PowerShell Schleifen – Zusatz

Grundsätzliche Schreibweise

PowerShell orientiert sich bei Schleifen an C-ähnlichen Sprachen.

Die Schleifenbedingung steht meistens in runden Klammern.

```
while ($x -lt 5)
```

Der Schleifenkörper steht in geschweiften Klammern.

```
{
   Write-Host $x
}
```

PowerShell-Befehlswörter sind nicht case-sensitive.

Das ist also gleichbedeutend:

```
for
For
FOR
```

Abweisende Schleifen

Abweisende Schleifen prüfen die Schleifenbedingung vor dem ersten Durchlauf.

Wenn die Bedingung direkt am Anfang falsch ist, wird der Schleifenkörper kein einziges Mal ausgeführt.

Beispiele:

```
while ($x -lt 5) {
}

for ($i = 1; $i -le 10; $i++) {
}
```

Nicht-abweisende Schleifen

Nicht-abweisende Schleifen prüfen die Schleifenbedingung am Ende des Schleifenkörpers.

Der Schleifenkörper wird deshalb mindestens einmal ausgeführt.

Beispiele:

```
do {
   Write-Host $x
}
while ($x -lt 5)

do {
   Write-Host $x
}
until ($x -ge 5)
```

---

for-Schleife

Die for-Schleife eignet sich, wenn man eine Laufvariable direkt in der Schleifenanweisung initialisieren und inkrementieren möchte.

Aufbau:

```
for (Initialisierung; Schleifenbedingung; Inkrementierung) {
   Anweisungen
}
```

Beispiel:

```
for ($i = 1; $i -le 10; $i++) {
   Write-Host $i
}
```

Bedeutung:

```
$i wird mit 1 initialisiert.
Die Schleife läuft, solange $i kleiner oder gleich 10 ist.
```

```
Nach jeder Iteration wird $i inkrementiert.
```

Laufvariable

Eine Laufvariable steuert, wie oft eine Schleife läuft.

Beispiel:

```
$i
```

In einer for-Schleife kann die Laufvariable direkt in der Schleifenanweisung gesetzt und verändert werden.

```
for ($i = 1; $i -le 10; $i++) {
}
```

Initialisieren

Initialisieren bedeutet: Eine Variable bekommt einen Startwert.

Beispiel:

```
$i = 1
```

Inkrementieren

Inkrementieren bedeutet: Eine Variable wird erhöht.

Beispiel:

```
$i++
```

Das ist gleichbedeutend mit:

```
$i += 1
```

---

while-Schleife

Die while-Schleife ist eine abweisende Schleife.

Sie prüft die Schleifenbedingung vor dem ersten Durchlauf.

Beispiel:

```
while ($i -le 10) {
   Write-Host $i
   $i++
}
```

Bei while besteht der Ausdruck nur aus einer Bedingung.

Wenn man eine Laufvariable verwenden möchte, muss man sie vorher initialisieren und im Schleifenkörper verändern.

Beispiel:

```
$i = 1

while ($i -le 10) {
   Write-Host $i
   $i++
}
```

---

while mit Eingabe und Zuweisung

In der Schleifenbedingung einer while-Schleife darf man auch Werte zuweisen oder Eingaben abfragen.

Beispiel:

```
while (($inp = Read-Host "Wählen Sie einen Befehl") -ne "Q") {
   Write-Host "Eingabe war $inp"
}
```

Bedeutung:

```
Read-Host fragt eine Eingabe ab.
Der Wert wird in $inp gespeichert.
```

```
Solange $inp nicht Q ist, läuft die Schleife weiter.
```

---

do-while

do-while ist eine nicht-abweisende Schleife.

Der Schleifenkörper läuft mindestens einmal.

Beispiel:

```
do {
    Write-Host $x
}
while ($x -lt 5)
```

Bedeutung:

```
Die Schleife läuft, solange die Bedingung wahr ist.
Sie bricht ab, wenn die Bedingung nicht mehr erfüllt ist.
```

---

do-until

do-until ist ebenfalls eine nicht-abweisende Schleife.

Der Schleifenkörper läuft mindestens einmal.

