

PowerShell

Referenz

2.262

Verwendete Version: PowerShell V6.0

© Stefan Menne 2008-2017

| Elementares | | |
|--|--|-------------------------------------|
| → Groß- und Kleinschreibung sind beliebig! | | |
| Cursortasten: | letzte Befehle anzeigen/bearbeiten | |
| → [Tab] | Befehl/Dateinamen/Optionen ergänzen | |
| ^C | Abbruch von laufenden Programmen | |
| ` | Zeilenumbruch entwerfen (Accent Grave!) | |
| [ret][ret] | Abschluss für manuelle Texteingabe | |
| ; | Befehlstrenner | |
| Get-History | Letzten Befehle anzeigen | |
| F7 | Pop-Up: Befehl wiederholen | |
| Hilfen | | |
| Get-Alias -? | Hilfe zum CmdLet Get-Alias | |
| Get-Help Get-Alias -Det | Detaillierte Hilfe zum CmdLet Get-Alias | |
| Get-Help * | Zeigt alle Hilftemen an. | |
| -detailed | Erweiterte Anzeige mit Beispielen | |
| -full | Komplette technische Hilfe | |
| Get-Help *Objekt* | Zeigt Themen mit "Objekt" im Namen | |
| Get-Help about* | Hilfe zu Powershell-Anweisungen | |
| Get-Help about_wildcard | Hilfe zu Wildcards | |
| Get-Help about_commonparameters | Fehlerbehandlung etc. z.B. -ErrorAction SilentlyContinue Hilfdateien aktualisieren | |
| Update-Help | | |
| <u>Informationen zu Kommandos:</u> | | |
| Get-Command *process* | Alle mit "process" im Namen | |
| Get-Alias * | Alias-Namen zeigen ("get-alias g*") | |
| <u>Internet-Tipps:</u> | | |
| blogs.msdn.com/Powershell | Microsoft Entwickler | |
| Terminal | | |
| Clear-Host | Terminal löschen | |
| Get-Host | Infos anzeigen | |
| Start-Transcript | Bildschirmprotokoll anfertigen | |
| Stop-Transcript | Stoppen des Bildschirmprotokolls | |
| Ausgabe | | |
| Write-Host "Text" | Bildschirmausgabe, Optionen: -ForegroundColor green -BackgroundColor red -NoNewline | |
| <u>Farben:</u> Black, Gray, Blue, Green, Cyan, Red, Magenta, Yellow, White, DarkRed, ... | | |
| Write-Output | Einfache Textausgabe, Aliase: echo , write | |
| Write-Error | Text als Fehler ausgeben | |
| Get-Content | Inhalt einer Datei ausgeben | |
| Format-Table | Tabellarisch anzeigen, Option: -AutoSize | |
| Format-List | Zeilenweise Eigenschaften anzeigen (fl) | |
| Format-Wide -column 2 | Mehrspaltige Ausgabe | |
| Out-GridView | Grafische Anzeige | |
| Out-Host -Paging | In einer Pipe: Seitenweises Anzeigen | |
| Out-File out.txt | Ausgabe in Datei speichern (-Append) | |
| Out-Printer | Direktes Ausdrucken | |
| <u>Sonder-Codes: (Wie Unix "\")</u> | | |
| " -- Single quote | " -- Double quote | `0 -- Null |
| `a -- Alert | `b -- Backspace | `f -- Form feed |
| `n -- New line | `r -- Carriage return | `t -- Horizontal tab |
| Bsp: Write-Host "Zeilen`n`nbruch`\$alpha" | | |
| <u>Konvertierung von Pipe-Daten:</u> | | |
| ConvertTo-Html | Ausgeben als HTML | |
| ConvertTo-SecureString | Verschlüsselt ausgeben | |
| Export-Csv | Ausgeben als CSV | |
| Export-Clixml | Ausgeben als XML | |
| <u>Here-Scripte:</u> | \$a=@" | \$a=@' |
| | Ein langer Text mit \$(3+4) | langer Text ohne Variablen "@ |
| Ausgaben-Umlenkung | | |
| Kanäle: * Alles, 1=stdout, 2=stderr, 3=Warnung, 4=Verbose, 5=Debug | | |
| 1> datei | stdout in Datei | |
| 1>> datei | stdout an Datei anhängen | |
| 2> datei | stderr umlenken | |
| 5>&1 | Debugmeldungen mit ausgeben | |
| <u>Befehls substitution:</u> | | |
| \$(cmd) | z.B. Stop-Process \$(ps ? {\$_.Name -match "fox" }) | |
| Datumsberechnungen | | |
| Get-Date -uformat "%Y-%m-%d" | Unix-Formatierung | |
| (Get-Date).AddDays(100) | Tage addieren | |
| (Get-Date -date "2010-3-12").AddDays(200) | Tage nach einem Datum | |
| ((Get-Date -date "1989-09-26").AddYears(30)).DayOfWeek | Wochentag | |
| ((Get-Date -date "2038-01-01") - (Get-Date "0:00")).Days | Differenz | |
| Freigaben | | |
| <u>Freigaben:</u> | | |
| New-Item "C:\Public" -type directory | | |
| New-SmbShare -FullAccess "Jeder" -name "Public" -path "C:\Public" | | |
| Get-SmbShare | Freigaben anzeigen | |
| Get-SmbShareAccess Public | Rechte abrufen | |
| Profile | | |
| \$Profile | Autostart-Datei für PS (auch PS-ISE) | |
| notepad \$Profile | Profile-Datei bearbeiten | |

