

# PowerShell

## Referenz



Verwendete Version: PowerShell V5.1

© Stefan Menne 2008-2019

## PowerShell-Grundlagen

Elementares		
Groß- und Kleinschreibung sind beliebig!		
→ [Tab]	Befehl/Namen/Param./Optionen ergänzen	
^C	Abbruch von laufenden Programmen	
↓ ↑ ← →	Cursortasten: letzte Befehle bearbeiten	
`	Zeilenumbruch entwerfen (Accent Grave!)	
[ret][ret]	Abschluss für manuelle Texteingabe	
;	Befehlstrenner	
Get-History	Letzte Befehle anzeigen	
Hilfen		
Get-Alias -?	Hilfe anfordern (hier zum CmdLet Get-Alias)	
Get-Help	Hilfe zur Hilfe, Bsp:	
Get-Help Get-Alias -Detailed	Get-Help *	
Get-Help about*		
Update-Help	Hilfdateien aktualisieren	
Informationen zu Kommandos:		
Get-Command *process*	Alle mit "process" im Namen	
Get-Alias *	Alias-Namen zeigen ("get-alias g*")	
Internet-Tipps:		
blogs.msdn.com/Powershell	Microsoft Entwickler	
Einige Kommandos		
Get-ChildItem	Directory-Listing	
Get-Service	Alle Dienste anzeigen	
Get-Process	Alle Prozesse anzeigen	
Get-EventLog -logname application -newest 10	Neueste 10 Ereignisse anzeigen lassen	
Get-PSSnapin	Powershell-Erweiterungen anzeigen	
Show-Command Get-Process	Kommandos mit grafischem Frontend	
Terminal		
Clear-Host	Terminal löschen	
Get-Host	Infos anzeigen	
Start-Transcript	Bildschirmprotokoll anfertigen	
Stop-Transcript	Stoppen des Bildschirmprotokolls	
Ausgabe		
Bildschirm-Ausgaben:		
Write-Host "Text"	Direkte Bildschirm-Ausgabe, Optionen:	
-ForegroundColor green -BackgroundColor red -NoNewline		
Farben: Black, Gray, Blue, Green, Cyan, Red, Magenta, Yellow, White, DarkRed, ...		
Write-Output	Einfache Textausgabe, Aliase: <b>echo</b> , <b>write</b>	
Write-Error	Text als Fehler ausgeben	
Format-Table	Tabellarisch anzeigen, Option: -AutoSize	
Format-List	Zeilenweise Eigenschaften anzeigen (fl)	
Format-Wide -column 2	Mehrspaltige Ausgabe	
Weitere Ausgaben:		
Out-GridView	Grafische Anzeige	
Out-Host -Paging	In einer Pipe: Seitenweises Anzeigen	
Out-File out.txt	Ausgabe in Datei speichern (-Append)	
Get-Content	Inhalt einer Datei ausgeben (kurz: cat)	
Out-Printer	Direktes Ausdrucken	
Sonder-Codes: (vgl. Unix "\")		
' -- 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`n`nbruch \$alpha"		
Konvertierung von Pipe-Daten:		
ConvertTo-Html	Ausgeben als HTML	
Export-Csv	Ausgeben als CSV	
Export-Clixml	Ausgeben als XML	
Formatierung der Ausgabe		
Formatierung mit -f: {index[,alignment][:formatString]}		
Bsp:	"{0,-10}" -f "test" Parameter 0: 10 Stellen linksbündig	
	"{1,40}" -f "test","hallo" Parameter 1: 40 Stellen rechtsbündig	
	"{0,10:N2}" -f 123.4567 Zahl mit 10 Stellen, 2 Nachkomma	
	"{0,20:F1}" -f 1223.345e1 Fließkommazahlen 20 Stellen, 1 NK	
Format-Codes: {Index:Code[Genauigkeit]}		
Codes: N (Zahl) C (Währung), D (Dezimal), P (Prozent), X (Hex.), F (Float)		
Beispiel:		
\$x = "{0,8:N2} GB von {1,8:N2} GB sind frei" -f 345.34, 876.3325		
write-host -f "green" \$x		
Ausgaben-Umlenkung		
Kanäle: * Alles, 1=stdout, 2=stderr, 3=Warnung, 4=Verbose, 5=Debug		
> datei	stdout in Datei (oder: ">")	
>> datei	stdout an Datei anhängen (oder: ">>")	
2> datei	stderr umlenken	
5>&1	Debugmeldungen mit ausgeben	
Befehlssubstitution:		
{cmd}	z.B. Stop-Process \$( ps   where Name -match "fox" )	
Paketmanagement		
Get-WindowsPackage -online		
Get-WindowsOptionalFeature -online		
Disable-WindowsOptionalFeature -FeatureName WindowsMediaPlayer -online		

