Powershell Befehle

Befehlsübericht

Comandlet	Beschreibung
`Get-command`	Zeigt alle Befehle an
`get-host`	Zeigt die Version von Powershell an
`Get-Computerinfo`	Gibt die Computerinfo wieder
`Get-Content`	Damit kann eine Datei ausgelesen werden Beispiel: Get-Content C:\Users\XY\text.txt
`Get-ChildItem`	Ordner Anzeigen
`clear-host`	Bildschirm leeren
`Write-host`	Print Befehl
`Read-host`	Benutzereingabe in der Konsole
`Set-Location`	Ändert das Verzeichnis von Powershell
`Start-Transcript`	Aufzeichnen der Powershell
`Start-Sleep`	Zeit warten
`Get-PSDrive`	Zeigt alle zur verfügungstehenden Laufwerke an
`Get-NetIPConfiguration`	Zeigt Netzwerkkarte an
`Get-Help`	Zeigt die Hilfe zu einem Cmdlet an Beispiel: Get-Help Test-Connection auführlich: -detailed
`Get-Help x  -Online`	Zeigt die Umfangreichere Onlinehilfe an Beispiel: Get-Help Test-Connection -Online
`Test-Connection`	Ping ausführen, kann mehrere Pings gleichzeitig ausführen Beispiel: Test-Connection orf.at,8.8.8.8 -Count 1 -Quiet
`Test-NetConnection`	Beipiel: Test-NetConnection 25.25.15.3 -TraceRoute Macht einen Ping auf ein Gerät. - TraceRoute gibt den Weg zurück
`Tracert`	Genauere TraceRoute
`Req query`	Auslesen von Registry Eintragen
`Remove-Item`	Datei löschen

Comandlet	Beschreibung
`Psexec`	Remotezugriff Ausführen eines Befehls auf einem Remoterechner
`Copy-item`	Beispiel: Copy-Item -Path Pfad+Datei -Destination Zielpfad
`CSV`	https://techexpert.tips/powershell/powershell-read-lines-from-csv-file/
`Get-Hotfix`	Zeigt die letzten Windowsupdates an

Ausführungsrichtlinien

Um zu verhindern, dass ps1 Dateien unberechtigter weise ausgeführt werden können gibt es die ExecutionPolicy. Diese ist normalerweise undifined und damit können keine PS Dateien ausgeführt werden.

```
# Anzeigen
Get-ExecutionPolicy -List

# Ändern
Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy Bypass
```

Vergleichsoperatoren

Befehl	Beschreibung
-eq	gleich
-ne	ungleich
-lt	kleiner
-le	kleiner oder gleich
-gt	größer
-ge	größer oder gleich

Parameterbeschreibung

-WhatIf	Zeigt was geschehen würde wenn…
-Confirm	Fordert zur Bestätigung auf
-Verbose	Zeigt an was gerade geschieht

Weitere Befehle

Systeminformationen anzeigen

```
msinfo32
```

Computername anzeigen:

```
nslookup +ip-Adresse
```

Systeminformationen anzeigen:

```
Systeminfo
```

Gruppenrichtlinien als HTML ausgeben

```
gpresult /H datei.html
```

Gruppenrichtlinien manuell updaten

```
gpupdate /?
```

Anpingen...

```
ping -a
```

Weg verfolgen zu einem Rechner

```
tracert
```

Code-Beispiele

Liste von ip-Adressen anpingen

```
1..255 | ForEach { Write-Host 169.254.66.$_, "-",
([System.Net.NetworkInformation.Ping]::new().Send("169.254.66.$($_)")).Status}
```

Remote einschalten

```
Set-Service -ComputerName haeb2pc4 -Name WinRM -StartupType Automatic -Status Running
Invoke-Command -ComputerName haeb2pc4 -Scriptblock {
    Get-ComputerInfo # In dem Scriptblock kann alles eingegeben werden, was auch auf einem Lokalen Rechner
verwendet werden kann
}
Set-fService -ComputerName haeb2pc4 -Name WinRM -StartupType Manual
```

Anmeldeinformationen speichern

```
$credentials = Get-Credential
$credentials | Export-Clixml -Path "C:\Pfad\Zur\Datei\credentials.xml"
```

Anmeldeinformationen abrufen

```
$credentials = Import-Clixml -Path "C:\Pfad\Zur\Datei\credentials.xml"
```