| Kommandos | |
|---|---|
| Get-Alias | Alias-Namen anzeigen |
| Get-ChildItem | Directory-Listing |
| Show-Command Get-Process | Kommandos mit grafischem Frontend |
| Get-EventLog -logname application | -newest 10 |
| | Neueste 10 Ereignisse anzeigen lassen |
| Get-Service | Alle Dienste anzeigen |
| Get-Process | Alle Prozesse anzeigen |
| Get-PSSnapin | Powershell-Erweiterungen anzeigen |
| Start-Sleep -m 500 | 500 ms Pause |
| [intpr]::size * 8 | Zeigt Anzahl Bits für Integer an |
| Objekt-Zugriff | |
| Get-Member | Methoden + Eigenschaften anzeigen ("gm") |
| Bsp.: Get-Date gm * | |
| Format-List * | Alle Eigenschaften anzeigen (fl) |
| Bsp.: Get-Date fl * | |
| \$_ | Zugriff auf aktuelles Objekt (foreach, where) |
| (Get-Host).privatedata | Zugriff auf Eigenschaften eines Objekts |
| (get-host).PrivateData.ErrorForegroundColor="yellow" | |
| (Get-Host).GetType() | Zugriff auf Methoden eines Objekts |
| (Get-Process).count | Anzahl Objekte ermitteln |
| [DateTime]::Now | Statistischer Zugriff, [DateTime]::IsLeapYear(2012) |
| Variablen | |
| \$a = "Stefan Menne" | Variable neu anlegen |
| \$a | Variable anzeigen |
| \$b = Get-Process | Kommandoausgabe speichern |
| [int] \$a = 32 | Typvorgabe |
| (\$a).Length | Länge ausgeben |
| \$a,\$b,\$c = 1,2,3 | Mehrfach-Zuweisung |
| \$nn,\$vn = "Stefan;Menne".split(";") | Mehrfach-Zuweisung mit Split() |
| \$a,\$b = \$b,\$a | Inhaltstausch |
| Set-Variable -name pi -value 3.1415 -option constant | Konstante |
| Set-Variable -name euro -value 1.382 -option readonly | Nur Lesen |
| <u>Zugriff auf verbundene Variablen:</u> | |
| Get-EventLog System % { "\$(\$_.EntryType) : \$((\$_.Message))" } | |
| <u>Automatische Variablen:</u> | |
| \$\$ | Letztes Token der letzten Kommandozeile |
| \$? | Return-Code des letzten Kommandos |
| \$^ | Erstes Token der letzten Kommandozeile: i.A. Kommando |
| \$_ | Aktuelles Pipeline-Objekt |
| \$Args | Argumente des Scripts |
| <u>Variablen zur Programmierung:</u> | |
| \$true | Boolean WAHR |
| \$false | Boolean FALSCH |
| Test-Path Variable:xxx | Existenz testen |
| \$null | "nichts", also ein NULL-Objekt (zum Löschen von Objekten) |
| \$OFS | Trennzeichen für Konvert. von Arrays zu String (Default: space) |
| Arrays | |
| @() | Leeres Array |
| @(7) | Array mit einem Element |
| @(1..50) | Array mit 50 Elementen |
| \$a = 1, 'zwei', 'drei', 4 | Array definieren |
| \$a[2] | Array-Element anzeigen |
| \$a[0..2] | Mehrere El. anzeigen, \$a[-2..-1] |
| \$a.Length | Anzahl Elemente |
| \$b=@(Get-Process) | Array mit Prozessen erstellen |
| \$c=\$a + \$b | Zwei Arrays zusammenfügen |
| <u>Assoziative Arrays: {..}</u> | |
| \$a = @{} | Leeres assoziatives Array |
| \$a+=@{,Anton" = 1} | Element hinzufügen |
| @{Autos=3; Kinder=5} | assoziatives Array definieren |
| \$a=[ordered]@{Häuser=2; Autos=7; Kinder=3} | Sortiertes Array erzeugen |
| Dateiverarbeitung: | |
| <u>Daten-Datei "user.txt":</u> | |
| sorglos;Sorglos;Susanne | |
| wolters;Wolters;Willi | |
| <u>Powershell-Script:</u> | |
| foreach (\$User in Get-Content "user.txt") { | |
| \$LoginName, \$Name, \$Vorname = \$User.split(",") | |
| write-host "\$LoginName": \$Vorname \$Nachname" | |
| } | |
| <u>CSV-Datei "user.csv":</u> | |
| Logname,Nachname,Vorname,Funktion | |
| sorglos,Sorglos,Susanne,Dozent | |
| wolters,Wolters,Willi,Studierender | |
| <u>Powershell-Script:</u> | |
| foreach(\$i in Import-Csv "daten.csv") { echo \$i.Nachname } | |
| # oder: | |
| Import-Csv \daten.csv % { \$_.Nachname } | |

