

Powershell

Schleifen und Verzweigungen

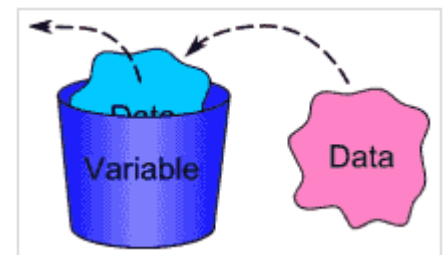
Inhaltsverzeichnis

1	Variablen.....	1
2	Verzweigungen (if-cmdlet).....	3
3	Schleifen	4
4	Das Where-Objekt.....	6
5	Übungen	6

1 Variablen

[piet]

Der Zweck von Variablen besteht bekanntlich darin, Werte zu speichern, um später im Code darauf zugreifen zu können. Auch hier bietet PowerShell weit mehr Möglichkeiten als der veraltete Batch-Interpreter und kann sich mit anderen modernen Script-Sprachen messen.



Für die Vergabe von Variablennamen folgt PowerShell einer ähnlichen Konvention wie PHP oder Perl, indem es ihnen das Dollarzeichen '\$' voranstellt. Darüber hinaus dürfen die Namen nur alphanumerische Zeichen oder den Unterstrich '_' enthalten. Will man unbedingt noch weitere Zeichen verwenden, dann muss man den Namen in geschweifte Klammern setzen.

Hier einige Beispiele für gültige Namen von Variablen:

- \$myVariable
- \$MyVariable_1
- \${my-variable}

Ungültig sind dagegen:

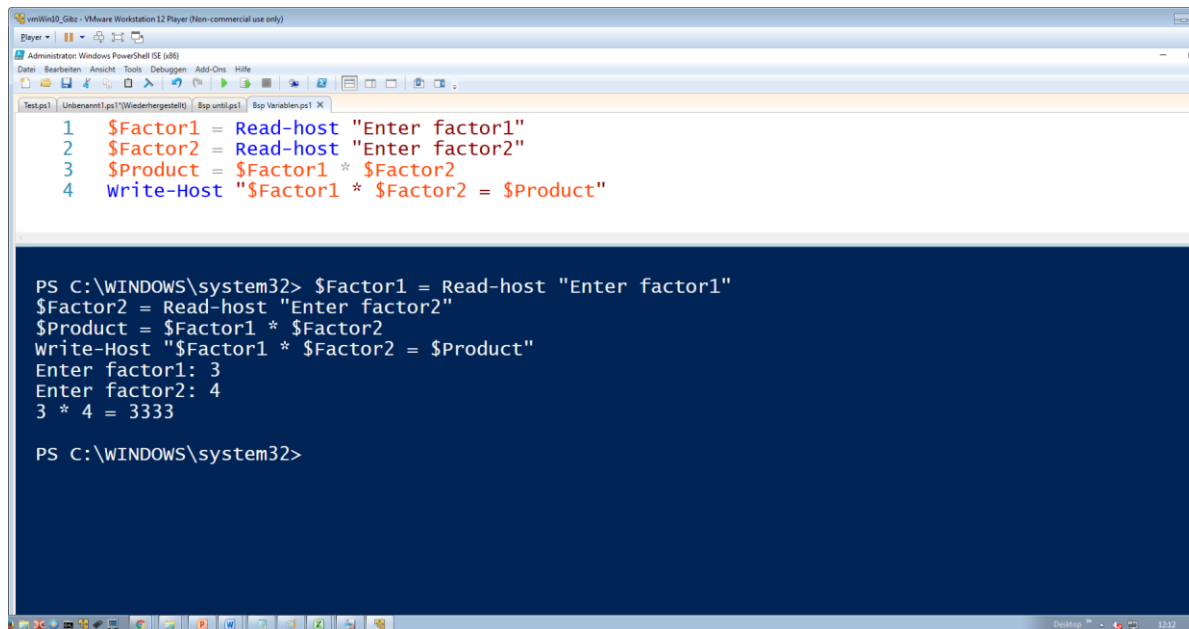
- myVariable
- \$my-variable
- \$my variable

[piet]

Datentypen

Der Datentyp muss in Powershell nicht explizit angegeben werden. Der Interpreter wird den Datentyp wählen, welcher ihm plausibel erscheint.

