Kapitel 5: PowerShell-Skripte

Abseits der interaktiven Ausführung kann die PS auch Skripte verarbeiten, die mit der Dateierweiterung ".ps1" gekennzeichnet werden. Die "1" steht für die Version der PS.

PS-Skripte weisen durchaus Ähnlichkeiten zu den traditionell verbreiteten Batch-Formaten (.cmd, .bat), imperative Strukturen sind üblich, wobei das Objekt-Pipelining und die fortgeschrittenen Programmfunktionen der PS die klassischen Windows-Shell-Skripte weit hinter sich lassen.

In Anlehnung an die Sicherheitskonzepte aus der UNIX-Welt sind *.ps1-Dateien mit vollständigem oder relativem Pfad anzugeben:

```
c:\skripte\demo1.ps1
.\demo1.ps1
```

PS-Skripte lassen sich auch aus beliebigen Ordnern heraus ohne Angabe des vollständigen Pfades starten, in so fern der fragliche Heimatordner (zum Beispiel "c:\ps") in die Pfadvariable des lokalen Computers aufgenommen wird. In diesem Fall gelten die aufgerufenen Skripte als "vertrauenswürdig", innerhalb der PS-Konsole kann dann sogar die Dateierweiterung weggelassen werden.

Um Skripte überhaupt ausführen zu können, muss zunächst die so genannte "Ausführungsrichtlinie" (engl. "execution policy") angepasst werden:

```
Get-ExecutionPolicy
```

a) _____
(Notieren Sie die Ausgabe!)

Der Vorgabewert (den Sie soeben ermittelt haben, siehe a) unterbindet jedwedes Ausführen von Skripts. "Set-ExecutionPolicy" ändert diese Beschränkung. Mit dem nachfolgenden Befehl können Sie alle möglichen Richtlinien ermitteln:

```
Get-Help Set-ExecutionPolicy -param *
```

b) _____

c) _____

d) _____
(Notieren Sie die drei fehlenden Richtlinien)

Will man auf einem Testsystem die Richtlinien skriptgesteuert ändern, so steht man vor einem "Henne/Ei"-Problem: um ein Skript auszuführen, muss zunächst die Richtlinie liberalisiert werden. Abhilfe schafft hierbei zum Beispiel ein kleines Batchskript, dass zunächst die Richtlinie (ohne Verwendung der PS) modifiziert:

```
REG ADD
HKLM\SOFTWARE\Microsoft\PowerShell\1\ShellIds\Microsoft.PowerShell
/v ExecutionPolicy /t REG_SZ /d Unrestricted /f
```

In Produktivumgebungen sollte man die Ausführungsrichtlinie mittels GPO anpassen:

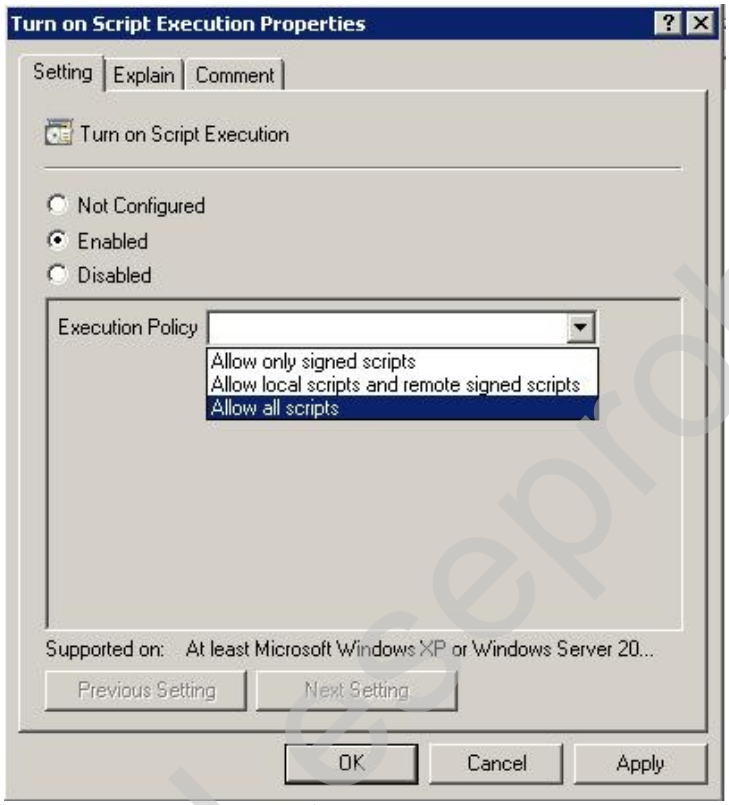
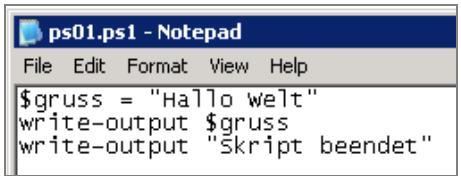