Netzlaufwerk verbinden

```
$driveLetter = "Z"
$networkPath = "\\ServerName\SharedFolder"

net use $driveLetter: $networkPath /user:$($credentials.UserName)
$($credentials.GetNetworkCredential().Password)
```

Trennen

```
$driveLetter = "Z:"
net use $driveLetter /delete
```

CMDledts

- Get- Verben nehmen niemals Änderungen vor

CMDlets erkunden

```
Start-Transcript #Logfile erzeugen
Set-Location HKLM:  #Wechselt in die Registry

## Parameter

- Whatif #zeigt was geschen würde wenn
- Confirm #Fordert zur Bestätigung auf
- Verbose Zeigt an was gerade geschiet
```

Hilfen anzeigen

Um mit den Comandos besser zurecht zu kommen kann man sich die Hilfen dazu wie folgt anzeigen lassen:

`Get-Help Get-Date -ShowWindow`

Bei dem Befehl wird die Hilfe von Get-Date in einem neuen Fenster angezei

Pipe

Die Pipe "|" lenkt das Ergebnis von der linken Seite an die rechte Seite weiter

Get-Hotfix angepasst

```
Get-Hotfix
```

Foreach

```
$testrechner = "10.198.48.102","10.198.48.103"  # Erstellt ein Array mit zwei IP-Adressen

Foreach ($i in $testrechner)                    # Foreach Schleife die jedes Element in Testrechner durchgeht
{ Test-Connection $i -Count 1 -Quiet }          # Führt einen Ping auf die Elemente aus dem Array aus
```

Formatierung und Ausgabe

Format Table

```
# Ausgabe
Get-Process -Name VSSVC

# Umleitung an Format-Table
Get-Process -Name VSSVC | Format-Table Id,ProcessName

# Herausfinden wie Atribite heißen
Get-Process -Name VSSVC | Get-Member
```

Format List

```
Get-Process -Name VSSVC | Format-List ID,ProcessName
```

Out-CMDlets

```
Get-Command | Out-File C:\Temp\cmdlets.txt # Speichert die Commandos in eine Datei aus

# OUt-Gridview erstellt ein Grafisches Raster

Get-Process | Out-GridView -PassThru | Stop-Process # Damit kann ich ein Fenster erstellen und danach den Prozess beenden
```

Variable $_

($_ und $PSItem ist das selbe)

Beinhaltet das aktuell von der Pipeline verarbeitet Objekt

```
# Beispiel
Get-Process | Where-Object { $_.CPU -GT 10 } # Filtert nach Objekten die > 10 sind Bei jedem Durchlauf wird geschaut ob das Objekt was durch $_ zurückgegeben wird größer ist
```

Filtern

```
Get-ADComputer -Filter 'enable -eq $false' | irgendwas
# Hier wird vor der Pipline schon gefilter. Dadurch wird das Programm viel schneller, da es nicht alle Computer in die Pipeline weiter gibt.
```

Sort-Object

```
Get-Process | Sort-Object
```

Select-Object

```
Get-Process | Sort-Object -Proberty XXX | Select-Object -Last 5
# Mit Select-Object * kann man sich alles anzeigen lassen.

# Spezilles Parameter aus einem Objekt am Beispiel Get-ComputerInfo
Get-ComputerInfo | Select-Object OsName,
```

Ping mit ExpandProberty

```
$b = Get-ADComputer -Filter * | Select-Object -ExpandProperty Name
# Dadurch wird der Wert abgerufen. Damit kann man weiter arbeiten, da es kein Objekt ist.

Test-Connection $b
```

Objekte und Klassen

Wie bei Python kann auch in Powershell eine Klasse erstellt werden. Hier ein Beispiel

```
# Klassen

class Person {
    [string]$Name
    [int]$Age

    Person([string]$name, [int]$age) {
        $this.Name = $name
        $this.Age = $age
    }



    [void] SayHello() {
        Write-Host "Hello, my name is $($this.Name) and I am $($this.Age) Years old"

    }
}

$person = [Person]::new("John", 30)
$person.SayHello()
```

Powershell Kurs für Administratoren und Udemy

Schulungsunterlagen

> Willkommen bei Skillpipe!
> Hallo Hermann Pelzer,
> Mit Skillpipe kannst du deine Trainingsmaterialien jederzeit und überall auf deinem Lieblingsgerät aufrufen. Dein Lernfortschritt, die Markierungen, Notizen und Lesezeichen werden automatisch auf all deinen Geräten synchronisiert.
>
> Zugriff auf Skillpipe