Das dies auch schiefgehen kann zeigt folgendes Beispiel:

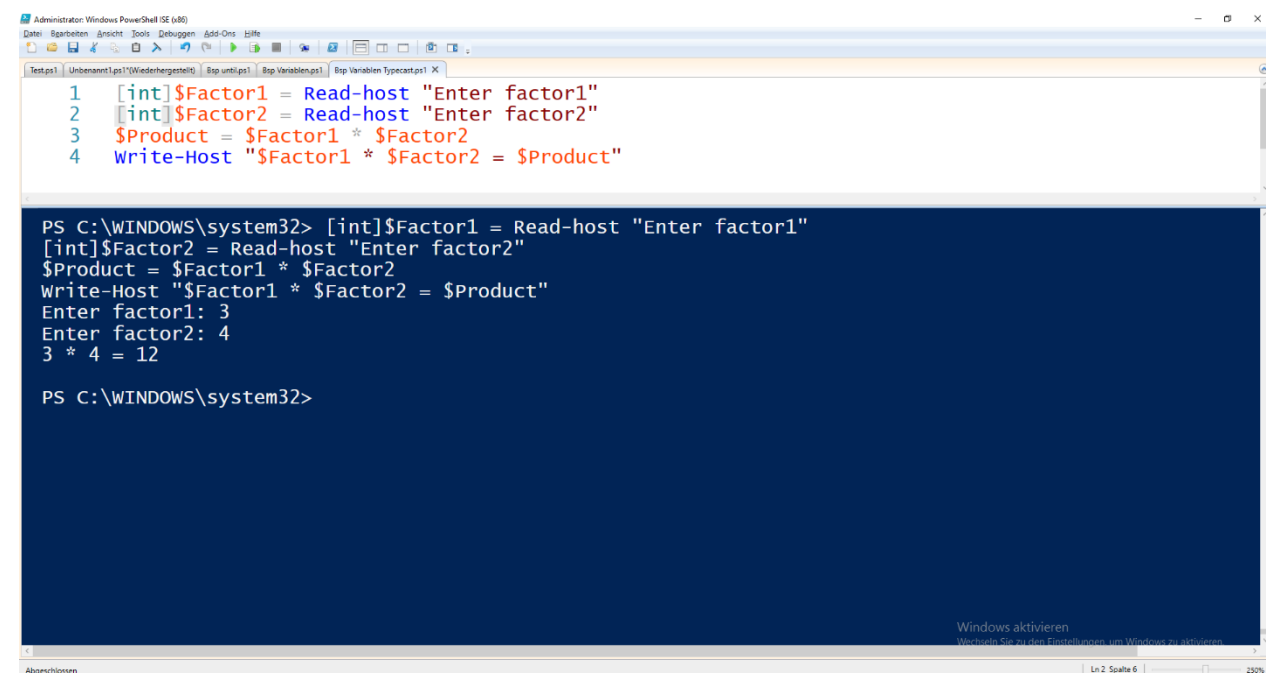


```
1 $Factor1 = Read-host "Enter factor1"
2 $Factor2 = Read-host "Enter factor2"
3 $Product = $Factor1 * $Factor2
4 Write-Host "$Factor1 * $Factor2 = $Product"

PS C:\WINDOWS\system32> $Factor1 = Read-host "Enter factor1"
$Factor2 = Read-host "Enter factor2"
$Product = $Factor1 * $Factor2
Write-Host "$Factor1 * $Factor2 = $Product"
Enter factor1: 3
Enter factor2: 4
3 * 4 = 3333

PS C:\WINDOWS\system32>
```