Beispiel:

```
do {
    Write-Host $x
}
until ($x -ge 5)
```

Bedeutung:

```
Die Schleife läuft, bis die Bedingung wahr wird.
Sie endet, wenn die Schleifenbedingung TRUE wird.
```

---

do-while und do-until gegenseitig ersetzen

do-while und do-until können sich grundsätzlich gegenseitig ersetzen, wenn man die Bedingung negiert.

Beispiel mit do-while:

```
do {
   $i++
}
while ($i -lt 5)
```

Gleiche Logik mit do-until:

```
do {
   $i++
}
until ($i -ge 5)
```

Merke:

```
do-while läuft, solange die Bedingung wahr ist.
do-until läuft, bis die Bedingung wahr wird.
```

---

break

break dient zur Schleifensteuerung.

break verlässt den aktuellen Block.

In Schleifen beendet break die Iteration und setzt den Programmablauf mit der nächsten Anweisung außerhalb des Schleifenkörpers fort.

Beispiel:

```
while ($true) {
   if ($fertig) {
      break
   }
}
```

Bedeutung:

```
Die Schleife läuft eigentlich dauerhaft.
Sobald $fertig wahr ist, beendet break die Schleife.
```

---

break in switch-Blöcken

break kann nicht nur in Schleifen verwendet werden, sondern auch innerhalb von switch-Blöcken.

Beispiel:

```
switch ($inp) {
    "L" { "Datei wird gelöscht"; break }
    "A" { "Datei wird angezeigt"; break }
    "R" { "Datei erhält Schreibschutz"; break }
    "Q" { "Ende"; break }
    default { "Ungültige Eingabe" }
}
```

Bedeutung:

```
break verlässt den switch-Block.
```

---

break bei verschachtelten Schleifen

Bei verschachtelten Schleifen verlässt break den aktuellen Schleifenblock.

Beispiel:

```
foreach ($a in 1..3) {
    foreach ($b in 1..3) {
        break
    }
}
```

Bedeutung:

```
break beendet hier die innere Schleife.
Die äußere Schleife kann weiterlaufen.
```

---

break mit Sprungmarke

PowerShell erlaubt break auch mit einer Sprungmarke.

Eine Sprungmarke beginnt mit einem Doppelpunkt.

Beispiel-Schema:

```
:MeineSchleife while ($true) {
    break MeineSchleife
}
```

Wichtig:

```
break mit Sprungmarke ist möglich.
Es sollte aber sparsam verwendet werden, weil es Code schwerer verständlich machen kann.
```

---

continue

continue dient ebenfalls zur Schleifensteuerung.

continue bricht die weitere Ausführung des aktuellen Schleifenkörpers ab.

Im Unterschied zu break verlässt continue die Schleife nicht komplett, sondern startet die nächste Iteration.

Beispiel:

```
foreach ($zahl in 1..5) {
    if ($zahl -eq 3) {
        continue
    }

    Write-Host $zahl
}
```

Ausgabe:

```
1
2
4
5
```

Bedeutung:

```
Bei 3 wird der restliche Schleifenkörper übersprungen.
Danach geht es mit der nächsten Iteration weiter.
```

Iteration

Eine Iteration ist ein einzelner Schleifendurchlauf.

Wenn eine Schleife 5-mal läuft, hat sie 5 Iterationen.

Implizite Iteration über Array-Elemente

PowerShell kann bei Arrays in bestimmten Fällen automatisch über die Elemente iterieren.

Beispiel:

```
($user).Surname
```

Wenn $user mehrere Objekte enthält, wird die Eigenschaft Surname für jedes Element ausgegeben.

Bedeutung:

```
Man braucht nicht immer eine explizite Schleife.
PowerShell kann Eigenschaften von Array-Elementen teilweise automatisch ausgeben.
```

---

foreach-Anweisung

Die foreach-Anweisung wird verwendet, um über Elemente eines Arrays oder einer Sammlung zu iterieren.

Beispiel:

```
$user = Get-ADUser -Filter *

foreach ($u in $user) {
    $u.Surname
}
```

Bedeutung:

```
$user enthält mehrere Objekte.
$u enthält immer das aktuelle Element des Arrays.
Der Schleifenkörper wird für jedes Element einmal ausgeführt.
```

ForEach-Object

ForEach-Object ist ein Cmdlet für die Verarbeitung von Pipeline-Objekten.

