PowerShell

- Ausgaben mit PowerShell filtern
- CSV mit Powershell einlesen
- Active Directory
  - Computerobjekte sortiert nach letztem Login ausgeben
- IPv4 Adresse mit PowerShell konfigurieren
- Fehlermeldungen
  - Powershell bricht ab - Meldung: "Ungültiger Aufzählungskontext"
- Ping Ergebnis in CSV schreiben
- SQL
  - Daten per PowerShell in eine Microsoft SQL Datenbank importieren
- Powershell Befehle Übersicht
- Hardware
  - Festplattengeschwindigkeit mit PowerShell testen
- Powershell Portscanner
- Uhrzeit und Datum formatiert in PowerShell ausgeben
- Windows Lizenz aus Bios mit PowerShell auslesen
- Windows Treiber über PowerShell installieren
- Eingabesprache mit der PowerShell auf Deutsch stellen
- Windows Serverrollen über die PowerShell installieren
- Änderungsdatum mit PowerShell ändern
- PowerShell Skript mit dem Windows Taskplaner automatisiert ausführen
- Active Directory Computer mit Angabe des Betriebssystem mit der PowerShell ausgeben

Ausgaben mit PowerShell filtern

Einleitung

Wenn du z.B. größere Log Dateien durchforsten möchtest, ist das filtern nach bestimmten Wörtern oder regulären Ausdrücken sehr hilfreich. Dies kannst du mit der PowerShell realisieren.

Anwendung

Du musst im ersten Schritt die Datei einlesen, dann können diese Daten weiter verarbeitet werden.

```
Get-Content "C:\temp\log.txt"
```

Diesen übergeben wir mit der Pipeline Variable an die Zeilenweise Ausgabe der Datei.

```
Get-Content "C:\temp\log.txt" | Out-String -Stream
```

Im letzten Schritt filterst du mithilfe von `Select-String` die Ausgaben. Dafür verwenden wir ein paar Argumente, die alle verwendet werden **können**.

Zuerst gibt es das Argument **-Pattern**. Mit diesem gibst du den Text an, nachdem gesucht werden soll. Es können hierbei auch Reguläre Ausdrücke verwendet werden.

Als zweites Argument gibt es noch **-SimpleMatch**. Damit teilst du der PowerShell mit, dass es sich **nicht** um einen Regulären Ausdruck handelt.

Als Letztes gibt es noch **-CaseSensitive**. Hier wird dann auch auf Groß- und Kleinschreibung geachtet.

Wir suchen in unserem Beispiel nach allen Einträgen, die das Wort "Hardware" enthalten.

Ein Wert nach dem gesucht werden soll

```
Get-Content "C:\temp\log.txt" | Out-String -Stream | Select-String -Pattern "Hardware" -SimpleMatch
```

Mehrere Werte nach dem gesucht werden soll

```
Get-Content "C:\temp\log.txt" | Out-String -Stream | Select-String -Pattern "Hardware","Software","Netzwerk" -SimpleMatch
```

Du erhältst nun gefilterte Ausgaben von PowerShell.

CSV mit Powershell einlesen

Einleitung

Du kannst mit der Powershell CSV-Dateien einlesen. Dies kannst du verwenden, um Listen von Daten einzulesen und diese z.B. als Parameter für Funktionen zu verwenden. So können automatisierte oder vereinfachte Prozesse gestaltet werden.

Code

Zuerst müssen wir eine CSV Datei erstellen. Im Beispiel erstellen wir eine Liste aus Rechnern, die später eingelesen und angepingt werden sollen. Über die Überschriften können die Daten dann später angesprochen werden.