Abhilfe schafft dabei die Typenwandlung:



```
1 [int]$Factor1 = Read-host "Enter factor1"
2 [int]$Factor2 = Read-host "Enter factor2"
3 $Product = $Factor1 * $Factor2
4 Write-Host "$Factor1 * $Factor2 = $Product"

PS C:\WINDOWS\system32> [int]$Factor1 = Read-host "Enter factor1"
[int]$Factor2 = Read-host "Enter factor2"
$Product = $Factor1 * $Factor2
Write-Host "$Factor1 * $Factor2 = $Product"
Enter factor1: 3
Enter factor2: 4
3 * 4 = 12

PS C:\WINDOWS\system32>
```

2 Verzweigungen (if-cmdlet)

Das if-cmdlet in Powershell muss folgende Bedingungen erfüllen:

- Die Bedingung muss in runden Klammern stehen ()
- Der „if-Block“ und der „else-Block“ müssen in geschweiften Klammern stehen {}

Der grösste Unterschied zu einem Ihnen bekannten if-Befehl aus c++ oder C# liegt in der Formulierung der Bedingung. Anstelle der Zeichen <, >, = und/oder ! werden Abkürzungen verwendet. Es gelten dabei folgende Vergleichsoperatoren:

Powershell-Syntax	Bedeutung
-eq	Gleich
-ne	-Nicht gleich
-gt oder -ge	Grösser als oder grösser gleich
-lt oder -le	Kleiner als oder kleiner gleich
-match	Entspricht
-notmatch	Entspricht nicht

In Powershell existiert zudem auch die Mehrfachverzweigung:

```
1 [int]$Number1 = Read-Host "Enter number 1"
2 [int]$Number2 = Read-Host "Enter number 2"
3 $Calc = Read-Host "Enter the calculation (A)ddition, (S)ubstraction, (M)ultiplication, (D)ivision : "
4
5 Switch($Calc){
6   A {$Result = $Number1 + $Number2}
7   S {$Result = $Number1 - $Number2}
8   M {$Result = $Number1 * $Number2}
9   D {$Result = $Number1 / $Number2}
10  default {"$Calc is not a valid operation"}
11 }
12 write-host $Result
13 |
```

```
PS C:\WINDOWS\system32> C:\Powershell\Bsp_switch.ps1
Enter number 1: 95
Enter number 2: 87
Enter the calculation (A)ddition, (S)ubstraction, (M)ultiplication, (D)ivision : S
8
```

