Sprache

Die Sprache ist das Fundament von PowerShell. Variablen, Operatoren, Strings und andere Sprachelemente bestimmen, was dein Code tut und warum er manchmal etwas völlig anderes tut als erwartet. Wenn du verstehen willst, wie PowerShell denkt, beginnt die Reise hier.

Escape-Sequenzen

Escape-Sequenzen sind spezielle Zeichenkombinationen, mit denen sich **Steuerzeichen** innerhalb von Strings darstellen lassen, z. B. Zeilenumbrüche oder Tabs.

In PowerShell wird dafür das **Backtick-Zeichen** verwendet:

```
`
```

Grundlagen

Escape-Sequenzen funktionieren **nur in doppelt-quoted Strings**:

```
"Text`nNeue Zeile"   # funktioniert
'Text`nNeue Zeile'   # kein Effekt
```

Wichtige Escape-Sequenzen

Sequenz	Bedeutung	Beschreibung
`` `n ``	New Line	Zeilenumbruch
`` `r ``	Carriage Return	Cursor an den Anfang der Zeile
`` `r`n ``	Windows-Zeilenumbruch	Kombination aus CR + LF
`` `t ``	Tab	Tabulator
`` `` ```	Backtick	Gibt ein Backtick-Zeichen aus
`` `" ``	Anführungszeichen	Doppelte Quotes innerhalb von Strings

Beispiele

Zeilenumbruch

```
"Hallo`nWelt"
```

Ausgabe:

```
Hallo
Welt
```

Tabulator

```
"Name`tAlter"
```

Ausgabe:

```
Name    Alter
```

Windows-Zeilenumbruch

```
"Hallo`r`nWelt"
```

Verhalten in Dateien

Beim Arbeiten mit Dateien (z. B. `Out-File`, `Set-Content`) wird häufig `` `r`n `` verwendet, da dies dem Windows-Standard entspricht.

Typische Stolperfallen

1. Falsche Anführungszeichen

```
'Hallo`nWelt'   # □ kein Umbruch
```

2. Backtick übersehen

Das Escape-Zeichen ist **kein Apostroph (`'`)**, sondern:

```
`
```

3. Unsichtbare Zeichen

Escape-Sequenzen sind nicht sichtbar im Code, wirken aber auf die Ausgabe. Das macht Debugging manchmal… sagen wir… charakterbildend.

Alternative Methoden

.Split() (kein Escape, aber oft verwandt)

```
"text1,text2" -split ","
```

Oder:

```
"text1,text2".Split(",")
```

`.Split()` ist eine **.NET String-Methode**, kein Escape-Mechanismus, wird aber oft im gleichen Kontext verwendet, wenn Strings verarbeitet werden.

Fazit

Escape-Sequenzen sind ein einfacher Weg, um Strings zu formatieren, ohne sie über mehrere Zeilen schreiben zu müssen.

Das wichtigste:

- Backtick = Escape

- Nur in `" "` aktiv
- `` `n `` für schnellen Zeilenumbruch
- `` `r`n `` für Windows-Kompatibilität

Enum-Werte

Enum-Werte bedeuten vordefinierte mögliche Werte

`[System.Environment+SpecialFolder]`

Wert	Bedeutung
`Desktop`	**Desktop** – Der Desktop vom aktuellen Benutzer
`MyComputer`	**Dieser PC** – beinhaltet Datenträger und Laufwerke
`Programs`	**Startmenü Programme** – Im Startmenü angezeigten Programme

MyDocuments
Personal
Favorites
Startup
Recent
SendTo
StartMenu
MyMusic
MyVideos
DesktopDirectory
NetworkShortcuts
Fonts
Templates
CommonStartMenu
CommonPrograms
CommonStartup
CommonDesktopDirectory
ApplicationData
PrinterShortcuts
LocalApplicationData
InternetCache
Cookies
History
CommonApplicationData
Windows

System
ProgramFiles
MyPictures
UserProfile
SystemX86
ProgramFilesX86
CommonProgramFiles
CommonProgramFilesX86
CommonTemplates
CommonDocuments
CommonAdminTools
AdminTools
CommonMusic
CommonPictures
CommonVideos
Resources
LocalizedResources
CommonOemLinks
CDBurning

```
x   (x)
```

PowerShell Variablen

Variablen die von PowerShell je nach Kontext, selbst deklariert werden.