	A	B	C	D
1	ID	PC-Name	IPv4-Adresse	Abteilung
2	1	PC-1	192.168.75.1	Einkauf
3	2	PC-2	192.168.75.2	Verkauf
4	3	PC-3	192.168.75.3	Verkauf
5	4	PC-4	192.168.75.4	Marketing
6	5	PC-5	192.168.75.5	Marketing
7	6	PC-6	192.168.75.6	IT
8	7	PC-7	192.168.75.7	IT
9	8	PC-8	192.168.75.8	IT
10	9	PC-9	192.168.75.9	Geschäftsleitung
11	10	PC-10	192.168.75.10	Geschäftsleitung
12				
13				

Im nächsten Schritt erstellen wir in Powershell ein **Array** in der die CSV Daten gespeichert werden. Dort wird dann über die Powershell-Funktion **Import-CSV** die Datei eingelesen. Dort geben wir als Parameter den Pfad und das Trennzeichen mit. Die Daten werden dann als **mehrdimensionales Array** in das **Array** übergeben. In der folgenden Schleife wird dann das **Array** durchgearbeitet. Über die Überschriften können dann die Daten angesprochen werden.

```
# Import der CSV Daten in das Array
$Daten = @()
$Daten = Import-Csv -Path "C:\Temp\computer.csv" -Delimiter ";"

# Schleife die Daten der CSV nacheinander durcharbeitet
foreach($d In $Daten)
{
```

```
# Code der ausgefÃ¼hrt wird
    if(Test-Connection -ComputerName $d.'IPv4-Adresse')
    {
        Write-Host("Der Computer mit dem Namen " + $d.'PC-Name' + " ist erreichbar!") -ForegroundColor Green
    } else {
        Write-Host("Der Computer mit dem Namen " + $d.'PC-Name' + " ist nicht erreichbar!") -ForegroundColor
Red
    }
}
```

Active Directory

Computerobjekte sortiert nach letztem Login ausgeben

Einleitung

In diesem kurzen Artikel geht es darum, wie ich in einem *Active Directory* mithilfe der **PowerShell** die **Computerkonten** der Domäne anzeigen lassen kann, sortiert nach dem letzten Login auf dem Rechner. Damit ist es mir möglich, das *Active Directory* etwas aufzuräumen. Nach der Durchführung des unten stehenden Codes erhalten wir eine Text-Datei, indem alle Computerkonten aufgelistet sind, und diese mit dem *ältesten Login* ganz oben stehen.

Computerkonten ausgeben

Um die Computerkonten auszugeben, muss in dem Befehl noch die **SearchBase** angepasst werden. Also die *Ordnungseinheit* und die *Domäne* angeben, in dem sich die **Computerkonten** befinden. Als letzte Anpassung müssen wir noch den Pfad der Datei ändern. Wenn im Anschluss der Befehl auf dem *Domänen-Controller* ausgeführt wird, erhalten wir eine Auflistung der Rechner.

```
Get-ADComputer -Filter * -SearchBase "OU=<Ordnungseinheit>,DC=<domain>,DC=<domain-endung>" -Properties Name,LastLogonDate | Select Name, LastLogonDate | Sort LastLogonDate | Out-File C:\tmp\logon.txt
```

IPv4 Adresse mit PowerShell konfigurieren

Einleitung

Mit der Hilfe von der PowerShell kannst du deiner Schnittstelle eine IP-Adresse zuweisen. Dies kann nützlich sein, wenn du z.B. öfters die IP-Adresse wechseln musst.

Hier erfährst du, wie du ein Skript schreibst, in der du deine IP-Adresse schneller wechseln kannst als andere, die über die GUI arbeiten.

Anwendung

Schnittstellen-Namen ermitteln

Im ersten Schritt müssen wir den Namen oder den Index der Schnittstelle ermitteln. Dies machst du, indem du dein PowerShell Fenster öffnest und folgenden Befehl in die Konsole eingibst.

```
Get-NetAdapter
```

Dort werden jetzt Schnittstellen mit der Interface-ID oder dem Interface Namen angezeigt. Dort suchst du das entsprechende Interface heraus, der du deine statische IP-Adresse geben möchtest.