Profile	
\$Profile	Autostart-Datei für PS (auch PS-ISE)
notepad \$Profile	Profil-Datei bearbeiten
Objekt-Zugriff	
Get-Date   Get-Member	Methoden + Eigenschaften anzeigen ("gm")
Bsp.: gm *	
Get-Date   Format-List *	Alle Eigenschaften + Werte anzeigen
\$_	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	Statischer Zugriff, [DateTime]::IsLeapYear(2012)
Datumsberechnungen	
Get-Date -uformat "%Y-%m-%d"	Unix-Formatierung
(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
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
\$vn,\$nn = "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
Zugriff auf verbundene Variablen:	
Get-EventLog System   % { "\$(\$_.EntryType) : \$(\$_.Message)" }	
Automatische Variablen:	
\$_	Return-Code des letzten Kommandos
^	1. Token des letzten Kommandos (i.A. Kommando), \$\$ = letztes Token
\$Args	Argumente eines Scripts
Variablen zur Programmierung:	
\$true	Boolean WAHR (oder: \$false)
Test-Path Variable:xxx	Existenz testen
\$null	NULL-Objekt
\$ofs	Trennzeichen für Konvert. von Arrays zu String (Default: space)
Variablen als Kommando ausführen: (s. Linux eval-Kommando)	
Invoke-Expression \$Befehl	
Here-Scripte:	\$a = @'
Ein langer Text	langer Text
mit \$(3+4)	ohne Variablen
@	'@
Arrays	
@(7)	Array mit einem Element (@) = Leer
@(1..50)	Array mit 50 Elementen (= 1..50)
\$a = 1, 'zwei', 'drei', 4	Array durch Aufzählung definieren
\$a[2]	3. Array-Element anzeigen
(1, 'zwei', 'drei', 4)[-2]	zweitletztes 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
Assoziative Arrays: {...}	
\$a = @{}	Leeres assoziatives Array
\$a+=@{"Anton" = 1}	Element hinzufügen
@{Autos=3; Kinder=5}	assoziatives Array definieren
\$wotag = @{"Monday" = "Montag"; Friday = "Freitag"}	
\$wotag[ (Get-Date).DayOfWeek.ToString() ]	
\$a=[ordered]@{Häuser=2; Autos=7; Kinder=3}	
Zugriff auf Systemumgebungen	
.NET:	
\$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 win32_bios	
Get-WmiObject win32_service -computername 127.0.0.1   ft	
Web-Requests:	
Invoke-WebRequest -URI http://google.de	

## Tools

### PowerShell-Tools

**Select-Object** (Alias: "select")  
 [-property] ID,Name,Cpu *Eigenschaften auswählen*  
 -first 7 *die ersten 7 Datensätze (oder: -last 12)*  
 -unique *gleiche Datensätze unterdrücken*  
 Bsp: Get-Process | Select-Object ws,cpu,name -last 5  
 Eigene Spalten def.: @{Name="Zeit"; Expression={\$\_.Cpu} }  
 Bsp: ls | Select Name, @{ N="Datum"; E={\$\_.LastWriteTime} }

