PowerShell für Einsteiger

Grundlagen und Lernsituationen

Inhaltsübersicht

- Begrüßung/Vorstellung
- Impulsvortrag PowerShell
- PowerShell Grundlagen (Cmdlets, Hilfe zur Selbsthilfe)
- Pipeline, Prozesse und Dienste verwalten
- Benutzer und Gruppen verwalten, Benutzerprofile
- Dateisystem und NTFS-Rechte, Freigaben, Netzlaufwerke
- Netzwerkkonfiguration
- Serveraufgaben, Loganalyse, Webzugriff, Jobs
- Programmierung mit PowerShell (ps1-Skripte, Zugriff auf .NET-Objekte)
- Klassenarbeiten und Lernsituationen mit PowerShell erstellen und vorstellen

Cmdlets in PowerShell

Aufbau und Funktion

Cmdlets vs Function

- In Programmiersprachen (PL) gibt es Funktionen.
  - Was eine Funktion ist, kann sich aber stark unterscheiden
  - Imperative PL (z.B. Java) vs. Funktionale PL (z.B. Haskell)
- In PowerShell gibt es Funktionen und Cmdlets.
- Cmdlets sind aus Anwendersicht identisch mit Funktionen (Functions)
- Aus Entwicklersicht gibt es aber Unterschiede:

Cmdlet vs Function

Cmdlet(s)

- ist eine .NET-Klasse
- wird in C# (oder anderer .NET-Sprache geschrieben)
- ist direkt in binärer Form verfügbar (in einer DLL)
- werden in Modulen gebündelt

Function(s)

- wird in PowerShell Language geschrieben
- kann am Prompt definiert werden
- ist in einer Textdatei verfügbar (nicht binär)
- werden in Skripten gebündelt

Cmdlet vs Function

- Ob ein Befehl Cmdlet oder Function ist, kann mit dem Befehl `Get-Command` leicht herausgefunden werden:

```
PS C:\Users\anr> Get-Command Get-WindowsUpdateLog

CommandType     Name                                               Version
-----------     ----                                               -------
Function        Get-WindowsUpdateLog                               1.0.0.0
```

```
PS C:\Users\anr> Get-Command Get-Content

CommandType     Name                                               Version
-----------     ----                                               -------
Cmdlet          Get-Content                                        7.0.0.0
```

- Hands-on: Was für ein CommandType hat `Get-Command` selbst ?

Functions

- Quelltext einer Funktion kann über den PSDrive *Functions* angezeigt werden.
- Beispiel: Funktion `Get-WindowsUpdateLog`

```
PS C:\Users\anr> Get-Content Function:\Get-WindowsUpdateLog

    [CmdLetBinding(
        SupportsShouldProcess = $true,
        ConfirmImpact = 'High')]
    Param(
```

- [Output abgeschnitten]

Cmdlets

- Befehle in der PowerShell heißen Cmdlets (sprich: Commandlets)
- Cmdlets liefern (meistens) Objekte zurück (keine Strings)
- Cmdlets folgen einer sog. Verb-Noun-Syntax:
  - Verb beschreibt, was mit dem Objekt geschieht
  - Noun beschreibt, mit welchem Objekt etwas geschieht

- Cmdlets werden konventionell in CamelCase notiert (keine Pflicht)
- Beispiel:

```
PS C:\Users\anr> Get-Help
```

Cmdlets Zugriff

- Die Cmdlet Rückgabeobjekte sind wie Objekte der OOP zu verstehen
- Sie haben…
  - Eigenschaften (Properties, Sing. Property)
  - Methoden (Methods, Sing. Method)

- Properties haben häufig Getter und manchmal Setter
- Zugriff auf Property mit Punktnotation:
  - `Objekt.Property`
  - syntaktisch wie C# Property
  - kein `getProperty()` wie in Java

Cmdlet Zugriff im Detail

- Beispiel:
  - SID (Security Identifier) des Benutzers anr anzeigen

```
PS C:\Users\anr> (Get-LocalUser -Name anr).SID.Value
S-1-5-21-2609673462-2318655437-1353779694-1002
```