Du suchst dir dort aus, ob du mit dem **Interface Namen** oder der **Interface-ID** arbeiten möchtest.

IP-Adresse ändern

Hier werden beide Wege beschrieben, wie du die IP-Adresse veränderst. Abhängig davon, ob du den Namen oder die ID verwenden willst.

Interface Namen

```
New-NetIPAddress -IPAddress <ipv4-adresse> -AddressFamily IPv4 -InterfaceAlias <interface-name> -DefaultGateway <gateway> -PrefixLength <prefix-lÃ¤nge>
```

Interface ID

```
New-NetIPAddress -IPAddress <ipv4-adresse> -AddressFamily IPv4 -InterfaceIndex <interface-id> -DefaultGateway <gateway> -PrefixLength <prefix-lÃ¤nge>
```

Wenn du jetzt den Befehl `ipconfig` in die CMD eingibst, siehst du, dass du die neue IP-Adresse gesetzt hast.

Fehlermeldungen

Fehlermeldungen

Powershell bricht ab - Meldung: "Ungültiger Aufzählungskontext"

Einleitung

In diesem kurzen Beitrag geht es kurz darum, wie wir verhindern können, dass unser **Powershell-Skript** mit der Meldung **ungültiger Aufzählungskontext** abbricht. Dieses Problem tritt auf, wenn das Ergebnis einer Rückgabe-Methode über **256 Objekte** zurückgibt.

Problem beheben

Zum Beispiel kann das Problem bei der Funktion `Get-ADComputer` auftreten. Das Skript könnte z.B. wie folgt aussehen:

```
Get-ADComputer -Filter *
```

Wenn das Problem bei dem oben genannten Skript auftritt, können wir das Skript etwas anpassen. Und zwar können wir mit den Parametern `-ResultPageSize` und -`ResultSetSize` festlegen, wie viele Objekte maximal zurückgegeben werden dürfen und wie groß die Anfangsgröße unserer Abfrage sein soll.

Wenn wir erwarten, dass wir knapp 2.000 Objekte zurückgegeben bekommen, und die Anfangsgröße dynamisch wachsen soll, können wir das Skript wie folgt anpassen:

```
Get-ADComputer -Filter * -ResultPageSize 2000 -ResultSetSize $null
```

Das Skript sollte jetzt beim nächsten Ausführen fehlerfrei durchlaufen.

Ping Ergebnis in CSV schreiben

Einleitung

Mit diesem kleinen Skript kann man in eine CSV Datei schreiben lassen, ob ein Ping erfolgreich durchgegangen ist oder nicht. Es wird dann immer die Uhrzeit und das Datum dazu geschrieben, um nachzuverfolgen, wann das genau war.

Du kannst in dem Skript festlegen, in welchen Abschnitten die Pings durchgeführt werden, wie viele Durchläufe durchgeführt werden sollen, und unter welchem Pfad die Logdatei liegen soll und wie diese heißt.

Die Datei wird bei jeder Durchführung gelöscht und neu erstellt, damit keine Altlasten zur Verwirrung sorgen können.

Code

```
$path = "C:\logs"
$filename = "result.csv"
$destAddress = "srv-windows1"


    Write-Host("============================") -ForegroundColor Green
    Write-Host("Starte Connection Tester") -ForegroundColor Green
    Write-Host("Ziel: " + $zielAdresse) -ForegroundColor Green
    Write-Host("Entwickelt von: Phillip Unzen") -ForegroundColor Green
    Write-Host("============================") -ForegroundColor Green
    Write-Host(" ")

    if(Test-Path -Path ($path + "\" + $filename))
    {
        Remove-Item -Path ($path + "\" + $filename)
        New-Item -Path $path -Name $filename -ItemType File
    } else {
        New-Item -Path $path -Name $filename -ItemType File
    }
```

```
for($i = 0; $i -le 10000; $i++)
{
   if(Test-Connection -ComputerName $destAddress -Count 1)
   {
      Write-Host("Ping erfolgreich!") -ForegroundColor Green
      $str = (Get-Date).ToString("dd.MM.yyyy") + ";" + (Get-Date).ToString("HH:mm:ss") + ";" + "Erfolgreich"

   } else {
      Write-Host("Keine Verbindung") -ForegroundColor Red
      $str = (Get-Date).ToString("dd.MM.yyyy") + ";" + (Get-Date).ToString("HH:mm:ss") + ";" + "Fehler"
   }

   $str | Out-File -FilePath ($path + "\" + $filename) -Encoding utf8 -Append

   Start-Sleep 2
}
```