**Sort-Object** (Alias: "sort")  
 [-property] Name *Sortierung nach der Eigenschaft*  
 -unique *gleiche Datensätze unterdrücken*  
 -descending *absteigend sortieren (default: ascending)*  
 -caseSensitive *Groß-/Kleinschreibung beachten!*  
 Bsp: Get-Childitem | sort-object -property length -descending

**Where-Object** (Alias: "?", "where")  
 get-service | where-object {\$\_.Status -eq "Stopped"}  
 get-process | ? {\$\_.ws -gt 10MB -and \$\_.cpu -ge 10}  
 Sonderfall: Für einfache Abfragen (ohne '-and' oder '-or')  
 get-process | where Workingset -gt 10MB

**ForEach-Object** (Alias: "%", "foreach")  
 @(30,40,80,200) | ForEach-Object {\$\_ / 10 }  
 400,600,800,100 | % { [Console]::Beep(\$\_/200) }  
 ls c:\ | % { \$n++; Write-Host "\$n: \${\$\_}name" }  
 Foreach-Anweisung: (siehe unter: Schleifen, S.4)  
 foreach (\$i in Get-Process) { \$i }

**Measure-Object**  
 -average|sum|minimum|maximum *Zahlen*  
 -line|word|character *Zeilen, Wörter, Zeichen zählen*  
 Bsp: Get-Childitem | measure-object -property length -min -max -av

**Group-Object:**  
 ps | Group-Object Name *Gruppierung nach Namen*

## Verschiedenes

### Operatoren, Funktionen

**Standard-Operatoren:** + - \* / % += -= \*= /= %= ++ --  
**Vergleiche:** -eq -ne -gt (>) -ge (≥) -lt (<) -le (≤)  
 42 -is [int] *Typ-Vergleich*  
 42 -as [string] *Typ-Konvertierung*  
 2,3,4 -contains 3 *Array-Vergleiche (-notcontains)*

**String-Vergleiche:**  
 "lauf" -match "einlaufen" *(-notmatch)*  
 "Stefan" -like "S\*" *(-notlike)*

**Logische Operatoren:** -and -or -not -xor !  
**Math. Funkt.:** [Math]::Truncate(1.23)  
 [System.Math] | Get-Member -Static -MemberType Methods

### Strings

**Formatierungen:**  
 47.1111.ToString("0000.00") *formatieren ("####.00")*  
 "test".PadLeft(8) *auf 8 Zeichen rechtsbündig*

**String-Operatoren:**  
 "Hans" + "Wurst" *Strings zusammenfügen*  
 "Menne" \* 500 *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 (auch ToLower)*  
 "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) *Substr. ab Pos.4 mit Länge 2*  
 "Ein Text".Remove(2,4) *Zeichen löschen*

### Datenerarbeitung:

#### Daten-Datei "user.txt":

sorglos;Sorglos;Susanne  
 wolters;Wolters;Willi

#### PowerShell-Script:

```
foreach ($User in Get-Content "user.txt") {
    $LoginName, $Name, $Vorname = $User.split(";")
    write-host "$LoginName: $Vorname $Nachname"
}
```

#### CSV-Datei "user.csv":

Logname,Nachname,Vorname,Funktion  
 sorglos,Sorglos,Susanne,Dozent  
 wahnsinn,Wahnsinn,Willi,Sekretariat ...

#### PowerShell-Script:

```
foreach ($i in Import-Csv "daten.csv") { echo $i.Nachname }
# oder: Import-Csv daten.csv | % { $_.Nachname }
```

### 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:

**Item:** *Verzeichnis/Registry-Eintrag/Variable/Active-Directory-Eintrag/Datei/...*  
**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*

### Dialogboxen

\$X=[reflection.assembly]::LoadWithPartialName("System.Windows.Forms")

\$box = [System.Windows.Forms.MessageBox]

\$box::Show("Meldung", "Ueberschrift", Buttons, Art)

