verzeichnis: C:\temp\_test


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----       17.08.2018     13:46         386820 01.png
-a----       17.08.2018     13:46         386820 02.png
-a----       17.08.2018     13:46         386820 03.png
-a----       17.08.2018     13:46         386820 04.png


PS C:\temp\_test>
```

Abbildung 1.34: Gefiltert nach png-Dateien

Ein alternativer Weg ist eine Filterung innerhalb des Commandlets „`Get-`

ChildItem“.

```
Get-ChildItem -Filter "*.png"
```

```
PS C:\temp\_test> Get-ChildItem -Filter "*.png"


    Verzeichnis: C:\temp\_test


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        17.08.2018     13:46         386820 01.png
-a----        17.08.2018     13:46         386820 02.png
-a----        17.08.2018     13:46         386820 03.png
-a----        17.08.2018     13:46         386820 04.png


PS C:\temp\_test>
```

Abbildung 1.35: Filterung innerhalb Get-ChildItem

Diese Methode ist schneller, als die erste Variante. Sie hat allerdings den Nachteil, dass Sie nur jeweils einen Filterwert angeben können. Mit dem Weg über „Where-Object“ können Sie mehrere Filterwerte angeben.

```
Get-ChildItem | Where-Object {$_.extension -eq '.png' -or $_.extension -eq '.csv'}
```

```
PS C:\temp\_test> Get-ChildItem | Where-Object {$_.extension -eq '.png' -or $_.extension -eq '.csv'}


    Verzeichnis: C:\temp\_test


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        16.08.2018     12:49           2304 0.csv
-a----        17.08.2018     13:46         386820 01.png
-a----        17.08.2018     13:46         386820 02.png
-a----        17.08.2018     13:46         386820 03.png
-a----        17.08.2018     13:46         386820 04.png
-a----        14.08.2018     12:51           2304 2.csv
-a----        14.08.2018     12:51           2304 3.csv
-a----        14.08.2018     12:51           2304 4.csv
-a----        14.08.2018     12:51           2304 5 (5).csv
-a----        14.08.2018     12:51           2304 5.csv


PS C:\temp\_test>
```

Abbildung 1.36: Filterung nach png- und csv-Dateien

Schick ist auch die damit verbundene Möglichkeit zusätzlich nach dem Erstelldatum oder Änderungsdatum zu filtern. Dank „Get-Member“ wissen wir, dass es bei „Get-ChildItem“ eine Eigenschaft „CreationTime“ gibt. Das läßt sich mit anderen Eigenschaften verknüpfen.

```
Get-ChildItem | Where-Object {$_.extension -eq '.txt' -and $_.Creationtime -lt ((get-date).AddDays(-2))}
```

So können Sie sich alle txt-Dateien anzeigen lassen, deren Erstelldatum älter ist als zwei Tage.

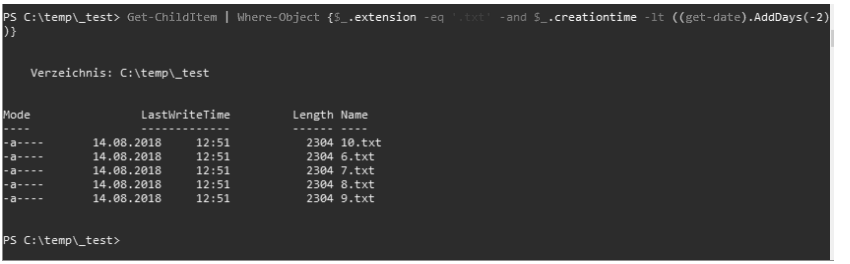


Abbildung 1.37: Auch nach Erstelldatum gefiltert

Damit haben Sie die Möglichkeit mit einem „Einzeiler“ in einem ganzen Verzeichnis samt Unterordner alle Dateien zu löschen, die älter als x Tage sind. (Hier: älter als 30 Tage)

```
Get-ChildItem –Path c:\Temp\_test –recurse -force | Where-Object {$_.Creationtime -lt ((get-date).AddDays(-30))} | Remove-item –force –recurse
```

Praxis-Tipp: Filtern Sie stets möglichst „Links“ und innerhalb eines Commandlets. Das ist schneller, als der Weg über die Pipeline.

Zum Beispiel hat „`Get-ChildItem`“ dafür den Parameter „`-Filter`“. Selbst bei so einem kleinen Vergleich wie der Filterung nach csv-Dateien ist die „rechte“ Filterung zweieinhalb Mal langsamer.