SQL

SQL

Daten per PowerShell in eine Microsoft SQL Datenbank importieren

Einleitung

In diesem Beitrag geht es kurz darum, wie wir mit einem **PowerShell Skript**, Daten in eine **Microsoft SQL Datenbank** importieren können. Dies verwenden wir, wenn wir z.B. Daten aus einer Abfrage heraus, in eine Datenbank importieren möchten, um aus dieser z.B. Auswertungen zu fahren.

Durchführung

Um die Daten nun in eine Datenbank zu importieren, brauchen wir natürlich ein entsprechendes Skript. Für diesen Vorgang habe ich eine Funktion geschrieben, welche wir in vorhandene Skripte einfach einbauen können. Je nachdem, ob die **Windows NT Authentifizierung** oder die **SQL-Server Authentifizierung** gewählt wird, muss die entsprechende Funktion verwendet werden:

Windows NT Authentifizierung

```
function Run-SQL {
    Param(
        [string]$sqlServer,
        [string]$sqlDatabase,
        [string]$sqlQuery
    )

    $connectionParameter = "Data Source=$sqlServer;Integrated Security=SSPI;Initial Catalog=$sqlDatabase"
    $sqlConnection = New-Object System.Data.SqlClient.SqlConnection($connectionParameter)
    $sqlCommand = New-Object System.Data.SqlClient.SqlCommand($sqlQuery, $sqlConnection)
    $sqlConnection.Open()
    $sqlDataAdapter = New-Object System.Data.SqlClient.SqlDataAdapter $sqlCommand
    $dataSet = New-Object System.Data.DataSet
```

```
    $sqlDataAdapter.Fill($dataSet) | Out-Null
    $sqlConnection.Close()
    $dataSet.Tables
}
```

SQL-Server Authentifizierung

```
function Run-SQL {
    Param(
        [string]$sqlServer,
        [string]$sqlDatabase,
        [string]$sqlQuery,
        [string]$sqlUser,
        [string]$sqlPassword
    )

    $connectionParameter = "Data Source=$sqlServer;UserID=$sqlUser;Password=$sqlPassword;Initial
Catalog=$sqlDatabase"
    $sqlConnection = New-Object System.Data.SqlClient.SqlConnection($connectionParameter)
    $sqlCommand = New-Object System.Data.SqlClient.SqlCommand($sqlQuery, $sqlConnection)
    $sqlConnection.Open()
    $sqlDataAdapter = New-Object System.Data.SqlClient.SqlDataAdapter $sqlCommand
    $dataSet = New-Object System.Data.DataSet
    $sqlDataAdapter.Fill($dataSet) | Out-Null
    $sqlConnection.Close()
    $dataSet.Tables
}
```

Jetzt kann in dem PowerShell Skript, mit Aufruf der Funktion und den entsprechenden Parametern, eine Abfrage an den **SQL-Server** getätigt werden. Ein Aufruf der Funktion könnte dann z.B. so aussehen:

```
Run-SQL -sqlHost "<IP SQL-Server>" -sqlDatabase "<Datenbank Name>" -sqlQuery "INSERT INTO
<Tabellenname> (<Spalten Name>) VALUES (<WERTE>);"
```

Sobald der SQL-Befehl auf die eigenen Bedürfnisse angepasst ist, und der Benutzer Zugriff auf die Tabelle hat, werden Daten erfolgreich in die Datenbank geschrieben.

