PowerShell – Parameter

Grundidee

Parameter sind Zusatzangaben zu einem PowerShell-Befehl.

Mit Parametern legt man fest, **was genau ein Cmdlet tun soll** oder **wie es arbeiten soll**.

Ein PowerShell-Befehl besteht meistens aus:

Bestandteil	Bedeutung
Cmdlet	Der eigentliche PowerShell-Befehl
Parameter	Zusatzangabe, die das Cmdlet genauer steuert
Parameterwert	Der konkrete Wert, der an den Parameter übergeben wird

Beispiel:

```
Get-Process -Name "notepad"
```

Bedeutung:

Teil	Erklärung
Get-Process	Cmdlet
-Name	Parameter
"notepad"	Parameterwert

PowerShell zeigt dadurch nicht alle Prozesse an, sondern nur den Prozess mit dem Namen **notepad**.

Merksatz

Cmdlet = Was soll passieren?
Parameter = Wie genau soll es passieren?
Parameterwert = Womit oder worauf soll es passieren?

Typischer Aufbau

Ein PowerShell-Befehl mit Parameter hat meistens diesen Aufbau:

```
Cmdlet -Parameter Wert
```

Beispiel:

```
Get-Service -Name "Spooler"
```

Bedeutung:

Teil	Erklärung
Get-Service	Dienste anzeigen
-Name	Nach einem bestimmten Dienstnamen suchen
"Spooler"	Druckwarteschlange als konkreter Dienst

Warum Parameter wichtig sind

Ohne Parameter führt ein Cmdlet oft eine allgemeine Aktion aus.

Beispiel:

```
Get-Service
```

Dieser Befehl zeigt viele Dienste an.

Mit Parameter wird die Ausgabe gezielter:

```
Get-Service -Name "Spooler"
```

Jetzt wird nur der Dienst **Spooler** angezeigt.

Parameter helfen also dabei, Befehle genauer, sicherer und gezielter auszuführen.

Parameter mit Wert

Viele Parameter benötigen einen Wert.

Beispiel:

```
Get-ChildItem -Path "C:\Temp"
```

Hier braucht der Parameter **-Path** einen konkreten Pfad.

Weitere Beispiele:

```
Get-Process -Name "notepad"

Get-Service -Name "Spooler"

Get-ChildItem -Filter "*.txt"
```

Typische Parameter mit Wert:

Parameter	Bedeutung
-Name	Name eines Objekts
-Path	Pfad zu Datei oder Ordner
-Filter	Filter für Dateien oder Objekte
-ComputerName	Zielcomputer
-Credential	Zugangsdaten
-Destination	Zielpfad
-Source	Quellpfad

Schalterparameter

Ein Schalterparameter braucht keinen eigenen Wert.

Er ist aktiv, sobald man ihn angibt.

Beispiel:

```
Get-ChildItem -Recurse
```

Der Parameter **-Recurse** bedeutet, dass auch Unterordner durchsucht werden.

Weitere Beispiele:

```
Remove-Item "C:\Temp\test.txt" -Force

Get-ChildItem -Hidden
```

```
Stop-Process -Name "notepad" -Force
```

Typische Schalterparameter:

Parameter	Bedeutung
-Recurse	Unterordner einbeziehen
-Force	Aktion erzwingen
-Hidden	Versteckte Elemente anzeigen
-WhatIf	Zeigt, was passieren würde, ohne es auszuführen
-Confirm	Fragt vor der Ausführung nach Bestätigung

Wichtige Sicherheitsparameter

Besonders wichtig sind **-WhatIf** und **-Confirm**.

Diese Parameter helfen, gefährliche Aktionen zu prüfen.

Beispiel:

```
Remove-Item "C:\Temp\test.txt" -WhatIf
```

PowerShell zeigt nur an, was gelöscht werden würde.
Die Datei wird nicht wirklich gelöscht.

Beispiel mit Bestätigung:

```
Remove-Item "C:\Temp\test.txt" -Confirm
```

PowerShell fragt vor dem Löschen nach.

Das ist besonders wichtig bei Befehlen wie:

Cmdlet	Risiko
Remove-Item	Dateien oder Ordner löschen
Stop-Process	Prozesse beenden
Restart-Computer	Computer neu starten
Set-Item	Werte ändern

Cmdlet	Risiko
New-Item	Neue Dateien, Ordner oder Einträge erstellen

Positionelle Parameter

Manche Parameter müssen nicht ausgeschrieben werden, weil PowerShell anhand der Position erkennt, welcher Parameter gemeint ist.