```
PS C:\temp\_test> measure-command {Get-ChildItem -Filter "*.csv" }

Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 18
Ticks             : 187772
TotalDays         : 2,17328703703704E-07
TotalHours        : 5,21588888888889E-06
TotalMinutes      : 0,000312953333333333
TotalSeconds      : 0,0187772
TotalMilliseconds : 18,7772


PS C:\temp\_test> measure-command {Get-ChildItem | Where-Object {$_.extension -eq '.csv'} }
```

```
Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 50
Ticks             : 508368
TotalDays         : 5,88388888888889E-07
TotalHours        : 1,41213333333333E-05
TotalMinutes      : 0,00084728
TotalSeconds      : 0,0508368
TotalMilliseconds : 50,8368
```

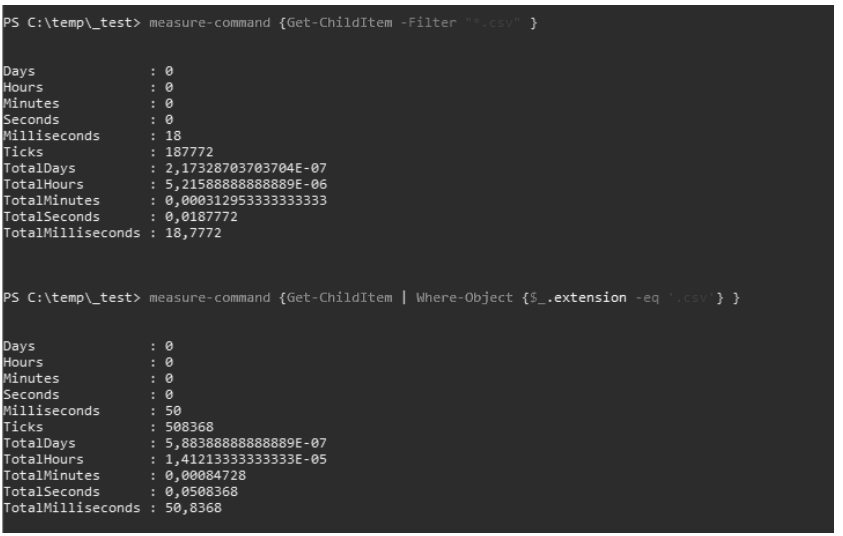


Abbildung 1.38: Zeitlicher Vergleich der Filterung

1.12 Alias

PowerShell bietet Ihnen auch die Verwendung und Erstellung[14] von Aliasen[15] an. Einige sind bereits standardmäßig definiert. Diese sollen dem Anfänger mit Batch- und/ oder bash-Erfahrung dabei helfen einen leichteren Einstieg in PowerShell zu finden. Beispielsweise sind „`Dir`“ und „`ls`“ Aliase für „`Get-ChildItem`“, die alle drei das gleiche

[14] Set-Alias ist hier das Commandlet Ihrer Wahl.

[15] Get-Alias listet Ihnen alle eingebauten Aliase auf.

Ergebnis liefern.

Praxis-Tipp: Sie dürfen gern einen Alias nutzen. Allerdings gilt hier ausschließlich die PowerShell-Syntax. „`Dir /s`“ für die rekursive Auflistung der Verzeichnisse wird nicht funktionieren. Dementsprechend führt „`Dir –recurse`“ zum Erfolg. Lassen Sie sich überraschen, was bei „`Get-Help dir`“ als Ergebnis zurückkommt.

Noch ein Praxis-Tipp: Verzichten Sie auf die Verwendung von Standard-Aliasen im Allgemeinen und von Selbsterstellten im Besonderen, wenn Sie noch Anfänger sind oder später kleinere Skripte erstellen, die evtl. auch Kollegen nutzen. Irgendein armer Kerl wird in einem Jahr oder noch später etwas ändern oder erweitern müssen. Dieser arme Kerl können auch Sie selbst sein. :-) Und da ist es besser die volle ungekürzte Syntax vor sich zu haben.

Anekdote: Erinnern Sie sich an den „Superfish“-Vorfall 2015 eines größeren Rechner-Herstellers? Es gab ziemlich schnell eine Anleitung mit Bildern, um diese falschen Zertifikate zu entfernen. Diese war mehrere Seite lang. Jeffrey Snover, der Vater von PowerShell, hat innerhalb kurzer Zeit einen PowerShell-Einzeiler über Twitter verbreitet, der komplett aus Aliasen bestand und das Problem gleich für alle Rechner im Netzwerk löste.