Tools

Powershell-Tools

Select-Object (Alias: "select")

```
[-property] ID,Name           Eigenschaften auswählen
-first 7                      die ersten 7 Datensätze
-last 12                      die letzten 12 Datensätze
-unique                       gleiche Datensätze unterdrücken
Bsp: get-Process | Select-Object -property ws,cpu,name -first 5
@{Name="Zeit"; Expression={$_.Cpu}} Eigene Spalten
Bsp: ls | Select Name,@{ N="Datum";E={$_.LastWriteTime}},
@{ N="Alter";E={ ((Get-Date) - $_.CreationTime).Days }
```

Sort-Object (Alias: "sort")

```
[-property] Name             Sortierung nach der Eigenschaft
-unique                       gleiche Datensätze unterdrücken
-descending                   absteigend sortieren
-caseSensitive                 Groß-/Kleinschreibung beachten!
Bsp: get-childitem | sort-object -property length -descending
```

Measure-Object

```
-property ...
-average|sum|minimum|maximum Zahlen
-line|word|character          Text
Bsp: get-childitem | measure-object -property length -min -max -av
```

Where-Object (Alias: "?", "where")

```
Bsp: get-service | where-object {$_.Status -eq "Stopped"}
get-process | where {$_.ws -gt 10MB -and $_.cpu -ge 10}
```

Falls nur 3 Parameter nötig sind:

```
get-process | where Workingset -gt 10MB
```

Foreach-Object (Alias: "%", "foreach")

```
Bsp: @(30,40,80,200) | ForEach-Object {$ _ / 10}
400,600,800,100 | ForEach-Object {[Console]::Beep($_,200)}
ls c:\ | % {
    $n+=1
    Write-Host "$n : ${$_name}"
}
```

Foreach-Anweisung: (siehe unter: Schleifen, S.4)

```
foreach ($i in Get-Process) {$i}
```

Operatoren

Standard-Operatoren:

```
+ - * / % = += -= *= /= %= ++ --
```

Vergleiche:

```
-eq (==) -ne (!=) -gt (>) -ge (>=) -lt (<) -le (<=)
42 -is [int]                Typ-Vergleich
42 -as [string]             Typ-Konvertierung
```

String-Vergleiche:

```
"lauf" -match "[abc]"      (-notmatch)
"Stefan" -like "S*"        (-notlike)
2,3,4 -contains 3          (-notcontains)
"Berta" -in "Anton","Berta","Claus" (-notin)
```

Logische Operatoren:

```
-and -or -not -xor !
```

Formatierung der Ausgabe

Formatierung mit -f: { index[,alignment][:formatString]}

```
Bsp: "{0,-10}"              Parameter 0: 10 Stellen linksbündig
"{0,20}"                   Parameter 0: 20 Stellen rechtsbündig
"{0,10:N2}"                Zahl mit 10 Stellen, 2 Nachkomma
"{0,10:F1}"                Fließkommazahlen 10 Stellen, 1 NK
```