Variable	Beschreibung
`$PSCommandPath`	Vollständiger Pfad zum aktuellen Skript
`$PSModulePath`	Beinhaltet die Pfade zu den Modulen
`$PSScriptRoot`	Der Pfad zum Ordner worin sich das aktuelle Skript befindet
`$MyInvocation`	Enthält Infos darüber, wie das Skript aufgerufen wurde

`$PSScriptRoot`

- **Datentyp:** `System.String`

`$PSCommandPath`

- **Datentyp:** `System.String`

`$PSScriptRoot`

- **Datentyp:**

`$MyInvocation`

- **Datentyp:** `InvocationInfo`

Enthält Infos darüber, wie das Skript aufgerufen wurde

- `$MyInvocation.MyCommand` – Informationen und die Idendität des aktuellen Kommando-Objekts
  - `$MyInvocation.MyCommand.CommandType`
  - `$MyInvocation.MyCommand.Path` – Enthält den Pfad zum aktuellen Skript

Operatoren

Operatoren sind spezielle Sprachelemente in PowerShell, die Werte verarbeiten, vergleichen, verknüpfen oder verändern. Sie bilden einen zentralen Bestandteil der Sprache und werden in nahezu jedem Skript verwendet.

Je nach Typ übernehmen Operatoren unterschiedliche Aufgaben:

- Werte vergleichen (`-eq`, `-ne`, `-gt`, ...)
- Logische Bedingungen verknüpfen (`-and`, `-or`, `-not`)
- Inhalte zusammenführen oder aufteilen (`-join`, `-split`)
- Berechnungen durchführen (`+`, `-`, `*`, `/`)

String- und Array-Operatoren

Diese Operatoren werden häufig verwendet, um Texte und Listen zu verarbeiten.

Split

Der Operator `-split` zerlegt einen String anhand eines Trennzeichens in mehrere Elemente eines Arrays.

Beispiel:

```
"Max;Mustermann;30" -split ";"
```

Ergebnis:

```
Max
Mustermann
30
```

Weitere Informationen findest du auf der Seite **Split**.

Join

Der Operator `-join` führt mehrere Elemente eines Arrays zu einem einzelnen String zusammen.

Beispiel:

```
"Max","Mustermann",30 -join ";"
```

Ergebnis:

```
Max;Mustermann;30
```

Weitere Informationen findest du auf der Seite **Join**.

Merksatz

`-split` macht aus einem String ein Array.

`-join` macht aus einem Array einen String.

Beide Operatoren werden häufig gemeinsam verwendet, wenn Daten eingelesen, verarbeitet und anschließend wieder ausgegeben werden.

Join

Der `-join` Operator in PowerShell wird verwendet, um mehrere Elemente (z. B. Strings in einem Array) zu einem einzigen String zusammenzufügen.

□□ Syntax

```
<array> -join <delimiter>
```

Oder ohne Trennzeichen:

```
<array> -join
```

□□ Grundlagen

- Gibt immer einen **String (`System.String`)** zurück
- Verbindet alle Elemente eines Arrays **in der gegebenen Reihenfolge**
- Der Delimiter wird **wörtlich verwendet** (kein Regex!)
- Standard-Delimiter ist ein **leerer String** (`""`)

□□ Beispiele

Einfaches Zusammenfügen

```
$array = "Apfel","Birne","Banane"
$result = $array -join ","
```

Ergebnis:

```
Apfel,Birne,Banane
```

Join ohne Trennzeichen

```
$array = "Hallo","Welt"
$result = $array -join
```

Ergebnis:

```
HalloWelt
```

Join mit Leerzeichen

```
$array = "Das","ist","ein","Test"
$result = $array -join " "
```

Ergebnis:

```
Das ist ein Test
```

Join mit Zeilenumbruch

```
$array = "Zeile1","Zeile2","Zeile3"
$result = $array -join "`n"
```

Ergebnis:

```
Zeile1
Zeile2
Zeile3
```

⚠ Wichtige Hinweise

1. Kein Regex

Im Gegensatz zu `-split`:

Der Delimiter wird **nicht** als regulärer Ausdruck interpretiert.

Das bedeutet:

- Sonderzeichen haben **keine spezielle Bedeutung**
- `"."` ist einfach ein Punkt, kein Platzhalter

---

2. `$null`-Werte

```
$array = "A",$null,"B"
$result = $array -join ","
```

Ergebnis:

```
A,,B
```

Erklärung:

- `$null` wird als **leerer String** behandelt
- Der Delimiter wird trotzdem eingefügt

---

3. Nicht-String-Werte

```
$array = 1,2,3
$result = $array -join "-"
```

Ergebnis:

```
1-2-3
```

Erklärung:

- Alle Elemente werden **automatisch in Strings konvertiert**

□□ Alternative Methoden

`[string]::Join()`

```
[string]::Join(",", $array)
```

Unterschied zu `-join`:

- Methodenaufruf statt Operator
- Gleiche Funktionalität, oft in .NET-Kontexten verwendet

□□ Typische Anwendungsfälle

- Array in einen String umwandeln
- CSV-Zeilen erzeugen
- Textausgaben formatieren
- Mehrzeilige Strings erzeugen

□□ Mini-Beispiel aus der Praxis

```
$user = "Max","Mustermann",30
$csv = $user -join ";"
```

Ergebnis:

```
Max;Mustermann;30
```

□□ Merksatz

> “ `-join` klebt alles zusammen. Ohne Struktur, wenn du keinen Delimiter setzt.

