

5

TRAINING



Windows PowerShell

Administration automatisieren

ppedv AG

Stefan Ober

5

TRAINING



Vorab

Agenda

- Einführung
- Commandlets
- ISE
- Module
- Variablen
- Operationen
- Pipeline
- Formatierung
- Sortieren, Messen,
Selektieren
- Konvertieren, Importieren,
Exportieren
- Filtern
- Aufzählen
- WMI/CIM
- Skripting
- Funktionen
- Verzweigungen
- Fehlerbehandlung

5

TRAINING



Einleitung

Was ist PowerShell?

- Windows PowerShell ist...
 - Die vielleicht beste Skript-Sprache für Windows
 - Vielseitig einsetzbar
 - Leicht zu erlernen
 - Sehr umfangreich
 - Bereits seit längerem funktionale Grundlage vieler GUIs
 - So ein bisschen wie Linux



Etwas genauer bitte...

- Skript- & Kommandosprache
- (Bessere) Alternative zur CMD
- Wurde 2006 eingeführt
- Bereits seit Windows 7 / Server 2008 R2 fester Bestandteil der Microsoft-Betriebssysteme
- Nachinstallierbar (WMF)
- Objektorientiert
- Mitgelieferte IDE inkl. Debugger (ISE)
- Basiert auf .NET Framework

Windows PowerShell Versionen

| | PowerShell 2 | PowerShell 3 | PowerShell 4 | PowerShell 5 |
|--|----------------|-----------------|----------------|----------------|
| Windows XP | Verfügbar | - | - | - |
| Windows Server 2003 | Verfügbar | - | - | - |
| Windows Vista | Verfügbar | - | - | - |
| Windows Server 2008 | Verfügbar | Verfügbar (SP2) | - | - |
| Windows 7 | Vorinstalliert | Verfügbar (SP1) | Verfügbar | Verfügbar |
| Windows Server 2008 R2 | Vorinstalliert | Verfügbar (SP2) | Verfügbar | Verfügbar |
| Windows 8 / Server 2012 | Verfügbar | Vorinstalliert | Verfügbar | Verfügbar |
| Windows 8.1 / Server 2012 R2 | Verfügbar | - | Vorinstalliert | Verfügbar |
| Windows 10 / Server 2016 & 2019 | Verfügbar | - | | Vorinstalliert |

Windows PowerShell 1 + 2 benötigen .NET Framework 2.0

Windows PowerShell 3 benötigt .NET Framework 4.0

Windows PowerShell 4 benötigt .NET Framework 4.5

Windows PowerShell 5.1/5.2 benötigt .NET Framework 4.6/4.7

Windows PowerShell Versionen

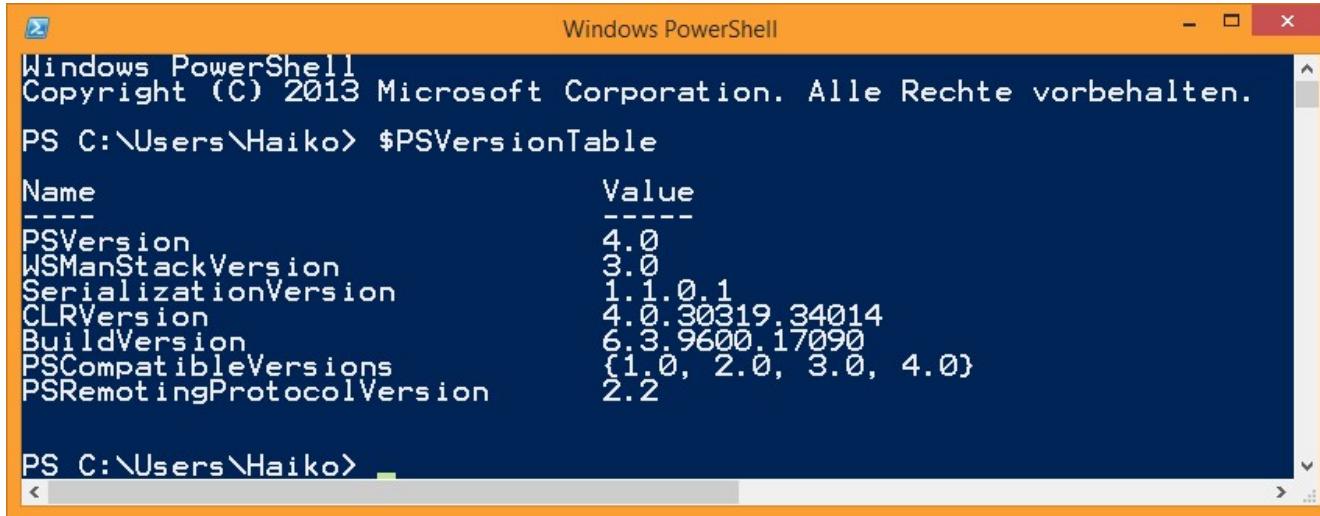
- Windows PowerShell kommt in einer Kern-Ausstattung
- beinhaltet je nach Version grundlegende Funktionen und einige wenige Befehle
- Großteil der Befehle stammt aus Erweiterungen – sog. Module
- Module sind für bestimmte Version konzipiert
- Module sind nicht Bestandteil einer PowerShell-Version

