PowerShell Kurz-Referenz

Argumente

Die Argumente zu einer Funktion sind in der `$args` Variablen. Man kann auf diese Argumente in einer ähnlichen Schleife durchlaufen wie diese:

```
foreach ($i in $args) {$i}
```

Der Zugriff auf jedes einzelne Argument erfolgt über den Index, beginnend bei 0.

```
$args[0]
```

Das letzte Element wir über den Index -1 zugegriffen.

```
$args[-1]
```

Bunter Text

Um eine Textausgabe bunt zu gestalten, kann man die Vordergrundfarbe im Kommando `Write-Host` ändern.

```
Write-Host "test" -foregroundcolor "green"
Write-Host "test" -backgroundcolor "red"
```

Zeilenumbruch

Das Zeichen `` `n `` bewirkt einen Zeilenumbruch.

```
Write-Host "Zeile 1.`nZeile 2."
```

Inverse Schrift

Das Kommando `Write-Warning` schreibt eine Nachricht invers.

```
Write-Warning "Es ist ein Fehler aufgetreten."
```

Kommentare

Eine Raute am Beginn eines Textes ist ein Kommentar.

```
# Das ist ein Kommentar und wird nicht ausgeführt.
```

Texteingabe von der Konsole

Für das Einlesen von Benutzereingaben wird `Read-Host` verwendet.

```
$a = Read-Host "Bitte einen Namen eingeben"
```

Umfalten einer Skriptzeile

Mit dem ‚Backtick' kann man eine Zeile in der nächsten Zeile fortsetzen.

```
Write-Host `
    "Das ist eine Fortsetzung der Zeile"
```

Auch beim Verkettungsoperator kann eine Zeile umgebrochen werden (sofern das Kommando eine Verkettung verwendet).

```
Get-ChildItem C:\Scripts |
    Sort-Object Length -Descending
```

Mehrere Kommandos in einer Zeile

Mehrere Kommandos in einer Zeile trennt man mit einem Strichpunkt.

```
$a = 1,2,3,4,5; $b = $a[2]; Write-Host $b
```

Vergleiche

Kommandos, die Vergleiche enthalten (z.B. `Where-Object`) benutzen besondere Vergleichoperatoren, die mit einem Bindestrich eingeleitet werden.

Ein c unmittelbar nach dem Bindestrich macht den Vergleich von der Schreibweise (groß-klein) abhängig. Beispielsweise ist -ceq die groß-klein-Variante von -eq.

-lt kleiner als (less than)

-le kleiner oder gleich (less than or equal to)

-gt größer als (greater than)

-ge größer oder gleich (greater than or equal to)

-eq gleich (equal to)

-ne ungleich (not equal to)

-like wie (like), erlaubt Wildcards

-notlike nicht wie (not like), erlaubt Wildcards

Lesen einer Textdatei

Das Kommando `Get-Content liest` den Text aus einer Datei in eine Variable.

```
$a = Get-Content C:\Scripts\Test.txt
```

Jede Zeile ist ein Element im Array $a. Jede Zeile kann über einen Index erreicht werden. Erste Zeile im Array $a:

```
$a[0]
```

Letzte Zeile im Array $a

```
$a[-1]
```

Die Anzahl der Zeilen, Wörter und Zeichen in einer Textdatei bestimmen.

```
Get-Content c:\scripts\test.txt |
Measure-Object -line -word -character
```

Schreiben einer Textdatei

Um eine Variable in eine Textdatei zu schreiben, benutzt man das `Out-File`–Kommando.

```
Get-Process | Out-File C:\Scripts\Test.txt
```

Um einen Text an eine bestehende Datei anzuhängen, muss der Parameter *–append* verwendet werden.

```
Get-Process | Out-File C:\Test.txt –append
```

Man kann auch die aus DOS bekannten Parameter zur Umlenkung verwenden (> schreiben, >> anhängen).

```
Get-Process > C:\Scripts\Test.txt
```

Eine andere Option erlaubt die Speicherung als CSV-Datei (Comma Separated Value).

```
Get-Process | Export-CSV C:\Test.csv
```

Ausdruck

Die Ausgabe an den Drucker erfolgt mit dem Kommando `Out-Printer`:

```
Get-Process | Out-Printer
```

Bedingte Anweisungen

Eine `if` -Anweisung schaut etwa so aus:

```
$a = "weiß"
if ($a -eq "red")
    {" Die Farbe ist rot."}
elseif ($a -eq "weiß")
    {" Die Farbe ist weiß."}
else
    {" Die Farbe ist blau."}
```

Mit `switch` vereinfacht sich eine Mehrfachentscheidung:

```
$a = 2
switch ($a)
    {
        1 {"Die Farbe ist rot."}
        2 {"Die Farbe ist blau."}
        3 {"Die Farbe ist grün."}
        4 {"Die Farbe ist gelb."}
        default {"Es ist eine andere Farbe."}
    }
```

For- und For Each-Schleifen

Um Vorgänge zu wiederholen, benutzt man `for` oder `foreach`-Schleifen.

```
for ($a = 1; $a -le 10; $a++) {$a}
foreach ($i in get-childitem c:\scripts) {$i.extension}
```

Do-Loop-Schleifen

Schleifen mit Abfrage am Ende:

```
$a = 1
do {$a; $a++}
while ($a -lt 10)

$a = 1
do {$a; $a++}
until ($a –gt 10)
```