- `Get-LocalUser -Name anr` liefert ein Objekt vom Typ `Microsoft.PowerShell.Commands.LocalUser`
- `(Get-LocalUser -Name anr).SID` liefert ein Objekt vom Typ `System.Security.Principal.SecurityIdentifier`
- `(Get-LocalUser -Name anr).SID.Value` liefert einen Wert vom Typ `String`

☞ Demo

Parameter von Cmdlets

Typen, Aufbau und Funktion

Cmdlets Parameter

- Cmdlets können Parameter haben (die meisten haben welche)
- Parameter werden mit – (Bindestrich, Hyphen, Dash) eingeleitet
- Beispiel:

```
Out-Host –Paging
```

(entspricht `more` in der cmd.exe)

Papierkorb voll?

```
Clear-RecycleBin -Force
```

- Online-Referenz für alle Cmdlets:

https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/?view=powershell-7.3

Sorten von Parametern

- Es gibt vier Sorten Parameter:

…und ein paar Unterarten und Zwitter

1) Named Parameter
2) Positional Parameter
3) Switch Parameter
4) Common Parameter

Named Parameter

- Parameter hat Namen (mit – davor) und Wert (Key-Value-Prinzip)
- Name muss eindeutig sein (aber nicht unbedingt vollständig)

- Beispiel:

```
PS> Get-ChildItem –Path C:\Users\anr\Documents
```

- Name (Key): Path
- Wert (Value): C:\Users\anr\Documents

☞Named Parameter kommen sehr häufig vor

Named Partial Parameter

- Unvollständiger Parametername heißt *partial parameter*.
- Präfixlänge beliebig, muss aber eindeutig sein

- Beispiele:

```
PS> Get-ChildItem -Path C:\Users\anr\Documents        fully named
      ✓
PS> Get-ChildItem -Pa C:\Users\anr\Documents          partial
      ✓
PS> Get-ChildItem -P C:\Users\anr\Documents           partial
      ✘
```

Named Partial Parameter

- Unvollständiger Parametername heißt partial parameter.
- Präfixlänge beliebig, muss aber eindeutig sein

- Beispiel:

```
PS> Get-ChildItem –P C:\Users\anr\Documents
```

partial (ambiguous)

```
PS C:\Users\anr> Get-ChildItem -P C:\users\anr\Documents\
Get-ChildItem: Parameter cannot be processed because the parameter name 'P' is ambiguous. Possible matches include: -Pat
h -PipelineVariable -LiteralPath.
```

Positional Parameter

- Wie Named Parameter ohne Name (Key), nur mit Wert (Value)
- Kürzer, aber schlechter lesbar

- Beispiel:

```
PS> Copy-Item a b
```

  - Wer ist Source (Path) und wer Destination?

- Best Practice:
  - grundsätzlich eher vermeiden (v.a. im Anfänger-Unterricht)
  - Verwendung nur bei leicht verständlichen Cmdlets, z.B. `Get-Process`
  - Ggf. Verwendung bei „Standardparameter“

Positional Parameter

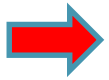
- Beispiel:

```
PS> Get-ChildItem C:\Users\anr\Documents
```

(`-Path` weggelassen)

-Path

Specifies a path to one or more locations. Wildcards are accepted. The default location is the current directory (`.`).



Type:	String[]
Position:	0
Default value:	Current directory
Accept pipeline input:	True
Accept wildcard characters:	True

☞ Die erwartete Position des Parameters steht in der Referenz!

Switch Parameter

- Entweder an- oder ausgeschaltet (engl. Switch ⇨ dt. Schalter)
- Kein Wert, Aktivierung bei Nennung

- Beispiel:

```
PS> Get-ChildItem -Path C:\users\anr\Documents -Name
```

- Aktiviert: zeigt nur Namen der Objekte im Result Set an
- Nicht genannt: weitere Informationen werden angezeigt

☞ Ein häufiger Switch Parameter ist `-Force`

Eigenschaften von Parametern

- Parameter können grundsätzlich außerdem…
  - Standardwerte haben — None (keiner); Default Value wie angegeben
  - Pipeline-Input verarbeiten — True / False
  - Wildcards akzeptieren — True / False
- In der Referenz bzw. Hilfe steht auch der Datentyp (String, UInt32, etc.)