Dateien

202205-PowerShell-Wiederholungsfragen.pdf

Alle Dateien

Als Administrator starten

Verknüpfung bearbeiten und dort auswählen die Powershell als Administrator zu starten

Welche Powershell version

`$PSVersionTable`

Commandlet

Befehle

		Beispiel
Get-Process	Taskmanger	
Get-Command		Get-Command -Verb Install
Get-Help Get-ChildItem -Online	Zeigt die Onlinehilfe an. Was aber bei uns meistens nicht funktioniert	
Get-ChildItem	Inhalt eines Ordner anzeigen	
where-Object	Zum Filtern von Ergebnissen	`-Filterscript {$_.handles -gt 1000}`
Select-Object	Hier kann man sich bestimmte Parameter anzeigen lassen	`Select-Object CsName, OsNmae, OsInstallDate`
Format-Table		

		Beispiel
Get-Member	Gibt eigenschaft wieder	
Get-NetIPConfiguration	Hier kann auf die ergebnisse wie auf ein Array zugegriffen werden	
Format-List	Damit kann man ausgegebene Listen formatieren	
	`>>` mit den beiden Pfeilen kann man an eine Datei anhängen	
	`Get-Content .\myfile.txt -Wait`	-Wait Wenn man live zusehen will, wie zum Beispiel eine Textdatei angepasst wird
`Export-Csv`	Als CSV ausgeben	
`Import-Csv`		
`Export-Clixml`	Als XML exportieren	Dadurch werden nicht alle daten in einen String umgewandelt wie bei CSV
`Add-Computer`	Computer zu einer Domain hinzufügen	Für Remote muss man Credential angeben weil man sich sonst nicht anmelden kann.
`$cred = Get-Credential`	Damit kann man Benutzername und Passwort eingeben	
`$cred = New-Object -TypeName pscredential -ArgumentList 'Administrator',(ConvertTo-SecureString -String 'Pa$$w0rd' -AsPlainText -Force)`	Damit kann man die Credentials direkt einspeichern, damit das nicht extra eingegeben werden muss.	
`Test-ComputerSecureChannel -Repair`	Kann Computer-Clients in der Domain reparieren.	Mal ausprobieren, wenn es hier probmele gibt. Test-ComputerSecureChannel kann annzeigen ob der Computer einen Fehler hat.
`1..3`	1 bis 3	eine Schleife die von 1 bis 3 zählt
`(Get-Date).AddDays(4)`	Gibt das heutige Datum + 4 Tage aus	
`"{} {}" -f $Temperatur,$Bedeckung`	-f arbeitet wie `.format()` aus python	
`New-Item -Path \\rechner\C$\ -Name ScriptShare -ItemType Directory`	Erstellt auf einem Rechner einen Ordner in C	

Notizen

Cmdlets bestehen immer aus Verb-Noun

Get bekomme etwas

Set etwas verändern

new um was neues zu erstellen

> Tipp: Mit Strg + Leertaste kann man sich alle Befehle anzeigen lassen

aliase: Andere Schreibweise für befehle. Machen dann aber das gleich wie das original

Wildcards * oder ? wobei Fragezeichen nur ein Zeichen ersetzt

Hilfe Updaten: Update-Help

Hilfe ausfühlricher:

```
Get-Help name -Detailed
```

Weiteres zu Hilfen:

```
get-help about_if -ShowWindow #zeigt wie man if verwenden kann.
```

```
Show-Command # Damit kann man Cmdlets erstellen
```

Umwandeln von Bite

`Wert / 1GB` # Gibt GB wieder

Variablen

```
$a = 100 # Variablen deklarieren


#Variable auslesen
$a


Get-Variable # Zeigt alle bestehenden Variablen an
```

```
# Arrays
# Leeres Array erstellen:
$myprocesses = @()

# Hashtables (=Dictionary)
$myHT = @{ Marke='Audi'; kw=140; Coupe=$false }
# Abfrage der Werte:
$myTH.Marke
```

Man kann auch Variablen innerhalb einer Zeichenkette mit Anführungsstriche direkt verwenden

```
"das ist ein Text mit $variable"

# Hashtablewerte in einen String übergeben
"Das ist ein Text mit einem Value $($wetter.temperatur)"
```

Skript muss immer in geschweiften Klammern sein

```
Get-Service | Where-Object -FilterScript { $_.Name -like 'x*' }
```