Abbildung 9: Script Execution Properties in GPO

"Allow all scripts" entspricht "Unrestricted".

"RemoteSigned" unterscheidet selbst erstellte Skripte von solchen, die aus dem Internet heruntergeladen wurden. Diese Unterscheidung wird möglich durch den so genannten "Zone Identifier", den zum Beispiel der IE beim Speichern eines Downloads erzeugt. Diese Meta-Information liegt als "Alternate Data Stream" (ADS) im NTFS-Dateisystem und ist dort weitgehend unsichtbar. "dir /r" macht sie sichtbar, "streams" aus der Sysinternals Suite erlaubt das suchen und löschen der ADS-Daten.

5.1 Ein erstes Skript



Erstellen Sie einen Ordner "c:\ps"; speichern Sie ein erstes Skript ab: Sie sehen oben ein Beispiel. Ihr Skript muss eine Variable enthalten. Variablen werden in der PS immer mit dem Dollarzeichen gebildet.

Rufen Sie Ihr Skript auf drei verschiedene Weisen auf:

```
a) c:\ps\ps01.ps1
b) cd c:\ps; .\ps01.ps1
c) . .\ps01.ps1
```

Testen Sie jeweils, ob nach Beendigung des Skripts die Variable $gruss noch einen Wert enthält. Es ist ausreichend, dass Sie einfach $gruss in der PowerShell eingeben.

In welchen Fällen bleibt der Wert der Variablen erhalten?

_____

Schließen Sie die PS. Starten Sie die PS erneut. Enthält die Variable noch einen Wert?

_____

Interaktion mit dem Benutzer:

```
$eingabe = Read-Host "Ihre Eingabe bitte"
Remove-Item $eingabe -whatif -errorAction "SilentlyContinue"
if ($?) { Write-Host "Datei existiert!" }
   else { Write-Warning "Datei existiert nicht!"}
Write-Host "Skript beendet"
```

Mittels "Read-Host" liest man Eingaben, "Remove-Item" löscht zum Beispiel Dateien (bzw. es tut so als ob, wenn es mit "-whatif" aufgerufen wird). Besonders interessant ist die Variable "**$?**": sie enthält den Fehlercode des letzten Aufrufs, gibt also Auskunft darüber, ob "Remove-Item" erfolgreich beendet wurde. "$?" gibt "FALSE" oder "TRUE" zurück.

Übung: Rückgabewert prüfen und Skript beenden

a) Verändern Sie das oben genannte Beispielskript:

- Lesen Sie, wie gehabt, einen Dateinamen mit "Read-Host" ein, Prüfen Sie im "whatif"-Modus, ob die Datei löschbar wäre.
- Prüfen Sie anschließend lediglich, ob "$?" **NICHT erfüllt** ist.
  (Denken Sie dabei an reguläre Ausdrücke, ein einzelnes Zeichen reicht um die Abfrage zu negieren.)
- Wenn die Datei nicht gelöscht werden kann, weil sie nicht existiert, so soll das Skript unmittelbar abbrechen. Es darf in diesem Fall keine Meldung "Skript beendet." erscheinen. Wenn jedoch die Datei existiert, dann soll abschließend eine Meldung erscheinen.

  Der dazu notwendige Ausdruck lautet in der PS "**break**" (entspricht "Exit Do" in VBScript)!