Beispiel mit ausgeschriebenem Parameter:

```
Get-ChildItem -Path "C:\Temp"
```

Kurzform:

```
Get-ChildItem "C:\Temp"
```

PowerShell erkennt hier, dass **"C:\Temp"** der Wert für **-Path** ist.

Für die Prüfung und für saubere Dokumentation ist die ausgeschriebene Variante besser:

```
Get-ChildItem -Path "C:\Temp"
```

Sie ist klarer und leichter nachvollziehbar.

Mehrere Parameter kombinieren

Ein Cmdlet kann mehrere Parameter gleichzeitig verwenden.

Beispiel:

```
Get-ChildItem -Path "C:\Temp" -Filter "*.txt" -Recurse
```

Bedeutung:

Teil	Erklärung
Get-ChildItem	Ordnerinhalt anzeigen
-Path "C:\Temp"	Im Ordner C:\Temp suchen
-Filter "*.txt"	Nur Textdateien anzeigen

Teil	Erklärung
-Recurse	Auch Unterordner durchsuchen

Der Befehl sucht also im Ordner **C:\Temp** und allen Unterordnern nach Dateien mit der Endung **.txt**.

Parameter und Datentypen

Parameter erwarten oft bestimmte Datentypen.

Beispiele:

Datentyp	Beispiel	Bedeutung
String	"notepad"	Text
Integer	10	Ganze Zahl
Boolean	$true / $false	Wahr oder falsch
Array	"PC1","PC2","PC3"	Mehrere Werte
Credential	Benutzername/Kennwort	Zugangsdaten
Switch	-Force	Schalterparameter ohne Wert

Beispiel mit mehreren Namen:

```
Get-Service -Name "Spooler","WinRM"
```

Hier bekommt der Parameter **-Name** mehrere Werte.

Parameter und Anführungszeichen

Texte mit Leerzeichen sollten in Anführungszeichen geschrieben werden.

Beispiel:

```
Get-ChildItem -Path "C:\Meine Dateien"
```

Ohne Anführungszeichen könnte PowerShell den Pfad falsch interpretieren.

Bei einfachen Werten ohne Leerzeichen sind Anführungszeichen oft nicht zwingend nötig:

```
Get-Service -Name Spooler
```

Trotzdem ist es beim Lernen oft übersichtlicher, Zeichenketten mit Anführungszeichen zu schreiben.

Parameter mit Wildcards

Viele Parameter unterstützen Platzhalterzeichen.

Das wichtigste Platzhalterzeichen ist:

Zeichen	Bedeutung
*	Steht für beliebig viele Zeichen
?	Steht für genau ein Zeichen

Beispiel:

```
Get-Service -Name "Win*"
```

Zeigt Dienste an, deren Name mit **Win** beginnt.

Beispiel:

```
Get-ChildItem -Path "C:\Temp" -Filter "*.txt"
```

Zeigt alle Dateien mit der Endung **.txt** an.

Pflichtparameter

Manche Parameter sind erforderlich.

Wenn ein Pflichtparameter fehlt, fragt PowerShell nach dem fehlenden Wert.

Beispiel:

```
Read-Host -Prompt
```

PowerShell erwartet hier einen Wert für **-Prompt**.

Besser:

```
Read-Host -Prompt "Bitte Namen eingeben"
```

Optionale Parameter

Viele Parameter sind optional.

Beispiel:

```
Get-Process
```

Dieser Befehl funktioniert ohne weitere Parameter.

Mit optionalem Parameter wird er genauer:

```
Get-Process -Name "notepad"
```

Der Parameter **-Name** ist hier nicht zwingend notwendig, aber nützlich.

Parameter und Pipeline

PowerShell arbeitet objektorientiert.
Die Pipeline übergibt nicht nur Text, sondern Objekte von einem Befehl zum nächsten.

Beispiel:

```
Get-Process -Name "notepad" | Stop-Process
```

Bedeutung:

Teil	Erklärung
Get-Process -Name "notepad"	Sucht den Prozess notepad
Pipe-Zeichen	Übergibt das gefundene Objekt weiter
Stop-Process	Beendet den übergebenen Prozess

Noch deutlicher:

```
Get-Process -Name "notepad" | Stop-Process -Force
```

Hier beendet PowerShell den Prozess **notepad** erzwungen.

Pipeline-Parameter

Einige Parameter können Werte aus der Pipeline annehmen.