Filtert das Service auf Objekte die mit X beginnen

Aus einem Computer einen DomainController machen

Man muss erst mal die Tools installiert werden, damit ein Rechner als DomainController arbeiten kann.

```
Get-WindowsFeature # Listet auf welche Features der PC hat

# Das ist nur auf einem Server möglich
Install-WindowsFeature -Name AD-Domain-Services -IncludeAllSubfeatures -IncludeManagementTools
```

Datum verwenden

```
$myDate = Get-Date -Year 2021 -Month 5 -Day 25
$myDate.Date # Uhrzeit wird nicht betrachtet
| where-Object { $_.lastwritetime.D -eq $myDate } # Filtern zum 25.05.
```

Provider

Sind Programme, die auf andere Programme zugreifen

`Get-PSProvider`

Provider	
Get-PSDrive	Zeigt alle Laufwerke an
`New-PSDrive - Name Scriptshare -Root \\Lon-Srv\c$\ScriptShare -PSProvider FileSystem`	Bindet das Laufwerk vom Server in die Powershell auf dem Client ein und man kann damit arbeiten.

Spezielle Variable:

$env: - Damit kann man auf Variablen und auf Computereigenschaften zugreifen

WMI

> Windows Management Instrumentation:

Cimclass

WQL ist die Sprache für WMI

Registry Editor

HKEY_Current_User: Einstellungen die der User macht
HKEY_Local_Machine: Einstellungen der Machine für alle User

```
cd hkcu #Wechsel ins Regestry Laufwerk


Get-Childitem #listet alle Schlüssel auf


#Neues Element in Software erstellen
cd .\Software\
New-Item -Name Az040 -Path . -ItemType Key
cd .\Az040\
```

```
New-ItemProberty -Path . -Name TlnAnzahl -Value 6 -ProbertyType dword
Set-ItemProberty -Path . -Name TlnAnzahl -Value 7 #Erstellt keinen neuen Wert kann aber einen vorhandenen bearbeiten
(Get-ItemProberty -Path .).Level # gibt den value von level wieder
```

mit sapien kann man aus ps1 auch exe machen kostet aber geld

Sapien Editor

isesteroids kann die ise erweitern

Skripte

> Skripte Starten
>
> - Pfad angeben um das Skript zu starten
> - Ausführungsrichtlinie kann verhindern, dass ein Skript gestartet wird
>   - Get-ExecutionPolicy -List
>   - Get-ExecutionPolicy > Restricted
>   - mit Bypass wird die Policy nicht überprüft

Beispiele

```
<# Mehrzeilige Kommentare
geht so.
ziemlich einfach#>
hostname

#region Region
mit dem region kann man bereiche ausklappbar machen fuer bessere uebersicht
#endregion

# Um einen Parameter bei starten eines Skripts abzufragen
param (
   $status
)
```

Parameter in Skript übergeben

```
# Verpflichtende Parameter
param (
    [parameter(Manatory=$true, Helpmessage='Bitte Parameter eingeben')]
    [validateSet('Running','Stopped')] #Welche werte sind erlaubt
    $status
)
```

Bedingungen

if

```
# if

if ( $a -gt 50 ) {

    mach das
}
else {

    mach jenes
}
```

Switch

```
# Switch

Switch ($color) {
    'Yellow'{
        'Die Farbe ist gelb'
    }
    default {
        "Die Farbe ist unbekannt"
    }
}

# Schleife abbrechen
break
return
```

Schleifen

For

```
# for ( $zähler; Endbedingung; Schrittweite)

for ( $i = 1; $i -le 5; $i = $i +1 ) {
    "Der Zähler hat den Wert $i"
}
```

foreach

> Hilfreich bei Arrays zu bearbeiten

```
$colors = 'White', 'Yellow', 'Magenta', 'Cyan', 'Darkgray'

foreach ($temp in $colors) {
    write-Host "Rainbow" - ForgroundColor $temp
}
```

Do while

```
$i = 1

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

```
$Kapital = 1000
$Zinssatz = 0.01
$Jahre = 0

$Endkapital = 1500

do {
    $Kapital = $Kapital * $Zinssatz + $Kapital
    $Jahre++
} while ($Kapital -le $Endkapital)
```

```
"Das Kapital muss $Jahre Jahre angelegt werden"
```

Fehlerbehandlung

`Set-StrictMode -Version latest` dann werden mehr Fehler angezeigt