```
icm -cn (cat computers.txt) {dir cert:\ -rec | where subject -match 'superfish' | remove-item}
```

Die komplett ausgeschriebene Version dazu:

```
Invoke-Command -ComputerName (Get-Content computers.txt) {Get-ChildItem cert:\ -Recurse | Where-Object {$_.subject -match 'superfish'} | remove-item}
```

Immer noch ein Einzeiler, aber nach längerer Zeit besser und einfacher zu lesen, wenn Skripting und Programmierung nicht das tägliche Brot ist. ;-)

1.13 Fehlermeldungen

Über kurz oder lang werden Sie eine Fehlermeldung erhalten, wenn Sie ein Commandlet ausführen. Diese sehen in ihrer roten Farbe auf den ersten Blick anfangs ziemlich angsteinflößend aus.

Sie sind in den meisten Fällen ziemlich aufschluss- und hilfreich, wenn man sie aufmerksam liest.

Wenn Sie den Bindestrich bei einem Cmdlet vergessen werden Sie dies als Fehlermeldung bekommen:

„new: Die Benennung "new" 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 enthalten), und wiederholen Sie den Vorgang.

In Zeile:1 Zeichen:1

+ new adgroup

+ ~~~

+ CategoryInfo : ObjectNotFound: (new:String) [], CommandNotFoundException

+ FullyQualifiedErrorId : CommandNotFoundException"

Das ist eindeutig. "New" allein ist keine korrekte Syntax.

Vergessen Sie bei „Get-ChildItem Path C:\Temp" den Bindestrich vor Path bekommen Sie folgende Rückmeldung.

Get-ChildItem : **Der Pfad "C:\Temp\Path" kann nicht gefunden werden, da er nicht vorhanden ist.**