Format-Codes: {Index:Code[Genauigkeit]}

```
Codes: N (Zahl) C (Währung), D (Dezimal), P (Prozent), X (Hex.)
Oder: $a = 47.111; $a.ToString("0,000.00")
```

Beispiel:

```
"{0}{1}" -f "Anton", 23
$X = "{0,8:N2} GB von {1,8:N2} GB sind frei" -f 345.34, 876.3325
write-host -f "green" $X
```

Strings

String-Operatoren:

```
„Hans“ + „Wurst“           Strings zusammenfügen
„Hans“* 50                  Zeichenkette wiederholen
```

Member-Functions: ("text" | gm)

```
"go east" -replace "east","west" Ersetzung
$a,$b,$c = "Hans,Berta,Claus" -split ","
$a,$b,$c = "Hans,Berta,Claus".split(",") String splitten+zuweisen
"Stefan","Menne" -join " "      Strings verbinden
"HansWurst.jpg".TrimStart("Hans") Anfang entfernen
"Ein Text".ToUpper()            Großschrift
"Ein Text".ToLower()           Kleinschrift
"Ein Text".Insert(4,"toller ") Text einfügen
"Ein Text".compareTo("Ein Test") Vergleich
"Ein Text".Contains("Ein")      Auf Enthalten prüfen
"Ein Text".Chars(5)            Ein Zeichen herausholen
" Ein Text ".Trim()            Text trimmen
" Ein Text ".Length            Länge ermitteln
" Ein Text ".TrimEnd().Length  Am Ende trimmen
"Ein Text".Substring(4)        Substring ab Pos. 4
"Ein Text".Substring(4,2)     Substring mit Länge 2
"Ein Text".Remove(2,4)         Zeichen löschen
```

Verschiedenes

Provider

Provider-Arten:

```
Get-PSProvider              Alle Provider anzeigen (Drives beachten!)
Get-PSDrive                 Anzeige aller Laufwerke
Bsp: cd HKLM:               In Registry-Ordner HKLM wechseln
New-PSDrive -Name remote -PSProvider FileSystem -Root \\ds\doz
                             Neues Laufwerk "remote:" erzeugen (statt: net use Z: \\ds\doz)
Remove-PSDrive remote      Laufwerk entfernen
```

Item-Tools:

Ein Item kann ein Verzeichnis, Registry-Eintrag, Variable, Active-Directory-Eintrag oder Datei sein.

```
Get-Item .                  Aktuelles Objekt holen/anzeigen
Get-Childitem .            Items des auflisten (-Recurse)
Get-ItemProperty .        Eigenschaften vom akt. Obj. abrufen
New-Item -path "H:\\" -name „test.txt“ -type „file“ -value „inhalt“
                             Item Objekt anlegen
```

```
Set-Item                   Existierendes Objekt mit Wert belegen
Clear-Item                 Inhalt löschen
```

Weitere Befehle: Copy-Item, Remove-Item, Rename-Item, Move-Item

Location-Befehle:

```
Get-Location               Aktuellen Ort anzeigen
Set-Location               Ort wechseln
Push-Location              Aktuellen Ort in Stack speichern
Pop-Location               Zum letzten Ort aus dem Stack wechseln
```

Zugriff auf Systemumgebungen

.NET:

```
$web = new-object System.Net.WebClient (ca. 12 Klassen relevant)
$web.DownloadFile("https://www.bib.de/", "d:\bib.html")
(Auch: Invoke-WebRequest „https://www.bib.de/“)
$D = New-Object -Type System.DateTime 2019,9,26
$D.get_DayOfWeek()
[Math]::Pi                  [Math]::Sin(x)
```

COM:

```
$obj = New-Object -comobject "wscript.network"
$obj.username
```

WMI:

```
Get-WmiObject -List
Get-WmiObject -List | ? {$_.Name -match "Share"}
Get-WmiObject win32_bios
Get-WmiObject win32_service -computername 127.0.0.1 | ft
```