Beispiel:

```
Get-Service -Name "Spooler" | Stop-Service
```

Der Dienst wird zuerst gesucht und dann an **Stop-Service** weitergegeben.

PowerShell erkennt anhand des Objekttyps, welcher Parameter des nächsten Cmdlets den Wert annehmen kann.

Das nennt man auch:

Begriff	Bedeutung
Pipeline Input	Eingabe aus der Pipeline
ByValue	Übergabe anhand des Objekttyps
ByPropertyName	Übergabe anhand eines passenden Eigenschaftsnamens

Für die IHK reicht meistens das Grundverständnis:

Die Pipeline übergibt das Ergebnis eines Befehls an den nächsten Befehl. Parameter können diese übergebenen Werte weiterverarbeiten.

Common Parameters

Viele PowerShell-Cmdlets besitzen gemeinsame Standardparameter.

Diese nennt man **Common Parameters**.

Wichtige Beispiele:

Parameter	Bedeutung
-Verbose	Gibt ausführlichere Informationen aus
-Debug	Gibt Debug-Informationen aus
-ErrorAction	Legt fest, wie Fehler behandelt werden
-ErrorVariable	Speichert Fehler in einer Variable
-WarningAction	Legt fest, wie Warnungen behandelt werden
-OutVariable	Speichert die Ausgabe in einer Variable
-PipelineVariable	Speichert Pipeline-Objekte in einer Variable

Beispiel:

```
Get-Service -Name "Spooler" -Verbose
```

Mit **-Verbose** können zusätzliche Informationen angezeigt werden.

Fehlerbehandlung mit Parametern

Ein wichtiger Parameter ist **-ErrorAction**.

Damit legt man fest, wie PowerShell bei Fehlern reagieren soll.

Beispiel:

```
Get-Item -Path "C:\NichtVorhanden" -ErrorAction SilentlyContinue
```

Bedeutung:

PowerShell gibt keinen sichtbaren Fehler aus, wenn der Pfad nicht existiert.

Häufige Werte für **-ErrorAction**:

Wert	Bedeutung
Continue	Fehler anzeigen und weitermachen
Stop	Fehler anzeigen und Ausführung stoppen
SilentlyContinue	Fehler nicht anzeigen und weitermachen
Inquire	Nachfrage anzeigen
Ignore	Fehler ignorieren

Parameter herausfinden

Man muss nicht alle Parameter auswendig kennen.

PowerShell bietet Hilfe-Befehle.

Wichtige Befehle:

```
Get-Help Get-ChildItem
```

Zeigt Hilfe zum Cmdlet an.

```
Get-Help Get-ChildItem -Detailed
```

Zeigt ausführlichere Informationen.

```
Get-Help Get-ChildItem -Examples
```

Zeigt Beispiele.

```
Get-Command Get-ChildItem -Syntax
```

Zeigt die Syntax des Cmdlets.

```
Get-Command Get-ChildItem | Select-Object -ExpandProperty Parameters
```

Zeigt die verfügbaren Parameter eines Cmdlets.

Syntax in der Hilfe verstehen

Bei PowerShell-Hilfeausgaben sieht man oft eckige Klammern.

Beispiel:

```
Get-ChildItem [[-Path] <string[]>] [-Filter <string>] [-Recurse]
```

Bedeutung:

Schreibweise	Bedeutung
[-Path]	Parameter ist optional
<string[]>	Erwartet einen oder mehrere Textwerte
[-Filter]	Optionaler Parameter mit Textwert
[-Recurse]	Optionaler Schalterparameter

Wichtig:

Eckige Klammern bedeuten in der Syntax meistens: **optional**.

Cmdlet-Namen und Parameter

PowerShell-Cmdlets folgen meistens dem Verb-Nomen-Schema.

Beispiele:

Cmdlet	Bedeutung
Get-Process	Prozesse abrufen
Stop-Process	Prozess beenden
Get-Service	Dienste abrufen
Start-Service	Dienst starten
Stop-Service	Dienst stoppen
Get-ChildItem	Ordnerinhalt anzeigen
Remove-Item	Datei/Ordner löschen
New-Item	Datei/Ordner erstellen
Copy-Item	Datei/Ordner kopieren
Move-Item	Datei/Ordner verschieben

Parameter passen sich an das jeweilige Cmdlet an.

Beispiel:

```
Get-Service -Name "Spooler"

Stop-Service -Name "Spooler"
```

Beide Befehle nutzen **-Name**, aber mit unterschiedlicher Aktion.

