Interaktiver Modus

Commandlet-Aufbau

`Verb-Noun`	**Verb:** Get, Add, Copy, Set, ... **Noun:** Process, Item, Help, ...

Hilfefunktionen

`Update-Help`	Als Admin: aktualisiert Hilfedateien
`Get-Help <cmd>`	Zeigt Hilfe zu Cmdlet
`Get-Help *Item`	Zeigt alle Cmdlets, die mit Item enden
`Get-Command <cmd>`	Hilfe zu Commands (`-Verb Get` holt alle Get-Cmdlets)
`Get-Help about_*`	Zeigt alle About-Docs (z. B. `about_if`).
`<var> | Get-Member`	Variablen und Commandlets untersuchen
`<Cmd> -WhatIf`	Ausführung emulieren

Aliase

`New-Alias`	Erstellt neuen Alias
`Remove-Alias`	Löscht einen Alias
`Get-Alias -Definition <cmd>`	Zeigt alle Aliase zu einem Commandlet

Standardkanäle

`<a> i> <file>`	Write Stream i von a in $file$
`<a> i>> <file>`	Append Stream i von a in $file$
`<a> i>&j`	Leitet Stream i von a in Str. j

Nr	PowerShell	Linux	Nr	PowerShell
0		`stdin`	4	`verbose`
1	`sucess`	`stdout`	5	`debug`
2	`error`	`stderr`	6	`information`
3	`warning`		*	`all streams`

Dateisystem

`Set-Location`	In Ordner wechseln
`Get-Location`	Aktuellen Ornerpfad holen
`New-Item`	Erstellt Datei oder Ordner
`Copy-Item`	Kopiert Datei oder Ordner
`Move-Item`	Bewegt Datei oder Ordner
`Remove-Item`	Löscht Datei oder Ordner
`Get-Item`	Holt Meta-Informationen eines Items (z. B. Datei) ein
`Set-Item`	Setzt Meta-Informationen
`Get-Content`	Liest Inhalt einer Datei ein
`Get-ChildItem`	Holt ein Item und seine Kinder-Items (Unterordner)
`Tree`	Zeigt Ordner rekursiv in Baumstruktur (nicht PS)

Ausgaben (kleine Auswahl)

`Write-Host`	Erzeugt eine Ausgabe auf `stdout`
`Write-Debug 'msg' -Debug`	Schreibt eine Debugnachricht und aktiviert den Debug-Modus
`Get-Help about_PreferenceVariables`	Hilfe zu $DebugPrefrence etc.

Grundlagen Vergleichsoperatoren

Vergleichsoperatoren können überall da genutzt werden, wo Ausdrücke ausgewertet werden.

`$a -eq 2`	Vergleich auf Gleichheit
`$a -gt 2`	Vergleich auf Größer
`$a -like "W*"`	Vergleich auf Wildcard-Pattern eines Strings

`Get-Help about_Comparison_Operators`

Pipelining

`<a> | <b>`	Leitet `stdout` von a in `stdin` von b
`Where-Object{ }`	Filtert Objekte basierend auf einer Bedingung.
`Select-Object`	Wählt bestimmte Eigenschaften eines Objekts aus.
`Sort-Object`	Sortiert Objekte nach einem bestimmten Kriterium.
`Foreach-Object{ }`	Anweisungen pro Objekt ausführen
`Group-Object`	Gruppiert anhand einer Eigenschaft der Objekte.
`Get-Member`	Metadaten zu Objekt ausgeben
`Measure-Object`	Min, Max, Sum, Avg
`Compare-Object`	Zwei Objektmengen vergleichen
`Format-List`	Ausgabe formatieren (viele Format-Varianten)
`Tee-Object <a> | <b>`	Splittet `stdout` in a und b auf
`Get-Help about_Pipelines`	Hilfeartikel zu Pipelines
`Out-Null`	Unterdrückt Output in einer Pipeline
`Out-Printer`	Sendet Output an Drucker
`Out-File`	Sendet Output an Datei

Skripting

Parameter in Skripten

`./script.ps1 <arg1> [, ..., <argN>]`	
`$args.Count`	Anzahl der Argumente prüfen
`$args.[i]`	**Positionale** Argumente an Stelle i auslesen
`./script.ps1 -par1 <w> [, ..., -parN <w>]`	
`param([typ]$par1, ... [typ]$parN)`	**Benannte** Parameter mit Typ definieren.

Umgang mit Variablen

`[int] $x = 5`	Zuweisung einer typisierten Variablen (Typ ist optional)
`[int] $x = "3.45" -as [Int]`	Konvertiert einen String-Wert in Int und schreibt ihn nach $x
`$x.GetType()`	Liefert Typinfos von $x
`$x.GetType(). FullName`	Liefert Typnamen von $x
`Clear-Variable x`	Löscht **Inhalt** von $x
`Remove-Variable x`	Löscht **Deklaration** von $x
`$true $false`	Wahr/falsch
`$Home`	Home-Folder des Nutzers
`$PSHome`	Installationsordner von PS
`$Error`	Liste aller Fehler seit Start der PowerShell
`Get-Item Variable:H*`	Zeigt alle definierten Variablen an, die mit H beginnen
`$x | Get-Member`	Zeigt Typ, Member, Methoden zu der Variablen an
`Get-Help about_Variables`	Hilfeartikel zu Variablen