Dialogboxen

```
$X=[reflection.assembly]::LoadWithPartialName("System.Windows.Forms")
[System.Windows.Forms.MessageBox]::Show("Meldung", "Überschrift",
Buttons, Art)
```

| Buttons: | Rückgabewerte: |
|--------------------------------|----------------------------|
| 0 = OK | 0 = nichts gedr. 4 = Retry |
| 1 = OK, Abbrechen | 1 = Ok 5 = Ignore |
| 2 = Abbrechen, Wiederholen, OK | 2 = Cancel 6 = Yes |
| 3 = Ja, Nein, Abbrechen | 3 = Abort 7 = No |
| 4 = Ja, Nein | |
| 5 = Wiederholen, Abbrechen | |

Symbol Werte:

```
K04_13 Frage 32 K04_15 Warnung 48
K04_14 Fehler/Stop 16 K04_16 Info 64
```

Zugriffsrechte ACL

```
$Datei = "C:\test.txt"      Datei muss existieren
$acl = Get-Acl $File
$zugriff = "bibtest\sorglos", "FullControl", "Allow"
                             Accounts ermitteln mit: Get-WmiObject Win32_Account
$regel = New-Object System.Security.AccessControl.FileSystemAccessRule $zugriff
$acl.SetAccessRule($accessRule)
$acl | Set-Acl $Datei
```

Netzwerk

IP anzeigen/setzen

```
Get-NetIPInterface -AddressFamily IPv4
Remove-NetIPAddress -InterfaceAlias "Ethernet0" -AddressFamily "IPv4" -
-Confirm:$false
Remove-NetRoute -InterfaceAlias "Ethernet0" -Confirm:$false
New-NetIPAddress -InterfaceAlias "Ethernet0" -AddressFamily "IPv4" -
-IPAddress 10.1.1.1 -PrefixLength 8 -DefaultGateway 10.0.0.1
Set-DnsClientServerAddress -InterfaceAlias "Ethernet0" -ServerAddr 10.0.0.1
```

Zugriff auf Fremdrechner

Remoting ermöglichen:

```
Winrm quickconfig; Enable-PSRemoting -Force
Set-Item WSMAN:\localhost\client\TrustedHosts *
Set-Item WSMAN:\localhost\client\TrustedHosts 10.0.0.1
Restart-Service WinRM
```

Zugriffsarten:

```
a) per Kommando:
$pw = Get-Credential Administrator
Invoke-Command -ComputerName 192.168.10.128 -ScriptBlock { Get-
Childitem C:\ } -credential $pw
b) Session starten:
Enter-PSsession 192.168.10.128
```

Scripte

Script-Einstellung

```
Get-ExecutionPolicy Aktuelle Einstellung abfragen
Set-ExecutionPolicy RemoteSigned Scripte erlauben
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser falls nicht Admin
Typen: Restricted, RemoteSigned, AllSigned, Unrestricted
```

Scripte, Programme starten

```
& befehl Befehl in neuer Umgebung starten
. script.ps1 Befehl in aktueller Umgebung starten
powershell.exe -noexit &"c:\machwas.ps1"
& $string Anweisungen im String ausführen
(oder: Invoke-Expression "dir *.exe")
```

Debug-Level setzen:

```
Set-Psdebug -step Schrittweise Ausführung von Scripten
Set-PsDebug -trace 1 Debug-Level setzen
```

Andere Programme starten:

```
&'C:\Programme\Microsoft Office\Office\EXCEL.EXE' Test.xls
start excel.exe .\Test.xls Programm starten (Start-Process)
```

Kommentare

```
# Kommentar bis Ende Zeile
<# mehrzeiliger Block-Kommentar
Kommentar #>
```

Werteübergabe, Variablen

Interaktive Eingabe:

```
$erg = Read-Host "Bitte eine Zahl eingeben"
$passwort = Read-Host "Passwort" -AsSecureString
```

Kommandozeilen-Argumente in C-Stil:

```
$args[0].. $args[n] Argumente
$^ Kommandozeilen-Programmname
```

Kommandozeilenparameter in PowerShell-Stil

```
param ( [Int] $Alter, [String] $Name = "Menne", [Switch] $Aktiv )
Aufrufmöglichkeiten:
.\script.ps1 40 Meierhofer
.\script.ps1 -Name „Müller“ -Alter 50 -Aktiv
```

Parameter erzwingen mit: [parameter(Mandatory = \$true)]

Hilfen: get-help about_functions_advanced_parameters

Parameter-Typ-Deklaration:

Zahlen: [int], [long], [double], [decimal], [float], [single], [byte]

Zeichen: [string], [char] **Datum:** [datetime] "11.4.2011"

Switch: [Switch] \$test **Dann:** if (\$test)