Windows PowerShell Versionen

- Sie werden aber z. T. mit bestimmten OS-Versionen geliefert
- Ergebnis: Das bloße Nachinstallieren einer neuen PS-Version liefert nicht automatisch den vollen Umfang der neueren Version auf einem neueren OS!

Windows PowerShell Versionen

- **\$PSVersionTable** zeigt PS-Version und unterstützt Umgebungen:



A screenshot of a Windows PowerShell window titled "Windows PowerShell". The output shows the contents of the \$PSVersionTable variable:

```
Windows PowerShell
Copyright (C) 2013 Microsoft Corporation. Alle Rechte vorbehalten.

PS C:\Users\Haiko> $PSVersionTable

Name                           Value
----                           ---
PSVersion                      4.0
WSManStackVersion               3.0
SerializationVersion             1.1.0.1
CLRVersion                      4.0.30319.34014
BuildVersion                     6.3.9600.17090
PSCompatibleVersions              {1.0, 2.0, 3.0, 4.0}
PSRemotingProtocolVersion        2.2

PS C:\Users\Haiko>
```

- PowerShell 1.0: Leeres Ergebnis

Windows PowerShell Versionen

- **powershell –version 2.0** started Windows PowerShell mit der PowerShell 2.0 Engine
 - Abwärts-Kompatibilität
- Auf 64Bit-System: 32Bit- und 64Bit-PowerShell
- PowerShell gibt es auch “elevated” (mit Admin-Rechten)



Windows PowerShell

- Es existieren mehrere Anwendungen:
 - Konsole
 - ISE
 - 3rd-Party-Anwendungen
- Konsole
 - Command-Line-Interface
 - Alle PS-Features, aber eingeschränkte Bearbeitung
- ISE
 - Maximale Bearbeitung, aber eingeschränkte Features

Commandlets

Commandlets

- Eigentlich „cmdlet“
- Viele bekannte Kommandos:
 - dir
 - cd
 - mkdir
 - ...
- Das sind aber keine Commandlets!
- ping, ipconfig und ähnliche: funktioniert i. Allg. (evtl. mit .exe!)

Commandlets

- Commandlets haben immer gleichen Aufbau:
 - **VERB-SUBSTANTIV** (Verb-Noun)
 - z. B. Get-Command, Set-Location, Start-Service, ...
- Jedes Commandlet hat eine eigene Hilfe
 - **Get-Help COMMANDLET** (oder auch „help“ oder „man“)
 - Diverse Schalter: **-Detailed, -Examples, -Full, -Online**

Commandlets

- Ab PowerShell 3.0: Hilfe wird nicht mehr offline mitgeliefert
- Muss einmalig heruntergeladen werden
- **Update-Help** prüft auf herunterladbare Inhalte
- **Save-Help** speichert Hilfe für Offline-Rechner

EH1 Save-Help -DestinationPath "C:\ps_help"

Eik Hitscherich; 21.08.2015

EH2 Pfad muss existieren!