Powershell Befehle Übersicht

Einleitung

Hier findest du einige hilfreiche Befehle, die für das Powershell Skripten wichtig sein könnten. Kann als Nachschlagewerk manchmal ganz praktisch sein. Die Liste ist nicht sortiert, da nach und nach neue Einträge hinzugefügt werden.

Übersicht

Befehl	**Funktion**
Start-Sleep <sekunden>	Startet einen Timeout. Code wird fortgeführt, wenn die Sekunden abgelaufen sind.
Test-Path -path <Pfad mit Datei / Pfad des Ordners>	Überprüft, ob eine Datei oder ein Ordner vorhanden ist und gibt **true** oder **false** zurück.
Remove-Item -path <Pfad mit Datei / Pfad des Ordners>	Löscht Datei/en oder Ordner.
New-Item -path <Pfad> -name <Name> -ItemType <Typ> (Directory | File)	Legt neue Ordner oder Dateien an.
Get-Date	Aktuelles Datum mit Uhrzeit erhalten
Get-Random	Zufallszahl ausgeben

> (Liste wird weitergeführt...)

Hardware

Hardware

Festplattengeschwindigkeit mit PowerShell testen

Einleitung

In diesem kleinen Artikel geht es kurz darum, wie wir mit der **PowerShell** unsere **Festplattengeschwindigkeit (Lesend & schreibend)** auslesen können.

Geschwindigkeit auslesen

Um die Geschwindigkeit auszulesen, müssen wir im ersten Schritt eine **Powershell Konsole** mit **administrativen Berechtigungen** starten.

Dort geben wir den folgenden Befehl ein:

```
winsat disk -seq -read -drive c
winsat disk -seq -write -drive c
```

Als Ausgabe erhalten wir als Erstes die lesende Geschwindigkeit, und danach die schreibende Geschwindigkeit unserer Festplatte.

> **Info:** Wir können den Buchstaben `c` hinter `-drive` durch einen anderen Buchstaben ersetzen, falls wir eine andere Festplatte testen möchten als die Betriebssystem Festplatte.

Powershell Portscanner

Einleitung

Hier findest du einen Portscanner, der komplett in PowerShell geschrieben ist. Das heißt, du kannst diesen auf jedem Windows-Rechner ohne weitere Software-Installationen ausführen.

Dieser ist, dafür, dass er in PowerShell geschrieben ist, leider nicht so schnell wie die üblichen Portscanner. Trotzdem funktioniert er gut!

Du musst bei dem Skript nur die IP-Adresse des zu scannenden Hosts angeben, und den Port Bereich. Das Skript beschränkt sich auf den **Well Known Ports** Bereich *(1 - 1023)*.

Code

```
$ipv4Address = "192.168.1.35"


for($i = 1; $i -lt 1024; $i++)
{
   if(Test-NetConnection $ipv4Address -InformationLevel Quiet -Port $i)
   {
      Write-Host("Port $i ist offen") -ForegroundColor Green
   }
}
```

Uhrzeit und Datum formatiert in PowerShell ausgeben

Einleitung

In PowerShell kannst du das aktuelle Datum oder die aktuelle Uhrzeit zurückbekommen, um mit dieser weiterzuarbeiten oder andere Dinge auszugeben.

Hier findest du die Codebeispiele wie du zum Beispiel nur das Datum, oder nur die Uhrzeit erhalten kannst.