Weitere Type: [bool], [array], [hashtable], [xml],[ADSI] [ADS], [WMI]

Gültigkeitsbereich-Deklarationen:

```
$global:a=1 Überall gültig $script:a=1 nur im Script gültig
$local:a=1 ab hier gültig $using:a=1 nur im Start-Job gültig
$private:a=1 Variable nur hier gültig
```

Funktionen

Variante A: Funktionsparameter

```
function summe ($a = 3, $b = 7) { return $a + $b }
Aufruf: summe 17 04
```

Variante B: benannte Parameter

```
function summe { param ($WertA = 5, $WertB = 10)
return $WertA + $WertB
}
Aufruf: summe -WertA 2 -WertB 3
summe 4 7
summe -WertB 4
```

Schleifen

Zählschleife:

```
for ($i = 0; $i -le 20; $i++) {Write-Host $i}
foreach ($i in 0..20) { $i }
0..20 dasselbe in kurz
```

Durcharbeiten von Arrays/CmdLets:

```
foreach ($i in $Array) { echo $i }
foreach ($file in Get-Childitem) { Write-Host $file }
ps | foreach -Begin { $i=0 }
-Process { $i++ }
-End {"# Prozesse: " + $i}
kurz: ps | foreach { $i=0 } { $i++ } { $i }
```

Weitere Strukturen:

```
if (<Test1>) { <Anweisungsliste 1> }
elseif (<Test2>) { <Anweisungsliste 2> }
else { <Anweisungsliste 3> }
switch ( "Montag" ) {
"Montag" { echo „Mon“ ; break }
default { echo "Was anderes" }
}
do { <Anweisungsliste> } while (<Bedingung>)
while (<Bedingung>) do { <Anweisungsliste> }
try { kommando > $NULL } catch; if ( ! $? ) { ... }
```

Passwörter

```
$pw=ConvertTo-SecureString -AsPlainText -Force -String "EinTest.11"
Dekodieren:
$pw = Read-Host "Passwort eingeben" -AsSecureString
[Runtime.InteropServices.Marshal]::PtrToStringAuto(
[Runtime.InteropServices.Marshal]::SecureStringToBSTR($pw))
```

ActiveDirectory

AD-Tools

Active-Directory –Module installieren/deinstallieren:

```
Install-WindowsFeature AD-Domain-Services
Import-Module ADDSDeployment
Import-Module ActiveDirectory Laden des AD-Moduls
Install-ADDSForest -DomainName "test.de" -SafeModeAdministratorPass $pw
Install-WindowsFeature RSAT-AD-AdminCenter
Uninstall-ADSDomainController -LocalAdministratorPassword $pw
-LastDomainControllerInDomain -RemoveApplicationPartitions
```

Active-Directory als Provider (Get-PSProvider):

```
cd ad: Ins Active-Directory wechseln
ls Aktuelles AD-Verzeichnis auflisten
cd "DC=bib,DC=de" Ins AD wechseln
cd "CN=Users" In den Ordner „Users“ wechseln
cd "AD:\CN=Users,DC=bib,DC=de" Absolute Adressierung
get-command *-*AD* Alle Kommandos zu AD zeigen
```

Informationen abfragen:

```
Get-ADDomain Anzeige der aktuellen Domain
Get-ADDomainController
Get-ADUser -Filter * -Properties * Alle Benutzer + Eigenschaften
Get-ADUser dozmen -Properties * Alle Eigenschaften zum Benutzer holen
Get-ADComputer -Filter * Computer auflisten
```

User anlegen, löschen:

```
New-ADUser -name "Stefan Menne" -samAccountName dozmen `
-UserPrincipalName dozmen@bib.de -AccountPassword $pw `
-Path "CN=Users,DC=bib,DC=de" -enabled $true
User wählen über: samAccountName, DistinguishedName oder SID
```

Name: DisplayName, GivenName, Surname, OtherName, EmailAddress, DistinguishedName, SID
Account: UserPrincipalName, AccountPassword, ChangePasswordAtLogon, Enabled, samAccountName
Adresse: City, Country, PostalCode, State, StreetAddress
Pfade: HomeDirectory, HomeDrive, ProfilePath, ScriptPath

Remove-ADUser dozmen -Confirm:\$false

User-Daten verändern/ergänzen:

```
Set-ADUser dozmen -HomeDirectory "\\Win2K12R2B\Home\$logname" `
-HomeDrive "H:" -ProfilePath "\\Win2K12R2B\Home\$logname\Profile"
Set-ADAccountPassword dozmen -NewPassword $pw
```

