Powershell Trainer

- Trainer 01 – Basics bis Schleifen
- Trainer 02 – Grundlagen
- Trainer 03 – Bedingungen und Schleifen
- Trainer 04 - Schleifen
- Trainer 05 - Schleifenlogik verstehen
- Trainer 06 – Do-While und Until
- PowerShell – Schleifen kompakt erklärt
- PowerShell Schleifen – Zusatz
- PowerShell Trainer – Benannte Parameter und Parameter-Dekorator
- PowerShell Trainer – Fragen und Antworten zu Parametern

Trainer 01 – Basics bis Schleifen

PowerShell-Trainer im Vollbild öffnen

https://trainer.ulrich-wiki.com/powershell-trainer-01-basics-loops.html?v=1

Trainer 02 – Grundlagen

PowerShell-Trainer 02 im Vollbild öffnen

https://trainer.ulrich-wiki.com/powershell-trainer-02-grundlagen.html?v=1

Trainer 03 – Bedingungen und Schleifen

PowerShell-Trainer 03 im Vollbild öffnen

https://trainer.ulrich-wiki.com/powershell-trainer-03-bedingungen-schleifen.html?v=1

Trainer 04 - Schleifen

Trainer im Vollbild öffnen

https://trainer.ulrich-wiki.com/powershell-schleifen-trainer-01.html?v=5

Trainer 05 - Schleifenlogik verstehen

Trainer im Vollbild öffnen

https://trainer.ulrich-wiki.com/powershell-trainer-05-schleifenlogik-verstehen.html?v=2

Trainer 06 – Do-While und Until

Vollbild öffnen

https://trainer.ulrich-wiki.com/powershell-trainer-06-do-while-until.html?v=2

PowerShell – Schleifen kompakt erklärt

Grundidee

Eine Schleife wiederholt einen Codeblock, solange eine Bedingung erfüllt ist oder bis eine Abbruchbedingung erreicht wird.

Wichtige Schleifentypen

Schleife	Bedeutung	Prüft wann?	Kann 0-mal laufen?
while	solange eine Bedingung wahr ist	vorher	ja
for	Zählschleife mit Start, Bedingung und Änderung	vorher	ja
foreach	läuft durch alle Elemente einer Liste	vorher	ja, wenn Liste leer ist
do while	läuft mindestens einmal, dann solange Bedingung wahr ist	danach	nein
do until	läuft mindestens einmal, dann bis Bedingung wahr wird	danach	nein

---

Abweisende Schleifen

Abweisende Schleifen prüfen die Bedingung vor dem ersten Durchlauf.

Sie können deshalb 0-mal laufen.

Beispiele:

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

for ($i = 0; $i -lt 3; $i++) {
}

foreach ($item in $liste) {
}
```

Nicht-abweisende / fußgesteuerte Schleifen

Diese Schleifen prüfen die Bedingung erst nach dem ersten Durchlauf.

Sie laufen deshalb mindestens 1-mal.

Beispiele:

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

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

Kopfgesteuert und fußgesteuert

Begriff	Bedeutung	Beispiele
kopfgesteuert	Bedingung wird vor dem Schleifenkörper geprüft	while, for, foreach
fußgesteuert	Bedingung wird nach dem Schleifenkörper geprüft	do while, do until

---

Schleifenkopf

Im Schleifenkopf steht die Bedingung oder Steuerung der Schleife.

Beispiel:

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

Schleifenkörper

Der Schleifenkörper ist der Code, der wiederholt ausgeführt wird.

Beispiel:

```
{
   Write-Host $i
   $i++
}
```

Iteration

Eine Iteration ist ein einzelner Schleifendurchlauf.

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

Abbruchbedingung

Die Abbruchbedingung beschreibt, wann die Schleife beendet werden soll.

Beispiel:

```
Abbrechen, sobald $i mindestens 5 ist.
```

Bei do until steht diese Bedingung direkt in der Schleife:

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

Weiterlaufbedingung

Die Weiterlaufbedingung beschreibt, solange die Schleife weiterlaufen soll.

Beispiel:

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

Bedeutung:

```
Wiederholen, solange $i kleiner als 5 ist.
```

---

while

while prüft vor jedem Durchlauf.

```
$i = 0

while ($i -lt 3) {
   Write-Host $i
   $i++
}
```

Ausgabe:

```
0
1
2
```

Warum keine 3?

```
3 -lt 3 = falsch
```

---

do while

do while läuft mindestens einmal und prüft danach.

```
$x = 10

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

Ausgabe:

```
10
```

Die Schleife läuft nur 1-mal, weil danach geprüft wird:

```
10 -lt 5 = falsch
```

---

do until

do until läuft mindestens einmal und stoppt, sobald die Bedingung wahr ist.

```
$punkte = 0

do {
   $punkte += 10
}
until ($punkte -ge 100)
```

Bedeutung:

```
Wiederholen, bis $punkte mindestens 100 ist.
```

---

for

for ist eine typische Zählschleife.

Aufbau:

```
for (Startwert; Bedingung; Veränderung) {
}
```

Beispiel:

```
for ($i = 0; $i -lt 3; $i++) {
   Write-Host $i
}
```

Ausgabe:

```
0
1
2
```

Ablauf:

Durchlauf	$i	Prüfung
1	0	0 -lt 3 = wahr
2	1	1 -lt 3 = wahr
3	2	2 -lt 3 = wahr
Ende	3	3 -lt 3 = falsch

Leerer Schleifenkörper

Auch wenn der Schleifenkörper leer ist, läuft die Schleife trotzdem.

```
for ($i = 0; $i -lt 3; $i++) {
}
```

Diese Schleife läuft 3-mal.