Code

Uhrzeit und Datum erhalten

```
Get-Date
```

Uhrzeit erhalten (24-Stunden Format)

```
(Get-Date).ToString("HH:mm:ss")
```

Uhrzeit erhalten (12-Stunden Format)

```
(Get-Date).ToString("hh:mm:ss")
```

Uhrzeit erhalten (ohne führende Nullen)

```
(Get-Date).ToString("h:m:s")
```

Tage / Monate zu einem Datum hinzufügen und dieses ausgeben

```
(Get-Date).AddDays(5).AddHours(3).ToString("dd.MM.yyyy_HH:mm")
```

Windows Lizenz aus Bios mit PowerShell auslesen

Einleitung

Bei manch einer Neuinstallation von Windows, kann es dazu kommen, dass der Lizenz Key aus dem BIOS nicht erkannt und eingefügt wird. Um den Lizenzkey dann in dem Betriebssystem einzugeben, brauchen wir diesen erst einmal. Dafür gibt es diverse Tools, oder wir nutzen die PowerShell, um mit Bordmitteln diesen Lizenzkey auszulesen.

Windows Key auslesen

Um den Lizenzkey aus dem BIOS auszulesen, müssen wir im ersten Schritt die **PowerShell** mit **administrativen Berechtigungen** öffnen. Dort angekommen, müssen wir nur den nachstehenden Code eingeben. Sobald wir den Befehl mit einem **Enter** abschicken, wird uns der Lizenzkey in der Ausgabe angezeigt.

```
(Get-CimInstance -Query "select * from SoftwareLicensingService").OA3xOriginalProductKey
```

Windows Treiber über PowerShell installieren

Einleitung

Sobald wir einen Windows-Client oder Server installieren, müssen wir Treiber installieren. In diesem Beitrag gehe ich kurz darauf ein, wie wir die Treiber über die **PowerShell** installieren können.

Treiber installieren

Im ersten Schritt müssen wir mit `cd` in den Ordner wechseln, in dem die Treiber liegen.

```
cd D:
```

Dort angekommen, geben wir den folgenden Befehl ein. Mit diesem Befehl wird der Ordner **rekursiv** gescannt und die benötigen Treiber daraus installiert.

```
Get-ChildItem -Filter *.inf -Recurse | Select-Object FullName | ForEach-Object {pnputil -a $_.FullName}
```

Sobald das Skript durchgelaufen ist, haben wir alle **Treiber** installiert.

Eingabesprache mit der PowerShell auf Deutsch stellen

Einleitung

In diesem Beitrag erläutere ich kurz, wie wir mit der **PowerShell** die Eingabesprache unseres **Windows Systems** auf Deutsch stellen können. Dazu ist lediglich nur ein Befehl nötig.

Eingabesprache umstellen

Um jetzt die **Eingabesprache** auf **Deutsch** zu stellen, müssen wir im ersten Schritt die **PowerShell** öffnen. Sobald das Fenster auf ist, geben wir nur den folgenden Befehl ein. Dann erfolgt die Eingabe auf **deutscher Sprache**.

```
Set-WinUserLanguageList -LanguageList de-DE
```

Windows Serverrollen über die PowerShell installieren

Einleitung

In diesem Beitrag erkläre ich kurz, wie wir über die **PowerShell Windows Serverrollen** installieren können.

Verfügbare Rollen anzeigen lassen

Damit wir erstmal wissen, welche Rollen wir installieren können, müssen wir uns diese Rollen erstmal anzeigen lassen. Dazu verwenden wir den **PowerShell Befehl** `Get-WindowsFeature`.

```
Get-WindowsFeature
```

Wir erhalten dann eine Liste mit allen **Rollen**, die wir auf unserem Server jetzt installieren können. Aus dieser Liste suchen wir eine Rolle heraus. Für unser Beispiel installieren wir die **DNS Rolle**, damit wir einen **DNS-Server** installiert haben.