Buttons:	Art:	
0 = OK	32 = Frage	48 = Warnung
1 = OK, Abbrechen	64 = Info	16 = Fehler(Stopp)
2 = Abbrechen, Wiederholen, OK	<b>Rückgabewerte:</b>	
3 = Ja, Nein, Abbrechen	0 = nichts gedr.	4 = Retry
4 = Ja, Nein	1 = Ok	5 = Ignore
5 = Wiederholen, Abbrechen	2 = Cancel	6 = Yes
	3 = Abort	7 = No

### Zugriffsrechte ACL

#### Verzeichnis erstellen und ACL anzeigen:

\$path = "C:\benutzer\sorglos"; mkdir \$path  
**(Get-Acl \$path).AccessToString**

#### Zugriffsrechtregel erstellen:

\$perm = "bittest\sorglos","FullControl","ContainerInherit,ObjectInherit","None","Allow"  
**\$ACE = New-Object System.Security.AccessControl.FileSystemAccessRule \$perm**

#### Zugriffsrechtregel auf Verzeichnis anwenden:

\$acl = Get-Acl \$path  
**\$acl.SetAccessRule(\$ACE)**

#### Benutzer "Jeder" entfernen:

\$acl = Get-Acl \$path  
**\$ACE = \$acl.access | where IdentityReference -match "Jeder"**  
**\$acl.SetAccessRuleProtection(\$true,\$true)**  
**\$acl.RemoveAccessRule(\$ACE)**

## Netzwerk

### IP anzeigen/setzen

**Get-NetIPInterface** -AddressFamily IPv4

**Remove-NetIPAddress** -InterfaceAlias "Ethernet" -AddressFamily "IPv4" -Confirm:\$false

**Remove-NetRoute** -InterfaceAlias "Ethernet" -Confirm:\$false

**New-NetIPAddress** -InterfaceAlias "Ethernet" -AddressFamily "IPv4" -IPAddress 10.1.1.1 -PrefixLength 8 -DefaultGateway 10.0.0.1

**Set-DnsClientServerAddress** -InterfaceAlias "Ethernet" -ServerAddr 10.0.0.1

### Zugriff auf Fremdrechner

#### Remoting ermöglichen:

**Set-NetConnectionProfile** -NetworkCategory Private  
**Set-WSManQuickConfig**  
 Set-Item WSMan:\localhost\client\TrustedHosts "10.0.0.1,10.0.0.2" -Force

#### Zugriffsarten:

**a) per Kommando:**  
 \$pw = Get-Credential Administrator  
 Invoke-Command -ComputerName 10.0.0.2 -ScriptBlock {  
 Get-Childitem C:\} -credential \$pw

#### b) Session starten:

Enter-PSSession 10.0.0.2

### Freigaben

#### Freigaben erstellen und anzeigen:

**New-Item** "C:\Public" -type directory  
**New-SmbShare** -FullAccess "Jeder" -name "Public" -path "C:\Public"

**Set-SmbPathAcl** *Pfad für ACL's vorbereiten*

**Get-SmbShare** *Freigaben anzeigen*

**Get-SmbShareAccess** Public *Rechte abrufen*

## 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

### Anderer 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"

#### 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: **param** ([parameter(Mandatory = \$true)] [String] \$Name )  
 Weitere Möglichkeiten: 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"  
 Boolesch: [switch] \$Descending, [bool]  
 Sonstige: [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 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 8

### Schleifen/Bedingungen

#### 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** { \$cpu -gt 10 } { Write-Host -f red "CPU > 10 !!" }  
**elseif** { \$cpu -lt 5 } { Write-Host -f green "CPU < 5 !!" }  
**else** { Write-Host "5 <= CPU <= 10 !!" }  
**switch** { \$tag } {  
 "Montag" { echo "Mon" }  
 default { echo "Was anderes" }  
 }  
**do** { <Anweisungsliste> } **while** { <Bedingung> }  
**while** { <Bedingung> } **do** { <Anweisungsliste> }  
**try** { kommando > \$NULL } **catch** { }; **if** { ! \$? } { ... }