Eik Hitscherich; 21.08.2015

Demo: Commandlets und Hilfe

- Anzeige der verfügbaren Commandlets
- Arbeiten mit der Hilfe



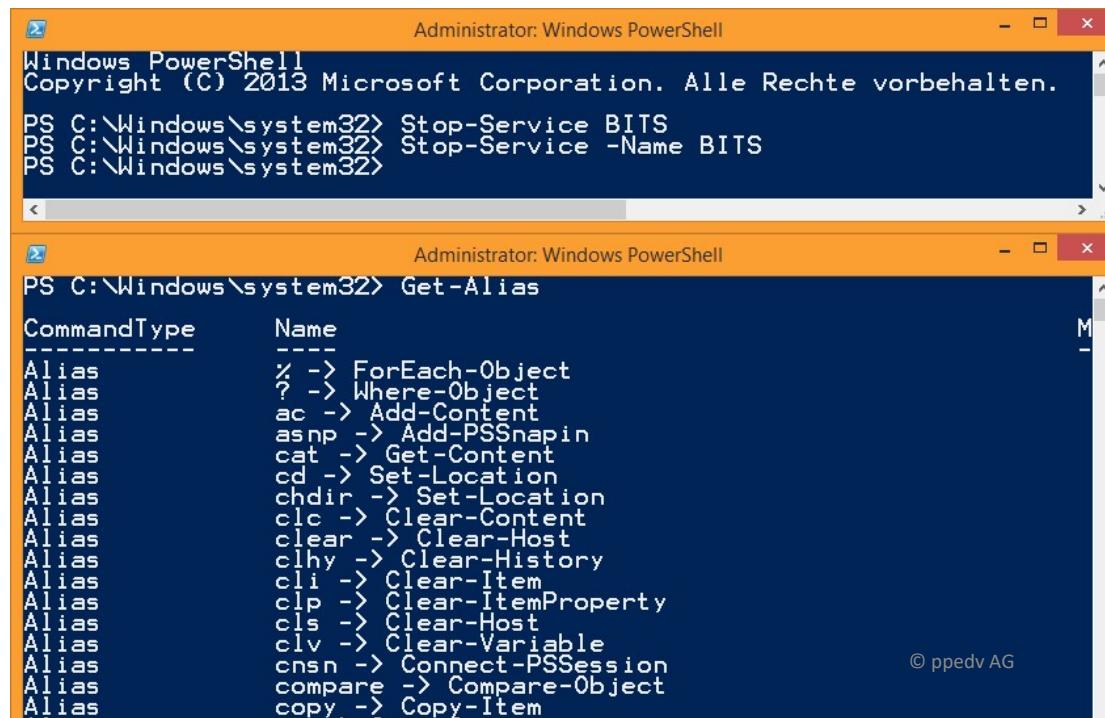
Demo: Commandlets finden

- Aus bereits bekannten Commandlets lassen sich oft verwandte Befehle ableiten:
 - **Get-Command –Noun**
 - **Get-Command –Verb**
 - **Get-Command *wort***
- Alternativ kann man sich alle Commandlets eines Modules auflisten lassen



Ausführen von Commandlets

- Kurzformen vs. Volle Syntax
 - Aliase
 - Parameter ohne Namen



The screenshot shows two separate Windows PowerShell windows. The top window is titled 'Administrator: Windows PowerShell' and contains the following commands:

```
Windows PowerShell
Copyright (C) 2013 Microsoft Corporation. Alle Rechte vorbehalten.

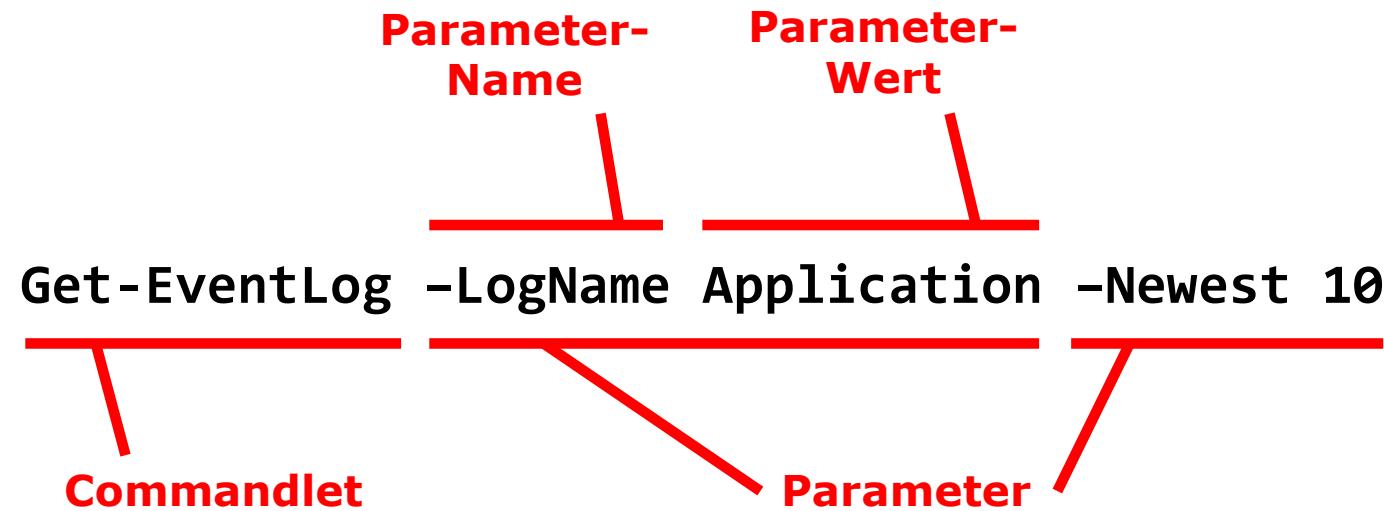
PS C:\Windows\system32> Stop-Service BITS
PS C:\Windows\system32> Stop-Service -Name BITS
PS C:\Windows\system32>
```