```
try {
$b = Read-Host "Bitte einen Wert eingeben: "
}
catch{
   "Ui, ein Fehler ist aufgetreten"
   return # Mit return kann man das Scipt direkt abbrechen. Gibt man noch eine Zahl mit, dann kann man einen Fehlercode mitgeben
}
```

Bei Cmdlets muss man die -ErrorAction auf Stop setzen um bei einem Try eine Fehlermeldung zu bekommen

modul 4,5,7

Remoting

```
Enable-PSRemoting

Get-PSSessionConfiguration

#Verbinden mit Remote

Get-Process w* -ComputerName xxx #Zeigt Prozesse eines anderen Rechner an

# Methode um auf anderem Rechner Cmdlets auszuführen
Invoke-Command -Computername xxx -ScriptBlock { Get-Process w* }

# Skript auf einem anderem Rechner ausführen
Invoke-Command -Computername xxx -FilePath C:\Script01.ps1

# Lokale Variablen werden über den Skriptblock nicht übergeben außer mit using
-ScriptBlock { Get-Service | Where-Object { $_.status -eq $using:status} | Select-Object -First 5 }
```

```
# Dauerhafte verbindung zu einem Rechner
$dc1Session = New-PSSession -Computername xxx

Invoke-Command -Session $dc1Session -Skriptblock { .. }
```

Invoke-Command Info

Kann nur ausgeführt werden wenn WinRM läuft

```
Get-Service -ComputerName xxx -Name WinRM # Nachschauen ob WinRm in Running ist

# WinRM starten und Automatisch beim h
Set-Service -Name WinRM -CumputerName xxx -StartupType Automatic -Status Running
```

Funktionen

Speichern unter als ps1 Datei oder besser .psm1

```
# Modul erstellen
function Write-Hello {
    $Get-Corp
    return
}

# Modul öffnen
Write-Hello

# Module Anzeigen
Get-Module -Listavailable -Name XXX -Refresh
```

Pfad zu Dokumenten:

`$env:PSModulpath`

Hintergrund Aktionen

```
Start-Job -name MyFirstJob -ScriptBlock { Get-ChildItem 'C:\Program Filse\' -Recurse}

# Aufrufen des Jobs:
Get-Job

# Ausgabe des Jobs:
Receive-Job -Job (Get-Job -Id 2) #Wird neu ein einziges mal angezeigt. Besser speichern

# Man kann auch den Parameter -asjob verwenden
-asjob

# Warten bis alle Jobs erledigt sind:
Get-Job | Wait-Job
```

Automatische Jobs

```
$jobtrigger = New-JobTrigger -At (get-ate).AddMinutes(1) -Once

$joboptions = New-ScheduledJObOpti
```

Udemy

Wiederholung Teil 1/3

Select-Object

```
Get-ComputerInfo |
Select-Object `
CsName, OsName, OsInstallDate, OsLastbootUptime |
Format-Table -AutoSize -Wrap # Macht ggf. Zeilenumbruch
```

Attribute

Alle Attribute auflisten

`Get-Disk | Get-Member  # Zeigt alle Attribute von Get-Disk an`

Mit Select-Object kann ich die Attribute greifen

```
Get-Disk | Select-Object  -Property DiskNumber,BusType,FriendlyName
```

().Count

```
(Get-Content irgendwas.txt).count # Zählt alle Objekte
```

Wert einer Property abrufen

```
Get-ADComputer -Filter * | Select-Object -ExpandProperty Name # Gibt den Wert von Property Name zurück

# Mann kann aber auch über die Methode aufrufen
(Get-ADComputer -Filter*).Name
```

Methoden

> Info: Erkunde mit Get-Member

Eine Methode auf aufrufen

```
(Get-Date).AddDays(8) # Fügt dem Aktuellen Datum 8 Tage hinzu
```

Dem Unterschied zwischen Attribut und Methode kann man schon während der Eingabe sehen

Beispiele

```
Get-Disk | Get-Member

# Alles Anzeigen
Get-Disk | Select-Object *

# Gezielt Attribute Abrufen
Get-Disk |
Select-Object -Property `
DiskNumber,BusType,FriendlyName,NomberOfPartisions,Firm
```

Array

Ein Array erzeugen (Liste)

```
$array = [array]('Peter', 'Margit')
$array[1] # Gibt
```

Revision #3
Created 21 March 2023 10:18:34 by Hermann
Updated 17 August 2023 10:15:19 by Hermann