Split

Der `-split` Operator in PowerShell wird verwendet, um einen String anhand eines Trennzeichens (Delimiter) in ein Array von Teilstrings aufzuteilen.

□□ Syntax

```
<string> -split <delimiter>
```

Oder mit erweiterten Optionen:

```
<string> -split <delimiter>, <max-substrings>, <options>
```

□□ Grundlagen

- Gibt immer ein **Array (`System.String[]`)** zurück
- Der **Delimiter ist ein regulärer Ausdruck (RegEx!)**, kein einfacher Text
- Groß-/Kleinschreibung kann beeinflusst werden

□□ Beispiele

Einfaches Splitten

```
$text = "Apfel,Birne,Banane"
$result = $text -split ","
```

Ergebnis:

```
Apfel
Birne
Banane
```

Split mit Leerzeichen

```
$text = "Das ist ein Test"
$result = $text -split " "
```

Mehrere Trennzeichen (Regex)

```
$text = "Apfel;Birne,Banane"
$result = $text -split "[,;]"
```

Erklärung:

- `[ , ; ]` bedeutet: splitte bei Komma **oder** Semikolon

□□ Verhalten bei `max-substrings`

Wenn die maximale Anzahl an Elementen erreicht ist:

> **Das letzte Element im Ergebnis enthält den gesamten verbleibenden Rest des Strings und wird nicht weiter gesplittet.**

```
$text = "A,B,C,D"
$result = $text -split ",", 2
```

Ergebnis:

```
A
B,C,D
```

Erklärung:

- Es werden maximal **2 Elemente** erzeugt
- Das erste Element entsteht durch den ersten Split
- Das zweite Element enthält **alles, was danach noch übrig ist** (`B,C,D`)

□□Split mit Optionen

```
$text = "a,b,c"
$result = $text -split ",", 0, "IgnoreCase"
```

Mögliche Optionen:

- `IgnoreCase`
  → Groß- und Kleinschreibung wird ignoriert
  → `"A,B,C" -split "a"` würde trotzdem funktionieren
- `CaseSensitive`
  → Groß- und Kleinschreibung wird beachtet
  → `"A,B,C" -split "a"` liefert **kein Ergebnis**, da kein Match
- `SimpleMatch`
  → Der Delimiter wird **als normaler Text behandelt, nicht als Regex**
  → Sonderzeichen wie `.`, `*`, `+` verlieren ihre Regex-Bedeutung
  Beispiel:

  ```
  "1.2.3" -split ".", 0, "SimpleMatch"
  ```

  → funktioniert ohne Escape (`\.`)

⚠ Wichtige Hinweise

1. Regex-Falle

```
$text = "1.2.3"
$result = $text -split "."
```

Problem:
`.` bedeutet im Regex „beliebiges Zeichen“

➡ Ergebnis: komplett zerlegt

Lösung:

```
$result = $text -split "\."
```

□□ 2. Leere Elemente

```
"text,,text" -split ","
```

Ergebnis:

```
text


text
```

Erklärung:

- Jeder Teilstring entsteht **zwischen zwei Trennzeichen (Delimiter)**
- Wenn zwei Delimiter direkt aufeinander folgen (`,,`), liegt dazwischen **kein Inhalt**
- PowerShell erzeugt dafür trotzdem ein Element im Array

> Dieses Element ist ein **leerer String (`""`)**, kein `null`

□□ Alternative Methoden

`.Split()` (kein Regex!)

```
$text = "A,B,C"
$result = $text.Split(",")
```

Unterschied zu `-split`:

- `.Split()` verwendet **kein Regex**
- oft schneller und einfacher
- weniger flexibel

`[regex]::Split()`

```
[regex]::Split("A,B,C", ",")
```

➡ Alternative mit explizitem Regex-Handling

□□ Typische Anwendungsfälle

- CSV-Daten zerlegen
- Log-Dateien parsen
- Benutzereingaben aufteilen
- Pfade oder Listen verarbeiten

□□ Mini-Beispiel aus der Praxis

```
$csv = "Max;Mustermann;30"
$name = $csv -split ";"

$vorname = $name[0]
$nachname = $name[1]
$alter = $name[2]
```

□□ Merksatz

> `-split` denkt in Regex. Wenn du das vergisst, bekommst du Chaos statt Struktur.