# PowerShell


- Active Directory
  - Computerobjekte sortiert nach letztem Login ausgeben
- Fehlermeldungen
  - Powershell bricht ab - Meldung: "Ungültiger Aufzählungskontext"
- SQL
  - Daten per PowerShell in eine Microsoft SQL Datenbank importieren
- Hardware
  - Festplattengeschwindigkeit mit PowerShell testen

# 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
```

# 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.

# 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.

# 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.