**Abbrüche:** **break** (Schleifenabbruch), **continue** (aktuellen Schleifendurchlauf abbrechen),  
**return** (zurück zum aufrufenden Programm), **exit** (Programmabbruch)

### Passwörter, Pausen

\$pw = Read-Host "Passwort eingeben" -AsSecureString  
 \$pw = ConvertTo-SecureString -AsPlainText -Force -String "EinTest.11"  
**Dekodieren:**  
 \$M = [Runtime.InteropServices.Marshal]  
 \$M::PtrToStringAuto(\$M::SecureStringToBSTR(\$pw))  
**Pause:**  
**Start-Sleep** -m 500 500 ms Pause

## ActiveDirectory

### AD-Tools

#### ActiveDirectory-Module installieren/deinstallieren:

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

#### Active-Directory als Provider (Get-PSPProvider):

**cd ad:** Ins Active-Directory wechseln  
**ls** Aktuelles AD-Verzeichnis auflisten  
**cd "DC=bib,DC=de"** In Domäne wechseln  
**cd "CN=Users"** Relativ 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**).DNSRoot 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:

\$pw = ConvertTo-SecureString -AsPlainText -Force -String "EinTest.11"  
**New-ADUser** -name "Stefan Menne" -samAccountName dozmen  
 -UserPrincipalName dozmen@bib.de -AccountPassword \$pw  
 -Path "CN=Users,DC=bib,DC=de" -enabled \$true

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

#### User-Daten löschen/verändern/ergänzen:

User selektieren über: samAccountName, DistinguishedName oder SID  
**Remove-ADUser dozmen** -Confirm:\$false  
**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"  
 -ProtectedFromAccidentalDeletion \$false  
**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:

**Test-Path** "AD:\CN=Gast,CN=Users,DC=bib,DC=de"

### DHCP

#### 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** -Scopelid 192.168.10.0

#### DHCP-Server an Netzwerkkarte 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*
Rename-Computer -NewName "win2" -Restart
$a = Import-Csv ServerNames.csv -Header OldName, NewName
Foreach ( $Server in $a ) {Rename-Computer -ComputerName $Server.OldName -
  NewName $Server.NewName -DomainCredential Domain01\Admin01 -Force -
  Restart}
```

*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](mailto:Stefan.Menne@gmx.de)

### Gruppenrichtlinien

```
Import-Module GroupPolicy -verbose
Get-Command *GP* -commandtype cmdlet
Get-GPO -name "Default Domain Policy"
Get-GPO -name "Default Domain Controllers Policy"
```

### Alternative AD-Zugriffe

WMI: Einige WMI-Objekte liefern auch AD-Objekte  
Win32\_Account, Win32\_UserAccount, Win32\_Group

ADSI (Active Directory Service Interface): Das ADSI (activeds.dll) erlaubt den kompletten Zugriff auf den LDAP-Server (LDAP://adsldp.dll) und die WinNT-Datenbank (WINNT://adsnt.dll).  
In der Powershell können Sie auf die ADSI-COM-Komponente zugreifen,  
Bsp:  
[ADSI]" "  
[ADSI]"LDAP://OU=Test,DC=bib,DC=local"  
\$ad = [ADSI]"LDAP://CN=Users,DC=bib,DC=local"  
\$ad.get\_children()

.NET-Klasse: `System.DirectoryServices`

Diese Klasse liefert leider nicht alle Funktionen des ADSI, hat aber einige Funktionen, die das Arbeiten mit Powershell erleichtern. ADSI ist der Untergrund dieser Klasse. Bsp:

```
$AD2 = new-object System.DirectoryServices.DirectoryEntry
Relevante .NET-Klassen:
```

TypeShortcut	Full Classname
[Adsi]	[System.DirectoryServices.DirectoryEntry]
[wmi]	[System.Management.ManagementObject]
[wmiClass]	[System.Management.ManagementClass]
[wmiSearcher]	[System.Management.ManagementObjectSearcher]