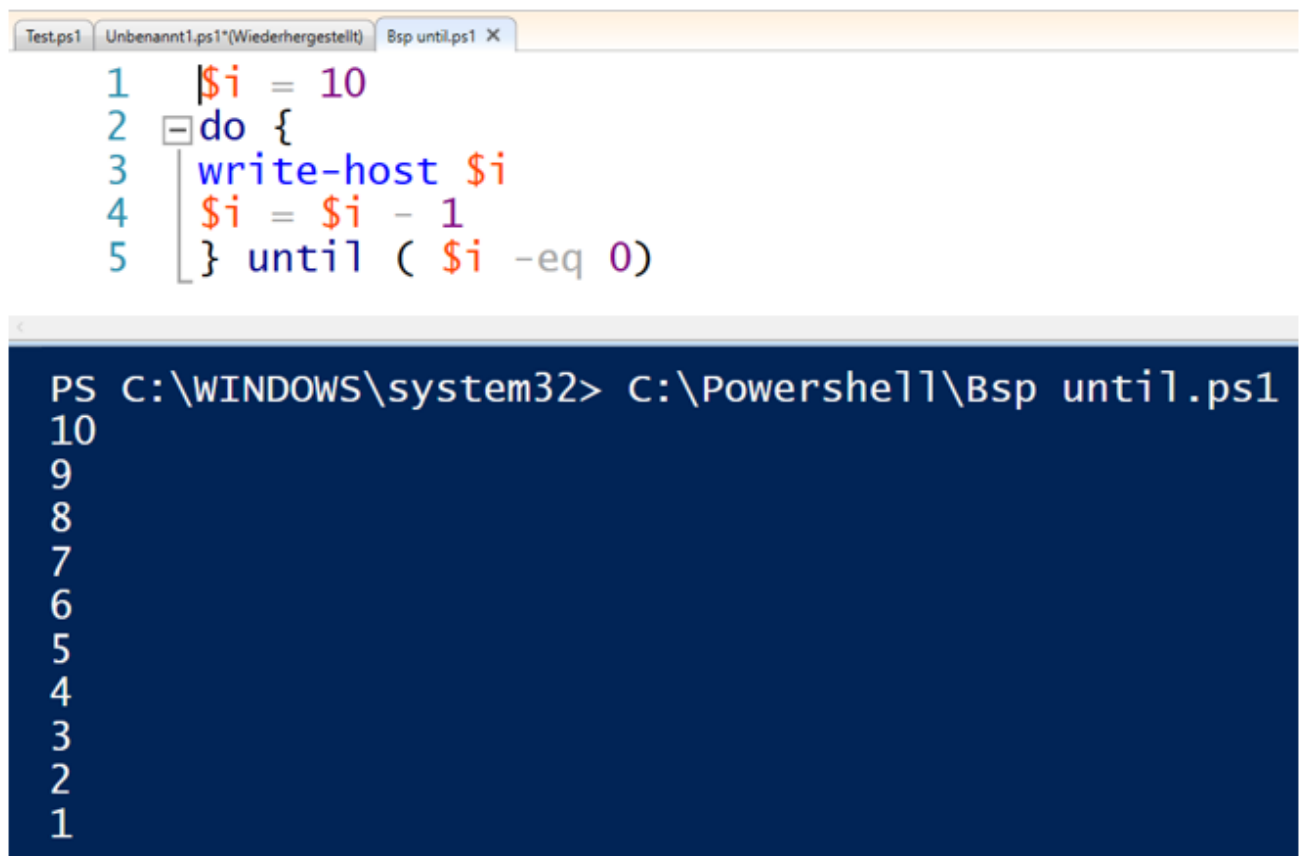
3 Schleifen

Zuerst die Ihnen bekannten Schleifen:

```
PS C:\WINDOWS\system32> for ($q=1; $q -le 10; $q++) {$q}
1
2
3
4
5
6
7
8
9
10
```

```
Test.ps1  Unbenannt1.ps1*(Wiederhergestellt)  Bsp While.ps1 X
1      $i = 1
2  while ($i -le 10){
3      write-host $i
4      $i = $i+1
5  }
```

```
PS C:\WINDOWS\system32> C:\Powershell\Bsp while.ps1
1
2
3
4
5
6
7
8
9
10
```



The screenshot shows a PowerShell script editor with three tabs: 'Test.ps1', 'Unbenannt1.ps1*(Wiederhergestellt)', and 'Bsp until.ps1'. The script in 'Bsp until.ps1' is as follows:

```
1 $i = 10
2 do {
3     write-host $i
4     $i = $i - 1
5 } until ( $i -eq 0)
```

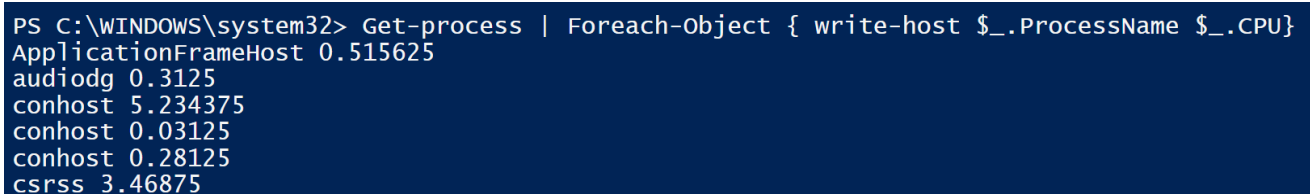
Below the editor, a terminal window shows the execution of the script. The prompt is 'PS C:\WINDOWS\system32> C:\Powershell\Bsp until.ps1'. The output is a vertical list of numbers from 10 down to 1:

```
10
9
8
7
6
5
4
3
2
1
```