In Zeile:1 Zeichen:1

+ Get-ChildItem Path C:\Temp

+ ~~~~~~~~~~~~~~~~~~~~~~~~~

+ CategoryInfo: ObjectNotFound: (C:\Temp\Path:String) [Get-ChildItem], ItemNotFoundException

+ FullyQualifiedErrorId: PathNotFound,Microsoft.PowerShell.Commands.GetChildItemCommand

Auch hier ist die Sache eindeutig. „Der Pfad C:\temp\path kann nicht gefunden werden."

Sollten Sie eine Fehlermeldung erhalten, dann lesen Sie diese zuerst aufmerksam durch. Die Lösung steht, wie schon geschrieben, sehr oft deutlich im Text. Sollte diese einmal nicht der Fall sein, dann hilft es in der Regel die Fehlermeldung in der Suchmaschine Ihres Vertrauens einzugeben.

Haben Sie einige Commandlets verkettet, dann gehen Sie einen Schritt zurück nach links. Prüfen Sie, ob vor der letzten Verkettung noch Daten ausgegeben werden.

Sie haben im Abschnitt 8 die folgende Kette samt Fehlermeldung schon kennen gelernt.

"Get-ChildItem | Where-Object -Filterscript {$_.length

`-gt '1000000000'} | Select-Object name, length | Sort-Object name -Descending | Format-Table –Autosize | Out-Gridview`"

Gehen Sie nun zur Fehlersuche einen Schritt "nach links".

"`Get-ChildItem | Where-Object -Filterscript {$_.length -gt '1000000000'} | Select-Object name, length | Sort-Object name -Descending | Format-Table –Autosize`"

Sie erhalten wieder eine sinnvolle Ausgabe und keine Fehlermeldung. Also liegt der Fehler bei der Übergabe der bisher erzeugten Daten von „`Format-Table`" zu „`Out-GridView`".

1.14 Übungen

Mit den vorangegangenen Kapiteln haben Sie nun das grundsätzliche Rüstzeug, um erste Aufgaben mit PowerShell umzusetzen. Die folgenden Übungen sollen Ihnen noch ein bißchen Praxis vermitteln. Probieren Sie ruhig den einen oder anderen Weg aus. Es führen fast immer mehrere Ansätze zu einer Lösung. Für die Übungen mit den Active Directory Commandlets benötigen Sie die RSAT-Tools auf Ihrem Rechner und ein Active Directory. Sie erkennen die jeweiligen Übungen schnell daran, dass nach dem Bindestrich das Präfix „AD" vorangestellt ist. Beispielsweise: „`Get-`**`AD`**`User`". Die Übungen wurden auf einem Windows 10 v1803 Rechner getestet. Ein paar dieser Übungen gehen dann nicht unter Windows 7/ 8, weil es da die Cmdlets noch nicht gab.

- Lassen Sie sich alle eingebundenen Drucker an Ihrem Rechner anzeigen.
- Welche Festplatten haben Sie im/ am Rechner? Wie groß sind diese und wieviel Platz ist auf diesen noch frei? Welches Dateisystem haben diese? (Tipp: ein Commandlet genügt für alle vier Fragen)
- Finden Sie alle „.jpg"-Dateien auf der Festplatte und **kopieren** Sie diese nach c:\temp\alle-jpg.
- Wann wurde bei Ihrem AD-Benutzer das letzte Mal das Passwort geändert?
- Welche Dateien auf Ihrem Rechner haben den Bestandteil „ume" im Namen? Wenn es bei der Ausgabe zu Fehlermeldungen kommen sollte: was sind die Ursachen?
- Wie heißt das Commandlet mit dem Sie auf einen existierenden Pfad prüfen können? (Tipp: Sie möchten etwas „**test**en".)
- Was ist die PowerShell-Entsprechung zu „Ping"? Wie lautet die

Parameter, um dieses Cmdlet „still“ und nur einmal auszuführen? (Tipp: Sie möchten etwas „**test**en“.)

- Erstellen Sie einen neuen Ordner namens „blasfasel“ unterhalb von C:\Temp.
- Lassen Sie sich alle Dienste anzeigen, die im Status „Running“ sind.
- Lassen Sie sich alle Dienste anzeigen, die automatisch gestartet werden, aber nicht gestartet sind.
- Lassen Sie sich alle exe-Dateien unterhalb c:\windows\system32 anzeigen, die größer als 5 MB sind.
- Lassen Sie sich alle Prozesse anzeigen, die als „Company“ **nicht** Microsoft haben.
- Wie viele Benutzer gibt es in Ihrem Active Directory? (Tipp: Wenn Sie die Syntax gefunden haben, welche Ihnen die Benutzer auflistet, dann setzen Sie diese in runde Klammern und hängen Sie ein „`.count`“ daran: `(hier Ihre Syntax).count`)
- Wie viele Rechner/ Server gibt es in Ihrem Active Directory?
- Wie viele dieser Rechner haben sich schon seit mehr als 90 Tagen nicht mehr am Netzwerk angemeldet?
- Wie viel **Mitglieder** hat die Gruppe der Domänen-Administratoren?
- Welche Domänen Controller gibt es in Ihrem Active Directory?

1.15 Zum Schluss

```
if (Learn-PowerShell -le $Null)
 {
      Get-Fired -recurse -force
 }
 else
 {
      Get-Raise –Path \\YourWallet\account -force
 }
# ;-)
```