Organisationseinheiten:

```
New-ADOrganizationalUnit -Name "Firma" -Path "DC=bib,DC=de"
Get-ADOrganizationalUnit -Filter *
(Remove-ADOrganizationalUnit, Set-ADOrganizationalUnit)
```

Gruppen:

Get-ADGroup -Filter *

Objekte verschieben:

Move-ADObject "CN=Stefan Menne,... -TargetPath "DC=bib,DC=de"

Testen auf Existenz:

```
Get-Childitem "AD:\CN=Stefan Menne,DC=bib,DC=de" 2>&1 > > $null
if ( ! $? ) { ... } # User existiert noch nicht
```

DHCP

Tools: DHCP installieren (deinstallieren mit Remove-...):

Install-WindowsFeature DHCP -IncludeManagementTools

DHCP-Server in ActiveDirectory (DC) registrieren:

Add-DhcpServerInDC -DnsName "bibtest.de"

Add Scopes:

```
Add-DhcpServerV4Scope -StartRange 192.168.10.10 -EndRange `
192.168.10.254 -SubnetMask 255.255.255.0 `
-Name "IP-Test-Bereich" -Description "Test" -Verbose
```

Alle DHCP-Scopes aktivieren:

Get-DhcpServerV4Scope | Set-DhcpServerV4Scope -State Active

DHCP-Optionen setzen:

```
Set-DhcpServerV4OptionValue -DnsDomain "menne.de" `
-Router 192.168.10.2 -DnsServer 192.168.10.3 -Force
```

Einzelnen Bereich löschen:

Remove-DhcpServerV4Scope -ScopeId 192.168.10.0

DHCP-Server an Netzwerkkonfig binden:

```
Set-DhcpServerV4Binding -BindingState $true -InterfaceAlias Ethernet
Get-DhcpServerV4Scope
```

DNS

DNS installieren:

Install-WindowsFeature -Name 'DNS' -IncludeManagementTools

DNS-Zonen erstellen:

```
Add-DnsServerPrimaryZone -Name "bib.de" -ReplicationScope Forest
Add-DnsServerPrimaryZone -Name "5.0.10.in-addr.arpa" `
-DynamicUpdate Secure -ReplicationScope Forest
```

DNS-Einträge erstellen:

```
Add-DnsServerResourceRecordA -ZoneName "bib.de" `
-Name "anton" -IPv4Address 10.0.5.1
Add-DnsServerResourceRecordPtr -ZoneName "5.0.10.in-addr.arpa" `
-Name 1 -PtrDomainName 'anton.bib.de'
```

DNS-Zonen abfragen:

```
Get-DnsServerZone
Get-DnsServerResourceRecord 10.20.172.in-addr.arpa
Get-DnsServerRootHint
```

Dialog-Verarbeitung

Formulare

```
[void][reflection.assembly]::LoadWithPartialName("System.Windows.Forms")
$form=new-object "System.Windows.Forms.Form"
$form.TopMost = $true
$form.Text="Dies ist ein leeres Formular!"

# Textinhalt erzeugen:
$label=new-object "System.Windows.Forms.Label"
$label.Text='Hallo Welt'; $label.Top=10

# Button erstellen:
$button = New-Object "System.Windows.Forms.Button"
$button.Text = "OK"; $button.Top=200
$button.Add_Click({$form.Close()}) # Aktion nach dem Klicken!

# Alles in die Form einfügen:
$form.Controls.Add($label)
$form.Controls.Add($button)
$form.ShowDialog()
```

Weitere Möglichkeiten

```
Get-Module -list -All
import-Module PowerShellGet
Install-Module *get-eventlog*
Get-NetFirewallRule
Get-Command *fire*
```

Diese Referenz ist im Rahmen meiner Unterrichtstätigkeit am bib entstanden. Die Anordnung und Auswahl der Befehle ist aus meinen persönlichen Bedürfnissen für unterrichtliche Zwecke entstanden.

Verwenden Sie diese Referenz als Nachschlagewerk für PowerShell-Befehle. Sie darf frei für Lehrveranstaltungen kopiert und verteilt werden (Relevant sind die Seiten 1-4), wenn mein Copyright unten auf jeder Seite lesbar bleibt.

Anregungen und Meinungen schreiben Sie bitte an:
Stefan.Menne@gmx.de