5.2 Skriptargumente auswerten

Übergibt man Argumente an ein Skript, so findet man diese in der Arrayvariablen $args wieder, $args[0] repräsentiert das erste Element. An den eckigen Klammern erkennt man das Array (Feld).

So könnte man, wenn auch etwas umständlich, auf nachfolgende Weise die übergebenen Argumente auswerten:

```
$i = ($args).count
write-host Sie haben $i Argumente übergeben.
While ($i -ne 0) {
  write-host $args[$i-1]
  $i = $i-1
}
```

Schon eleganter — und deutlich kürzer — kommt man so zum Ziel:

```
ForEach ($arg in $args) {
  write-host $arg
}
```

Die PS ist so "benutzerfreundlich", dass es sogar eine noch einfachere Variante gibt:

```
  write-host $args
```

Leerschritte trennen die einzelnen Argumente im Feld voneinander.

Die PS unterstützt auch die Übergabe von **benannten Argumenten**. Wie im nachfolgendem Beispiel zu sehen ist, wird die Variable explizit mit Namen beim Aufruf des Skripts übergeben, hierbei spielt die Reihenfolge der Übergabe keine Rolle mehr, das "$" wird beim Skriptaufruf durch "**-**" ersetzt. Optional wurde der Variablentyp im nachfolgenden Beispiel als [int] ("integer") definiert.

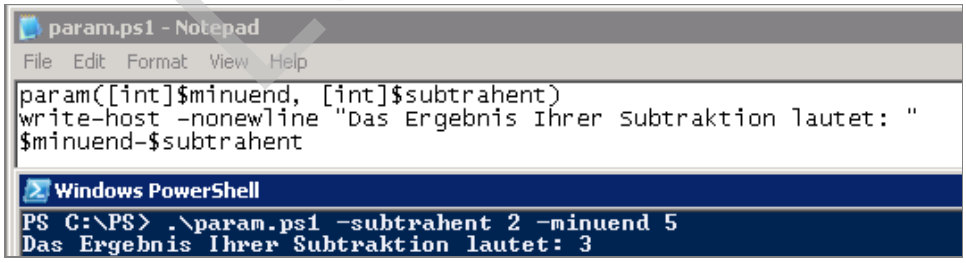


Abbildung 10: Benannte Parameter übergeben

Übung: Eine Fakultät berechnen

a) Sie möchten die Fakultät einer (ganzen) Zahl berechnen, zum Beispiel:
5! = 5*4*3*2*1 = 120

Die Fakultät von Null ist mit Eins definiert, also **0! = 1.**

Schreiben Sie ein Skript, dass Sie mit dem zu berechnenden Wert als Argument aufrufen:
`.\fakultät.ps1 5`
Ein Tipp: Lesen Sie zunächst den Hinweis im nachfolgenden Kasten zur "Rechenschwäche" der PS.

In der Interaktion mit dem Anwender entwickelt die PS ab und an scheinbar **Rechenschwächen**:

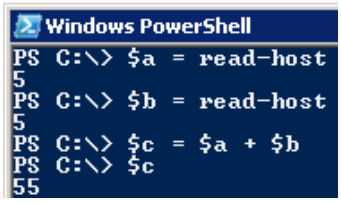


Abbildung 11: Rechenschwäche

Möchte man sicherstellen, dass Variablen nicht als Zeichenketten behandelt werden, so sollte man den gewünschten Typ benennen:

```
$c = [int]$a + [int]$b
```

In diesem Fall ist $c dann auch wieder 10.

5.3 Eine Funktion erstellen

Soll etwas immer mal wieder verwendet werden, so ist es hilfreich den fraglichen Programmcode in eine Funktion zu kapseln:

```
Function suche {cmd /c dir /s /a /b $args}
suche *.ps1
```

Alternativ kann der Suchfilter auch abgefragt werden:

```
Function suche {cmd /c dir /s /a /b $args}
suche(Read-Host "Was soll gesucht werden?")
```

Funktionen unterstützen in der gleichen Weise wie PS1-Skripte das Array "$args". Es ist möglich, aber nicht in jedem Fall nötig, die einzelnen Elemente des Feldes mittels "**args[**0,1,2,..**]**" anzusprechen. Will man arithmetische Funktionen mit einzelnen Elementen berechnen, so werden sie in aller Regel das gewünschte Elemente in der Form "**$arg[1]**" ansprechen; ansonsten laufen Sie Gefahr, dass das Argument als Zeichenkette misinterpretiert wird.