Umgang mit Strings

`"Hi"` bzw. `'Hi'` bzw. `@'Hi@'`	Versch. Stringliterale (@-Notation: "Here-String")
`"a"+ $x + "c"`	Konkatenation
`"PC $nr"`	Ausdruckauflösung
`"Date: $(Get-Date)"`	Ausdruckauflösung
`"x:\$($pc)_VHD.vhdx"`	Ausdruckauflösung
`$a.Substring(4,3)`	Text extrahieren [5, 7]
`$myArr = $x -Split "<del>"`	Splitten String am Delimiter <del> auf
`$x = $myArr -Join"<del>"`	Verbindet Teilstringe aus myArr in x
`$x.replace("ü", "ue")`	Case-Sensitives Ersetzten von Teilstrings
`$x -replace "\bÖ", "Oe"`	Ersetzten von Teilstrings m.H. von regulären Ausdrücken
`"" | Get-Member -MemberType Method`	String-Methoden ansehen

Umgang mit nicht definierten Variablen

`$x ??= "n/a"`	Nimm x falls definiert, ansonsten schreibe Standardwert hinein
`${x}?.Property`	Wähle Property aus, falls existent, ansonsten null zurückgeben
`${arr}[100]`	Falls `arr` nicht existiert, gib null zurück

Ein- und Ausgabe

`Write-Host`	Erzeugt Ausgabe auf `stdout`
`$x = Read-Host "x eingeben"`	Benutzereingabe wird nach x gespeichert
`Clear-Host`	Löscht die Ausgabe auf der Konsole

Dokumente lesen und schreiben

`Get-Content`	Textdatei einlesen
`$x[0]`	Textzeile 0 auswählen
`Set-Content`	Textdatei überschreiben
`Add-Content`	Text in Textdatei anhängen
`Import-Csv`	Text in Textdatei anhängen
`$x[0].SpaltenName`	Spalte in Objekt 0 auswählen
`ConvertFrom-Csv`	Aus String CSV extrahieren
`Export-Csv`	CSV-Datei schreiben
`Import-Clixml`	XML aus einer Datei einlesen
`$x.Node.ElemName`	`Node.ElemName` auswählen
`ConvertFrom-Xml`	Aus String XML extrahieren
`Export-Xml`	XML-Datei schreiben
`ConvertFrom-Json`	Aus String JSON extrahieren
`$x.propertyName`	`propertyName` auswählen
`ConvertTo-Json`	JSON-String erzeugen

Arrays

`$x = "a","b","c"`	Array definieren
`$x = @(1,2,3)`	Array definieren
`$x = 1..10`	Zahlen von 1 bis 10 in x schreiben
`$x.Count`	Anzahl der Elemente holen
`$z = $x + $y`	Zwei Arrays verbinden
`$x = ("a", "b"), ("c", "d")`	Zwei-dimensionales Array erzeugen
`$x[0][1]`	Element "b"an (0, 1) holen
`$x = @{a = "w1"; b = "w2"}`	Assoziatives Array erzeugen
`$x["a"]` bzw. `$x.a`	Wert von Index "a"auslesen
`$Get-Help about_Arrays`	Anzahl der Elemente holen

Schleifen

`for ($i=1; $i -lt 6; $i++) { anweisungen() }`	zählergesteuert
`while($i -lt 5) { }`	kopfgesteuert
`do {} while($i -lt 5)`	fußgesteuert
`do {} until($i -eq 5)`	mit Abbruchbed.
`foreach ($i in $menge) { }`	elementgesteuert

Verzweigungen

`if (condition){} elseif{} else{}`	If-Abfrage
`Condition ? True : False`	Ternärer Operator
`switch($x) { 1 { $y=" A"} 2 { $y=" B"} }`	Switch-Case Abfrage:

Funktionen

Alles was nicht `Write-*` ausgegeben wird, wird zum Rückgabewert der Funktion

`function Get-Alter()`	**positionale** Parameter
`function Get-Alter($age)`	**benannte** Parameter
`function Get-Age([int]$Age)`	**benannte** und typisierte Parameter
`function Get-Alter() { param([int] $Age) }`	Variante 2
`Get-Alter 20`	Aufruf mit **positionalem** Parameter
`Get-Alter -Age 30`	Aufruf mit **positionalem** Parameter
`return`	Funktion (vorzeitig) verlassen

Fehlerbehandlung

`$?`	`$True` für Erfolg, `$False` für Fehlschlag
`<a> && <b>`	Führt <b> aus, falls <a> erfolgreich
`<a> || <b>`	Führt <b> aus, falls <a> **nicht** erfolgreich
`try{ #unsafe } catch{ #catchIt } finally{ #after }`	Try unsafe code, Catch error and finally do something
`$_`	Zugriff auf Error-Objekt innerhalb des catch-Blocks

Error-Action und **Warning-Action**:
`Stop`, `Continue`, `SilentlyContinue`, `Inquire`

`Get-Help about_CommonParameters`

Reguläre Ausdrücke
Module einbinden