Aliase und Parameter

PowerShell besitzt auch Kurzformen, sogenannte Aliase.

Beispiele:

Alias	Eigentliches Cmdlet
dir	Get-ChildItem
ls	Get-ChildItem
cd	Set-Location
copy	Copy-Item
del	Remove-Item

Alias	Eigentliches Cmdlet
cat	Get-Content

Beispiel:

```
dir "C:\Temp"
```

Das entspricht ungefähr:

```
Get-ChildItem -Path "C:\Temp"
```

Für die IHK und Dokumentation ist das ausgeschriebene Cmdlet besser, weil es eindeutiger ist.

Parameter in eigenen Skripten

Auch eigene PowerShell-Skripte können Parameter besitzen.

Beispiel:

```
param(
    [string]$Name
)

Write-Output "Hallo $Name"
```

Wenn das Skript zum Beispiel **gruss.ps1** heißt, kann man es so starten:

```
.\gruss.ps1 -Name "Felix"
```

Ausgabe:

```
Hallo Felix
```

Mehrere eigene Parameter

Beispiel:

```
param(
    [string]$Name,
    [int]$Alter
)

Write-Output "$Name ist $Alter Jahre alt."
```

Aufruf:

```
.\person.ps1 -Name "Felix" -Alter 30
```

Hier sind:

Teil	Bedeutung
-Name	Parameter für den Namen
"Felix"	Wert für den Namen
-Alter	Parameter für das Alter
30	Wert für das Alter

Pflichtparameter in eigenen Skripten

Man kann Parameter als verpflichtend kennzeichnen.

Beispiel:

```
param(
    [Parameter(Mandatory=$true)]
    [string]$Name
)

Write-Output "Hallo $Name"
```

Wenn der Parameter **-Name** beim Start fehlt, fragt PowerShell nach.

Parameter mit Standardwert

Parameter können Standardwerte besitzen.

Beispiel:

```
param(
    [string]$Name = "Benutzer"
)

Write-Output "Hallo $Name"
```

Wenn kein Name angegeben wird, verwendet PowerShell automatisch **Benutzer**.

Parameter validieren

PowerShell kann Parameterwerte prüfen.

Beispiel mit erlaubten Werten:

```
param(
    [ValidateSet("Start","Stop","Restart")]
    [string]$Aktion
)

Write-Output "Gewählte Aktion: $Aktion"
```

Jetzt darf der Parameter **-Aktion** nur bestimmte Werte annehmen:

Erlaubter Wert
Start
Stop
Restart

Beispiel:

```
.\dienst.ps1 -Aktion Start
```

Ein falscher Wert würde abgelehnt werden.

Typische IHK-relevante Beispiele

Dienst anzeigen

```
Get-Service -Name "Spooler"
```

Zeigt den Dienst **Spooler** an.

Dienst starten

```
Start-Service -Name "Spooler"
```

Startet den Dienst **Spooler**.

Dienst stoppen

```
Stop-Service -Name "Spooler"
```

Stoppt den Dienst **Spooler**.

Prozess anzeigen

```
Get-Process -Name "notepad"
```

Zeigt den Prozess **notepad** an.

Prozess beenden

```
Stop-Process -Name "notepad"
```

Beendet den Prozess **notepad**.

Dateien in einem Ordner anzeigen

```
Get-ChildItem -Path "C:\Temp"
```

Zeigt den Inhalt des Ordners **C:\Temp** an.

Dateien rekursiv suchen

```
Get-ChildItem -Path "C:\Temp" -Filter "*.log" -Recurse
```

Sucht im Ordner **C:\Temp** und allen Unterordnern nach **.log-Dateien**.

Datei löschen mit Sicherheitsprüfung

```
Remove-Item -Path "C:\Temp\test.txt" -WhatIf
```

Zeigt an, was gelöscht werden würde, löscht aber nicht wirklich.

Datei löschen mit Bestätigung

```
Remove-Item -Path "C:\Temp\test.txt" -Confirm
```

Fragt vor dem Löschen nach.