Sie gibt nur nichts aus.

---

foreach

foreach wird verwendet, um jedes Element einer Liste zu verarbeiten.

```
foreach ($datei in $dateien) {
    Write-Host $datei
}
```

Bedeutung:

```
Für jede Datei in $dateien wird der Code einmal ausgeführt.
```

Wenn die Liste leer ist, läuft foreach 0-mal.

Vergleichsoperatoren in PowerShell

Operator	Bedeutung
-lt	kleiner als
-le	kleiner oder gleich

Operator	Bedeutung
-gt	größer als
-ge	größer oder gleich
-eq	gleich
-ne	ungleich

Beispiele:

```
$i -lt 5    # kleiner als 5
$i -le 5    # kleiner oder gleich 5
$i -gt 5    # größer als 5
$i -ge 5    # größer oder gleich 5
$i -eq 5    # genau 5
$i -ne 5    # nicht 5
```

Wichtig: -lt und -le

```
-lt 5 = kleiner als 5
```

Läuft bei:

```
0, 1, 2, 3, 4
```

Nicht bei:

```
5


-le 5 = kleiner oder gleich 5
```

Läuft bei:

```
0, 1, 2, 3, 4, 5
```

---

while vs until

Schleife	Bedeutung

while	läuft solange die Bedingung wahr ist
until	läuft bis die Bedingung wahr wird

Beispiel mit while:

```
do {
   $punkte += 10
}
while ($punkte -lt 100)
```

Bedeutung:

```
Wiederholen, solange $punkte kleiner als 100 ist.
```

Beispiel mit until:

```
do {
   $punkte += 10
}
until ($punkte -ge 100)
```

Bedeutung:

```
Wiederholen, bis $punkte mindestens 100 ist.
```

Bei 100 abbrechen

Wenn bei 100 Schluss sein soll, ist bei do until logisch:

```
until ($punkte -ge 100)
```

Oder bei do while:

```
while ($punkte -lt 100)
```

Nicht passend wäre:

```
while ($punkte -le 100)
```

Denn bei 100 ist diese Bedingung noch wahr und die Schleife läuft nochmal.

Zähler erhöhen

```
$i++
```

ist dasselbe wie:

```
$i += 1
```

Beide erhöhen den Wert um 1.

Zähler verringern

```
$i--
```

ist dasselbe wie:

```
$i -= 1
```

Beide verringern den Wert um 1.

Größere Schritte

```
$i += 5
```

erhöht um 5.

```
$i -= 5
```

verringert um 5.

Initialisierung

Initialisierung bedeutet: Startwert setzen.

Beispiel:

```
$i = 0
```

Inkrement

Inkrement bedeutet: Wert erhöhen.

Beispiel:

```
$i++
```

oder:

```
$i += 1
```

Dekrement

Dekrement bedeutet: Wert verringern.

Beispiel:

```
$i--
```

oder:

```
$i -= 1
```

Endlosschleife

Eine Endlosschleife endet nicht von selbst.

Beispiel:

```
while ($true) {
    Write-Host "läuft"
```

```
}
```

Oder:

```
$x = 1

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

Wenn $x nie verändert wird, bleibt die Bedingung immer wahr.

---

break

break beendet eine Schleife sofort.

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

---

continue

continue überspringt den Rest des aktuellen Durchlaufs und springt direkt zum nächsten Durchlauf.

```
for ($i = 1; $i -le 5; $i++) {
    if ($i -eq 3) {
        continue
    }

    Write-Host $i
}
```

Ausgabe:

```
1
2
4
```

```
5
```

Off-by-one-Fehler

Ein Off-by-one-Fehler passiert, wenn eine Schleife einmal zu viel oder einmal zu wenig läuft.

Typischer Unterschied:

```
$i -lt 5
```

läuft bis 4.

```
$i -le 5
```

läuft bis 5.

Merksätze

```
while = solange

until = bis

for = Zählschleife

foreach = für jedes Element

do while = mindestens einmal, dann solange

do until = mindestens einmal, dann bis

break = sofort raus

continue = nächster Durchlauf

-lt = kleiner als

-le = kleiner oder gleich
```

-gt = größer als

-ge = größer oder gleich

-eq = gleich

-ne = ungleich

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.

PowerShell Trainer – Benannte Parameter und Parameter-Dekorator

PowerShell Trainer – Benannte Parameter und Parameter-Dekorator

Interaktiver Trainer zu **param()**, **[Parameter()]**, **[Alias()]**, **[switch]**, **[ValidateSet()]**, Dot-Sourcing und Userprofile-Auswertung.

Trainer im Vollbild öffnen

https://trainer.ulrich-wiki.com/powershell-parameter-trainer/

PowerShell Trainer – Fragen und Antworten zu Parametern

PowerShell Trainer – Fragen und Antworten zu Parametern

Frage-Antwort-Trainer zu **PowerShell-Parametern**, **param()**, **[Parameter()]**, **Aliasen**, **Switch-Parametern**, **ValidateSet**, Dot-Sourcing und der Funktion **check-userprofile**.

Frage-Antwort-Trainer im Vollbild öffnen