Möchte man beim Aufruf eines Skripts die Argumente weiterreichen, die der Benutzer als Skriptparameter übergibt, so reicht man lediglich $args (Skript) an $args (Funktion) weiter:

```
Function suche { cmd /c dir /s /a /b $args }
suche $args
```

Allgemein kann man Funktionen in dieser Form beschreiben:

function <*NAME*> (*PARAMETER*) { SCRIPTBLOCK }

5.4 Rückgabe von Ergebnissen aus Funktionen

So wie Argumente mittels **args[**0,1,2,..**]** in die Funktion gegeben werden, so übergibt die Funktion ein Feld an den Hauptteil zurück.

```
function zweinachfolger {
  $args[0] + 1
  $args[0] + 2
}
$zahl = 5
$ergeb = zweinachfolger $zahl
write-host -nonewline "Der 1. Nachfolger von $zahl lautet: "; $ergeb[0]
write-host -nonewline "Der 2. Nachfolger von $zahl lautet: "; $ergeb[1]
```

5.5 Benannte Argumente

Die Benutzerfreundlichkeit lässt sich noch weiter verbessern, in dem man mit benannten Argumenten arbeitet, die sich an Funktionen in gleicher Weise übergeben lassen, wie an Skripte (siehe oben).

```
Function nachfolger($a) { $a + 1 }
nachfolger 12
```

a)
```
Function netto_in_brutto{
  param($netto, $ust_satz=19)
  $netto * ( 1 * (1 + $ust_satz / 100) )
}
netto_in_brutto 22 16
```

b)
```
Function netto_in_brutto{
  param($netto, $ust_satz=19)
  $netto * ( 1 * (1 + $ust_satz / 100) )
}
netto_in_brutto -ust_satz 16 -netto 22
```

Das Beispiel b) zeigt, dass die Reihenfolge der Parameter hier frei wählbar ist. Beide Varianten führen zum gewünschten Ergebnis.

Übung: Mehrere Fakultäten berechnen

a) Sie möchten die Fakultät von Zahlen berechnen. Schreiben Sie ein Skript, das es dem Anwender erlaubt beliebig, viele Argumente zu übergeben. Rechnen Sie zu jedem übergebenen Wert die Fakultät aus. Verwenden Sie zur Berechnung der Fakultät eine Funktion, um den Programmcode sauber zu gliedern. Üblicherweise verwendet man für Fakultätsfunktionen das Wort "fact".

Beispiel, so könnte Ihre Ausgabe aussehen:

```
PS > .\fakultäten.ps1 5 7 11 12 44

Die Fakultät von 5 ist: 120
Die Fakultät von 7 ist: 5040
Die Fakultät von 11 ist: 39916800
Die Fakultät von 12 ist: 479001600
Die Fakultät von 44 ist: 2,65827157478845E+54
```

Abschlussfragen

1. Sie möchten ein Skript namens "myscript.ps1" ausführen und die darin enthaltenen Funktionen und Variablen nach Beendigung des Skripts weitherin zur Verfügung haben. Wie gehen Sie vor?

_____

_____

2. Sie lesen skriptbasiert zahlreiche Dateien in verschiedenen Ordnern ein. Sie können nicht ausschließen, dass der Anwender Dateien einliest für die er keine ausreichenden Zugriffsrechte besitzt. Wie können Sie Ihr Skript modifizieren, um das Skript auch in solchen Fällen nicht abbrechen zu lassen?

_____

_____

3. Ein Anwender ruft Ihr Skript wie folgt auf: "myscript.ps1 recurse png". Angenommen Sie wollen den zweiten Parameter "png" während der Skriptbearbeitung an der Konsole ausgeben lassen, wie lautet der Befehl?

_____

_____