Die wohl mit Abstand am häufigsten verwendete Schleife ist die `foreach`-Schleife. Diese hat kein Pendant in `c++` oder `c#`. Am häufigsten verwendet wird sie, weil sie automatisch sämtliche Elemente einer Liste durcharbeitet. Genau das, was beim Scripten mit Powershell häufig verwendet wird: Mit einem `cmdlet` erhalten Sie eine Liste von Objekten, wenn Sie jedes einzelne davon bearbeiten möchten und sich nicht darum kümmern müssen, wieviele es sind, eignet sich das `foreach` Statement am Besten:

```
Get-process | Foreach-Object { write-host $_.ProcessName $_.CPU}
```

„`$_`.“ steht dabei für das aktuelle Listenelement. Es kann dabei auf alle Eigenschaften und Methoden des Elementes zugegriffen werden.



The screenshot shows the execution of the command `Get-process | Foreach-Object { write-host $_.ProcessName $_.CPU}` in a PowerShell terminal. The output lists the process names and their CPU usage percentages:

```
PS C:\WINDOWS\system32> Get-process | Foreach-Object { write-host $_.ProcessName $_.CPU}
ApplicationFrameHost 0.515625
audiodg 0.3125
conhost 5.234375
conhost 0.03125
conhost 0.28125
csrss 3.46875
```

4 Das Where-Objekt

Das where-Objekt gehört zwar nicht zu den Steuerelementen wie die Verzweigung und/oder die Schleifen. Trotzdem soll es an dieser Stelle erwähnt sein, da es zum Filtern von Daten geeignet ist.

```
PS C:\WINDOWS\system32> get-help where-Object

NAME
    where-Object

ÜBERSICHT
    selects objects from a collection based on their property values.
```

Beispiel: Alle gestoppten Services anzeigen:

```
Get-service | where-object {$_.status -eq „stopped“}
```

Für das where Objekt gibt es eine vereinfachte Schreibweise:

```
Get-Service | where Status -eq „Stopped“.
```

Diese Schreibweise ist aber nur für einfache Bedingungen zulässig, sobald mehrere mit einem logischen Operator verknüpft werden, produziert PowerShell eine Fehlermeldung.

5 Übungen

13. Sortieren Sie die Services absteigend nach ihrem Namen. Wählen Sie von der erhaltenen Liste den 3. 4. und 5. Eintrag aus und speichern Sie ihn in einer Variablen (z.B. \$ServiceList).
14. Erstellen Sie ein Skript das die Zahlen n..m addiert. Erweitern Sie ihr Script, so dass m und n eingelesen werden können.
15. Erstellen Sie eine Liste aller Services.
 - a) Geben Sie nur die Eigenschaften Status und Namen aus. Verwenden Sie dazu das foreach-Statement!
 - b) Färben Sie abhängig vom Status die Zeile Rot oder Grün ein. Erweitern Sie dazu das Statement aus a)
16. Führen Sie alle Dienste auf, die den Starttyp „Automatisch“ haben. Verwenden Sie hierzu das Where-Objekt!

In diesem Dossier verwendete Quellen

[piet] Michael Pietrofte, <https://www.windowspro.de/script/variablen-powershell-namen-werte-datentypen>

Historie

Dokument erstellt	R. Müller	07.09.2017
Aufgabe 16 präzisiert	R. Müller	18.09.2017