Rolle auf dem Server installieren

Um jetzt die entsprechende **Rolle** auf dem Server zu installieren, verwenden wir jetzt den **PowerShell Befehl** `Install-WindowsFeature`.

```
Install-WindowsFeature DNS
```

Wir können dazu auch die **Management-Tools** mitinstallieren. Damit erhalten wir dann in der Administrationsoberfläche des **Windows Servers** die Programme, um den **DNS-Server** zu verwalten. Dafür müssen wir lediglich den Parameter -`IncludeManagementTools` zu dem Befehl hinzufügen.

```
Install-WindowsFeature DNS -IncludeManagementTools
```

Nachdem der Fortschrittsbalken durchgelaufen ist, ist die Installation abgeschlossen.

Änderungsdatum mit PowerShell ändern

Einleitung

Mit der Hilfe von der **PowerShell** können wir das **Änderungsdatum** von Dateien ganz einfach ändern.

Datum und Uhrzeit ändern

Um das **Datum** und die **Uhrzeit** jetzt zu ändern, müssen wir lediglich nur einen Befehl absetzen. In diesem müssen wir nur noch die Parameter für den **Pfad** und den **Zeitstempel (Datum und Uhrzeit)** anpassen. Dann wird durch das Ausführen des Skriptes das Datum und die Uhrzeit verändert.

```
$files = gci -path "PFAD" -Recurse
foreach($f in $files)
{
   $f.LastWriteTime = Get-Date "2016-01-01 00:00:01"
}
```

PowerShell Skript mit dem Windows Taskplaner automatisiert ausführen

Einleitung

Bei sich wiederholenden Prozessen lohnt es sich ein **PowerShell Skript** zu entwickeln, damit solche Prozesse schneller abgearbeitet werden können. Um die Ausführung von Skripten zu automatisieren, kann man den **Taskplaner** von **Windows** nutzen. Mit diesem ist es möglich, Skripte und Programme automatisch zu beliebigen Ereignissen auszuführen.

Wie man ein PowerShell Skript automatisiert ausführen lässt, wird in dieser Anleitung kurz erklärt.

PowerShell Skript ausführen

Im ersten Schritt müssen wir das PowerShell Skript erstellen. Sobald wir dies erledigt haben, müssen wir das Skript auf dem Rechner ablegen oder auf einen Netzwerkpfad, wo der Benutzer des auszuführenden Computers Zugriff hat.