Beispiel:

```
Get-ADUser -Filter * | ForEach-Object {
    $_.Surname
}
```

Bedeutung:

```
Die Objekte kommen über die Pipeline.
ForEach-Object verarbeitet jedes Objekt einzeln.
```

---

foreach als Alias für ForEach-Object

Wichtig: foreach kann zwei Bedeutungen haben.

Als Schleifenanweisung:

```
foreach ($u in $user) {
    $u.Surname
}
```

Als Alias für ForEach-Object in der Pipeline:

```
Get-ADUser -Filter * | foreach {
   $_.Surname
}
```

Merke:

```
foreach (...) { }     = foreach-Anweisung
| foreach { }         = Alias für ForEach-Object
```

% als Alias für ForEach-Object

% ist ein weiterer Alias für ForEach-Object.

Diese Schreibweisen sind vergleichbar:

```
Get-ADUser -Filter * | ForEach-Object {
   $_.Surname
}

Get-ADUser -Filter * | foreach {
   $_.Surname
}

Get-ADUser -Filter * | % {
   $_.Surname
}
```

Merke:

```
% = Alias für ForEach-Object
```

$_

$_ ist eine automatische Variable.

Bei ForEach-Object steht $_ für das aktuelle Objekt aus der Pipeline.

Beispiel:

```
Get-ADUser -Filter * | ForEach-Object {
   $_.Surname
}
```

Bedeutung:

```
$_ = aktuelles Pipeline-Objekt
```

---

foreach-Anweisung vs. ForEach-Object

Die foreach-Anweisung arbeitet mit einer vorhandenen Sammlung.

```
foreach ($u in $user) {
   $u.Surname
}
```

ForEach-Object arbeitet typischerweise mit Pipeline-Ausgabe.

```
Get-ADUser -Filter * | ForEach-Object {
   $_.Surname
}
```

Merke:

```
foreach-Anweisung = gut für vorhandene Arrays oder Sammlungen
ForEach-Object    = gut für Pipeline-Verarbeitung
```

Performance: foreach-Anweisung und ForEach-Object

Bei vielen Schleifendurchläufen und komplexeren Berechnungen im Schleifenkörper ist die herkömmliche foreach-Schleife oft schneller.

Beispiel foreach-Anweisung:

```
$user = Get-ADUser -Filter *

foreach ($u in $user) {
```

```
    $u.Surname
}
```

Beispiel ForEach-Object:

```
Get-ADUser -Filter * | ForEach-Object {
    $_.Surname
}
```

Merke:

```
foreach-Anweisung = oft schneller bei vielen Durchläufen
ForEach-Object    = oft kürzer und praktisch in der Pipeline
```

Typische Auswahl der Schleife

Zweck	Passende Schleife
Laufvariable direkt initialisieren und inkrementieren	for-Schleife
Nur eine Bedingung prüfen	while-Schleife
Mindestens einmal ausführen und dann Bedingung prüfen	do-while / do-until
Elemente eines Arrays verarbeiten	foreach-Anweisung
Pipeline-Objekte verarbeiten	ForEach-Object
Schleife sofort verlassen	break
Aktuelle Iteration überspringen	continue

Wichtige Merksätze

```
Schleifenbedingung = Bedingung, die den Ablauf der Schleife steuert.

Schleifenkörper = Anweisungen, die wiederholt ausgeführt werden.

Abweisende Schleifen können 0-mal laufen.

Nicht-abweisende Schleifen laufen mindestens 1-mal.

for-Schleife eignet sich besonders mit Laufvariable.
```

while-Schleife eignet sich, wenn nur eine Bedingung geprüft werden soll.

do-while läuft, solange die Bedingung wahr ist.

do-until läuft, bis die Bedingung wahr wird.

break verlässt den aktuellen Block.

continue startet die nächste Iteration.

foreach-Anweisung ist nicht dasselbe wie ForEach-Object.

foreach und % können Aliase für ForEach-Object sein.

$_ steht für das aktuelle Pipeline-Objekt.