Common Parameter

- Für jedes Cmdlet verfügbar
- Hauptsächlich für Debugging oder Logging gedacht

- Beispiele:

```
-ErrorAction: Break | Ignore | SilentlyContinue …
-Verbose
```

Risikominderungsparameter

- *Risk mitigation parameters*: `-WhatIf`, `-Confirm`
  - Für viele Cmdlets verfügbar
  - Nützlich bei Systemänderungen, Syntaxprüfung etc.
  - Sind eigentlich auch Switch Parameter

- `-WhatIf`: Zeigt (als Text) an, was bei Ausführung passieren wird
- `-Confirm`: Fordert explizit Bestätigung per Keystroke an
- `-Confirm:$False` überschreibt explizit Abfrage nach Bestätigung
  - Nützlich, wenn ein Cmdlet `-Force` nicht implementiert, aber etwas automatisiert werden soll.

Risikominderungsparameter

- Beispiel:

Nutze risk mitigation parameter `-WhatIf`, bevor Benutzer gelöscht wird.

```
PS C:\Users\anr> Remove-LocalUser anr -WhatIf
What if: Performing the operation "Lokalen Benutzer entfernen" on target "anr".
PS C:\Users\anr>
```

☞ Welche Sorte Parameter ist *anr* im obigen Befehl?

(Selbst-)Hilfe in PowerShell

Built-in Ressourcen nutzen

Hilfe zur Selbsthilfe

- "Gib einem Hungernden einen Fisch, und er wird einmal satt, lehre ihn Fischen, und er wird nie wieder hungern." - nach Laotse

- Wenn ich nicht mehr weiter weiß…
- ~~…gründ‘ ich einen Arbeitskreis!~~
- …helfen mir freundliche Cmdlets gerne weiter:
  - `Get-Help`
  - `Get-Command`
  - `Get-Member`

Cmdlets zur Selbsthilfe

- `Get-Help` zeigt Hilfe zu einem (bekannten) Cmdlet an:
  - Struktur
  - Verfügbare Parameter

- Mit `-example` werden Beispiele angezeigt
- Mit `-full` wird die detaillierte Hilfe angezeigt

- Beispiel:

```
PS> Get-Help Get-ChildItem -example
```

Cmdlets zur Selbsthilfe

- `Get-Command` sucht nach (noch unbekanntem) Cmdlet
  - `-Verb` gibt die Aktion an, die das Cmdlet können soll
  - `-Noun` gibt das Objekt (die Familie) des Cmdlets an

- Parameter können einzeln oder zusammen verwendet werden
- Wildcards sind bei beiden erlaubt

Verwendung von `Get-Command`

- Beispiel:
  - Zeige alle Cmdlets zum Verwalten lokaler Benutzer

```
PS> Get-Command –Noun LocalUser
```

```
PS C:\Users\anr> Get-Command -Noun LocalUser

CommandType     Name
-----------     ----
Cmdlet          Disable-LocalUser
Cmdlet          Enable-LocalUser
Cmdlet          Get-LocalUser
Cmdlet          New-LocalUser
Cmdlet          Remove-LocalUser
Cmdlet          Rename-LocalUser
Cmdlet          Set-LocalUser
```

Verwendung von `Get-Command`

- Beispiel:
  - Zeige alle Cmdlets, die etwas mit Benutzern zu tun haben

```
PS> Get-Command -Noun *User*
```

```
PS C:\Users\anr> Get-Command -Noun *User*

CommandType     Name
-----------     ----
Function        Set-PcsvDeviceUserPassword
Cmdlet          Disable-LocalUser
Cmdlet          Enable-LocalUser
Cmdlet          Get-LocalUser
Cmdlet          Get-WinUserLanguageList
Cmdlet          New-LocalUser
Cmdlet          New-WinUserLanguageList
Cmdlet          Remove-LocalUser
Cmdlet          Rename-LocalUser
Cmdlet          Set-LocalUser
Cmdlet          Set-WinUserLanguageList
```

Übung PS21
Selbsthilfe PowerShell

- Die Cmdlets `Get-Help` und `Get-Command` kennenlernen
- Einfache Cmdlets ohne Parameter ausführen
- Cmdlets mit Named Parameter verwenden (Syntaxgewöhnung)