COM-Objekt erzeugen

Die folgenden Zeilen öffnen Excel und schalten es ein.

```
$a = New-Object -comobject `
    "Excel.Application"
$a.Visible = $True
```

.NET-Objekt erzeugen

Eine Instanz des DotNet-Objekts `system.Net.DNS` erzeugen und die gewünschte Methode `resolve` aufrufen.

```
[system.Net.DNS]::resolve("207.46.198.30")
```

Eine Objekt-Referenz eines DotNet-Framework-Objekts erzeugen.

```
$a = new-object `
-type system.diagnostics.eventlog `
-argumentlist system
```

Eigenschaften auswählen

Um mit bestimmten Eigenschaften einer Sammlung zu arbeiten oder diese anzuzeigen, leitet man das Objekt an das Kommando `Select-Object` weiter:

```
Get-Process | Select-Object Name, Company
```

Daten sortieren

Sortieren nach der Eigenschaft ID:

```
Get-Process | Sort-Object ID
```

Änderung der Sortierrichtung

```
Get-Process | Sort-Object ID -descending
```

Nach mehreren Eigenschaften sortieren

```
Get-Process | Sort-Object ProcessName, ID
```

WMI

Ein WMI-Objekt öffnen, um Informationen über den Computer zu bekommen

```
Get-WMIObject Win32_BIOS
```

Wenn die gewünschte Klasse nicht im Namensraum cimv2 ist, muss der Parameter *-namespace* verwendet werden:

```
Get-WMIObject SystemRestore `
    -namespace root\default
```

Wenn es sich um einen anderen Computer handelt, muss der Parameter *-computername* verwendet werden:

```
Get-WMIObject Win32_BIOS `
    -computername atl-ws-01
```

Mit dem Parameter *-query* kann die Ausgabe eingeschränkt werden.

```
Get-WMIObject -query `
    "Select * From Win32_Service `
        Where State = 'Stopped'"
```

Active Directory

Um das Programm an einen Active Directory Account zu binden, muss der LDAP-Provder benutzt werden.

```
$a = [adsi] "LDAP://cn=kenmyer, `
    ou=Finance, dc=fabrikam, dc=com"
```

Alle Objekte in einer OU aufzulisten, ist etwas komplizierter. Man bindet das Programm an die OU und benutzt dann die Methode **PSBase_GetChildren()**.

```
$objOU = [ADSI]`
"LDAP://ou=Finance,dc=fabrikam,dc=com"
```

```
$users = $objOU.PSBase.Get_Children()
$users | Select-Object displayName
```

Lokale User

Um das Programm an einen lokalen User zu binden, benutzt man den WinNT provider:

```
$a = [adsi] "WinNT://atl-ws-01/kenmyer"
$a.FullName
```

Hilfe

Für jedes Kommando kann man mit `Get-Help` und dem Parameter `-full` eine ausführliche Information erhalten.

```
Get-Help Get-Process -full
```

Anzeige von Beispielen

```
Get-Help Get-Process -examples
```

Alle Kommandos erfährt man mit `Get-Command`.

```
Get-Command
```

Alle Alias-Namen erfährt man mit dem Kommendo `Get-Alias`:

```
Get-Alias
```

Sicherheits-Einstellungen

Um Skripts aus der PowerShell ausführen zu können, müssen die Sicherheitseinstellungen geändert werden, weil PowerShell in der Anfangseinstellung nur signierte Skripts ausführt. Um auch lokal verfasste Skripts ausführen zu können, die signiert oder auch unsigniert sein können, benutzt man folgendes Kommando:

```
Set-ExecutionPolicy RemoteSigned
```

Objekt untersuchen

Um Informationen über ein Objekt zu erhalten, legt man zuerst eine Instanz des Objekts an und leitet das Objekt an das Kommando **Get-Member** weiter. Beispielsweise erhält man mit dem folgenden Kommando alle Eigenschaften und Methoden der aktuellen Arbeitsumgebung.

```
Get-Process | Get-Member
```

Konsolenfenster löschen

Um den Inhalt des Kommandofensters zu löschen, benutzt man das Kommando **Clear-Host** oder dessen Alias **cls**.

Copy & Paste

Mit Windows-X-> PowerShell öffnen.

Das Eigenschaftsfenster durch einen Mausklick auf das Symbol links oben öffnen.

In der Dialog-Box **Options** den **QuickEdit Mode** auswählen und auf OK klicken.

Um einen Text zu kopieren, diesen Im Konsolenfenster markieren und auf Enter klicken. Um einen Text in der Zwischenablage einzufügen, auf die rechte Maustaste klicken.

Skript ausführen

Um ein Skript auszuführen gibt man den vollen Pfad ein:

```
C:\Scripts\Test.ps1
```

Wenn der Pfad Spaces enthält, muss die Zeile mit Anführungszeichen eingeschlossen werden.

```
&"C:\Scripts\My Scripts\test.ps1"
```

Von außerhalb der Windows PowerSell, etwas aus einer DisolgBox cmd.exe, ruft man PowerShell auf und übergibt den Pfad als Parameter:

```
powershell.exe –noexit C:\Scripts\Test.ps1
```

Mit dem Parameter **-noexit** wird sichergestellt, dass dasa Fenster auch nach Ausführung desSkripts offen bleibt.

Mehr Informationen

http://technet.microsoft.com/en-us/scriptcenter/dd742419.aspx.
http://technet.microsoft.com/de-de/library/cc196356.aspx