Wichtige Begriffe

Begriff	Erklärung
Cmdlet	PowerShell-Befehl im Verb-Nomen-Schema
Parameter	Zusatzangabe zu einem Cmdlet
Parameterwert	Konkreter Wert zu einem Parameter
Schalterparameter	Parameter ohne eigenen Wert
Pflichtparameter	Muss angegeben werden
Optionaler Parameter	Kann angegeben werden, muss aber nicht
Positioneller Parameter	Kann ohne Parameternamen verwendet werden
Pipeline	Übergibt Objekte von einem Befehl zum nächsten
Common Parameters	Gemeinsame Standardparameter vieler Cmdlets
Alias	Kurzname für ein Cmdlet

Häufige Fehler

Fehler	Erklärung
Parameter falsch geschrieben	PowerShell erkennt den Parameter nicht
Wert fehlt	Ein Parameter erwartet einen Wert

Fehler	Erklärung
Falscher Datentyp	Zum Beispiel Text statt Zahl
Pfad ohne Anführungszeichen	Fehler bei Leerzeichen im Pfad möglich
Zu gefährlicher Befehl ohne -WhatIf	Aktion wird direkt ausgeführt
Alias statt Cmdlet in Dokumentation	Kann unklar oder weniger professionell wirken

---

Gute Praxis

Empfehlung	Grund
Parameter ausschreiben	Bessere Lesbarkeit
Cmdlets statt Aliase verwenden	Eindeutiger und professioneller
Bei gefährlichen Befehlen -WhatIf nutzen	Verhindert versehentliche Änderungen
Pfade mit Leerzeichen in Anführungszeichen setzen	Vermeidet Fehler
Get-Help nutzen	Parameter müssen nicht auswendig gelernt werden
Befehle erst testen	Besonders bei Löschen, Stoppen oder Ändern

---

IHK-Einordnung

Für die IHK ist vor allem wichtig, dass man das Prinzip versteht.

Man muss normalerweise nicht jeden Parameter auswendig kennen.

Wichtig ist:

Prüfungsrelevanter Punkt	Erklärung
Cmdlet erkennen	PowerShell-Befehl verstehen
Parameter erkennen	Zusatzangabe zum Befehl erkennen
Parameterwert erkennen	Konkreten Wert zuordnen
Schalterparameter verstehen	Parameter ohne eigenen Wert
Pipeline verstehen	Ausgabe eines Befehls wird weitergegeben
Sichere Ausführung verstehen	-WhatIf und -Confirm kennen
Hilfe verwenden können	Get-Help und Get-Command kennen

Prüfungsnahes Beispiel

Aufgabe:

Ein Administrator möchte alle Textdateien im Ordner **C:\Temp** und dessen Unterordnern anzeigen lassen.

Passender PowerShell-Befehl:

```
Get-ChildItem -Path "C:\Temp" -Filter "*.txt" -Recurse
```

Erklärung:

Teil	Bedeutung
Get-ChildItem	Ordnerinhalt anzeigen
-Path "C:\Temp"	Startordner festlegen
-Filter "*.txt"	Nur Textdateien anzeigen
-Recurse	Unterordner einbeziehen

Prüfungsnahes Beispiel mit Sicherheit

Aufgabe:

Ein Administrator möchte prüfen, welche Dateien durch einen Löschbefehl betroffen wären, ohne sie wirklich zu löschen.

Passender PowerShell-Befehl:

```
Remove-Item -Path "C:\Temp\*.log" -WhatIf
```

Erklärung:

Teil	Bedeutung
Remove-Item	Elemente löschen
-Path "C:\Temp*.log"	Alle .log-Dateien im Zielpfad
-WhatIf	Nur anzeigen, was passieren würde

Zusammenfassung

Parameter sind ein zentrales Konzept in PowerShell.

Sie steuern, wie ein Cmdlet arbeitet und auf welche Objekte es angewendet wird.

Die wichtigsten Punkte:

Punkt	Kurz erklärt
Parameter steuern Cmdlets	Sie machen Befehle genauer
Manche Parameter brauchen Werte	Zum Beispiel -Name "Spooler"
Schalterparameter brauchen keinen Wert	Zum Beispiel -Recurse oder -Force
Parameter können kombiniert werden	Dadurch entstehen gezielte Befehle
-WhatIf und -Confirm erhöhen die Sicherheit	Besonders bei gefährlichen Aktionen
Get-Help zeigt verfügbare Parameter	Man muss nicht alles auswendig kennen
Die Pipeline übergibt Objekte weiter	Cmdlets können miteinander kombiniert werden

---

Kurzer Merksatz

Ein Parameter ist eine Zusatzangabe zu einem PowerShell-Cmdlet, mit der festgelegt wird, worauf und wie der Befehl angewendet wird.

Beispiel:

```
Get-Service -Name "Spooler"
```

Get-Service ist das Cmdlet.
-Name ist der Parameter.
"Spooler" ist der Parameterwert.