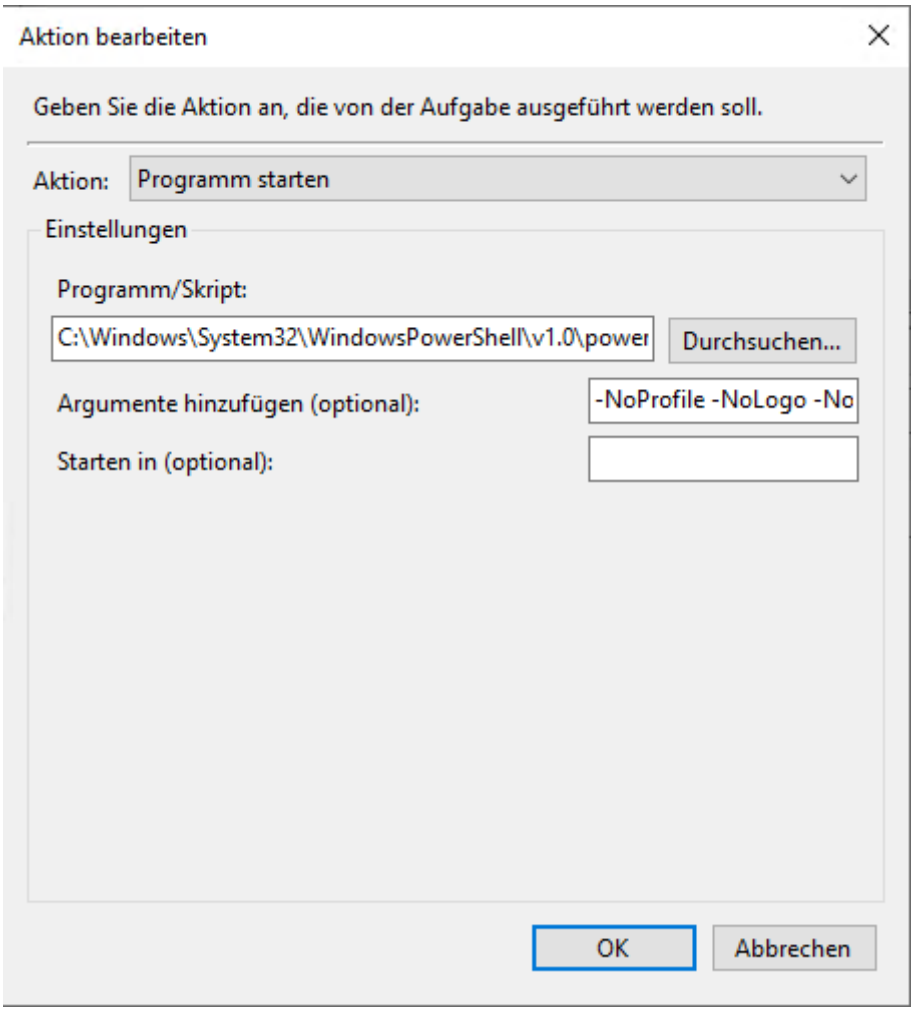
Aufgabe erstellen

Jetzt erstellen wir die Aufgabe, indem wir die **Aufgabenplanung** öffnen, und öffnen mit einem Rechtsklick das *Kontextmenü*. In dem *Kontextmenü* wählen wir den Punkt **einfache Aufgabe erstellen** aus. Dort vergeben wir einen gewünschten Namen und klicken uns weiter durch das Setup.

Jetzt fügen wir eine Aktion hinzu. Da wir ein PowerShell Skript ausführen wollen, öffnen wir die PowerShell Engine und geben das Skript nur als Parameter mit.

Die Maske wird dann wie folgt ausgefüllt:

```
Aktion: Programm starten
Programm/Skript: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Argumente hinzufÃ¼gen: -NoProfile -NoLogo -NonInteractive -ExecutionPolicy Bypass -File DATEI.ps1
```



Als letzten Schritt müssen wir, sobald das Setup fertig ist, die **Eigenschaften** der Aufgabe öffnen. Dort müssen wir lediglich nur einen Haken bei "*Mit höchsten Privilegien ausführen*" setzen. Danach drücken wir auf Ok und unsere Aufgabe wurde erstellt.

> **Wichtig:** Unter **Trigger** stellen wir ein, wann das Skript automatisiert ausgeführt wird. Häufig wird hier eine bestimmte Uhrzeit in einem festem Intervall eingestellt.

Active Directory Computer mit Angabe des Betriebssystem mit der PowerShell ausgeben

Einleitung

In diesem Beitrag erkläre ich kurz, wie wir auf unserem Domänencontroller mit der PowerShell alle Computer mit Angabe des installierten Betriebssystems ausgeben können. Dadurch können wir schnell eine Auswertung darüber ziehen, welche Rechner bei uns in der Domäne installiert sind, und welches Betriebssystem einsetzen.

Computer ausgeben

Um die Ausgabe zu erhalten, müssen wir im ersten Schritt die PowerShell-Konsole öffnen. Entweder die *ISE (Integrated Script Environment)* oder direkt die *Kommandozeile*. Dort fügen wir den folgenden Code ein, und ändern den Pfad zu der entsprechenden Datei:

```
Get-ADComputer -Filter * -SearchBase "OU=<Ordnungseinheit>,DC=<DOMAIN>,DC=<LOCAL>" -Properties * | Select Name, OperatingSystem | Out-File -FilePath <Pfad>
```

Als Ausgabe erhalten wir eine Liste mit den Computern mit der Angabe des Namens und des verwendeten Betriebssystems.