The bottom window is also titled 'Administrator: Windows PowerShell' and displays the output of the command 'Get-Alias':

```
PS C:\Windows\system32> Get-Alias

CommandType      Name
----           ---
Alias           % -> ForEach-Object
Alias           ? -> Where-Object
Alias           ac -> Add-Content
Alias           asnp -> Add-PSSnapin
Alias           cat -> Get-Content
Alias           cd -> Set-Location
Alias           chdir -> Set-Location
Alias          clc -> Clear-Content
Alias           clear -> Clear-Host
Alias           clhy -> Clear-History
Alias           cli -> Clear-Item
Alias           clp -> Clear-ItemProperty
Alias           cls -> Clear-Host
Alias           clv -> Clear-Variable
Alias           cnsn -> Connect-PSSession
Alias           compare -> Compare-Object
Alias           copy -> Copy-Item
```

Ausführen von Commandlets



Ausführen von Commandlets

- Mehrere Arten von Parametern:

- Benannter Parameter

- z. B. Get-ChildItem -Path C:\

- Switch-Parameter

- z. B. Get-ChildItem -Path C:\ -Recurse

- Positionaler Parameter

- z. B. Get-ChildItem C:\Windows *.log
 - Sollten nicht in Skripten oder ähnlichem genutzt werden
 - Nur für Direkteingaben (Lesbarkeit & Verlässlichkeit)

Ausführen von Commandlets

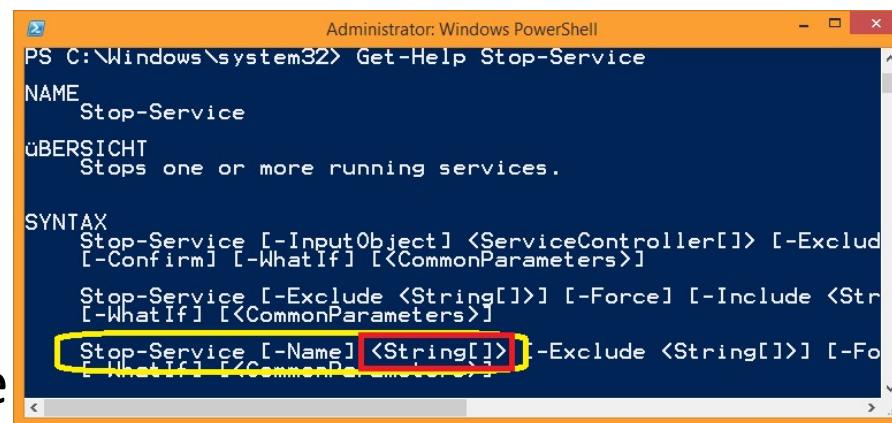
- Leerzeichen trennt
 - Commandlets und ihre Parameter
 - Parameter und ihre Werte
- Wenn ein Leerzeichen gefordert oder erlaubt ist, können auch mehrere verwendet werden
- Groß-/Kleinschreibung spielt in der Regel keine Rolle

Ausführen von Commandlets

- Einem Parameter können auch mehrere Werte übergeben werden:

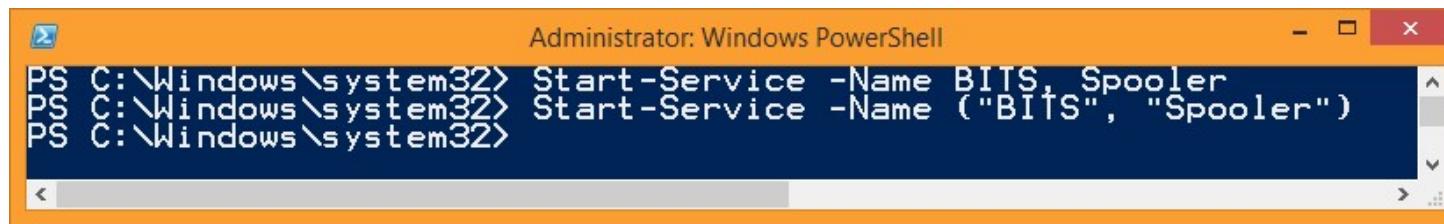
-Name <String[]>

- Dazu erzeugt man eine Liste



```
Administrator: Windows PowerShell
PS C:\Windows\system32> Get-Help Stop-Service
NAME
    Stop-Service
ÜBERSICHT
    Stops one or more running services.

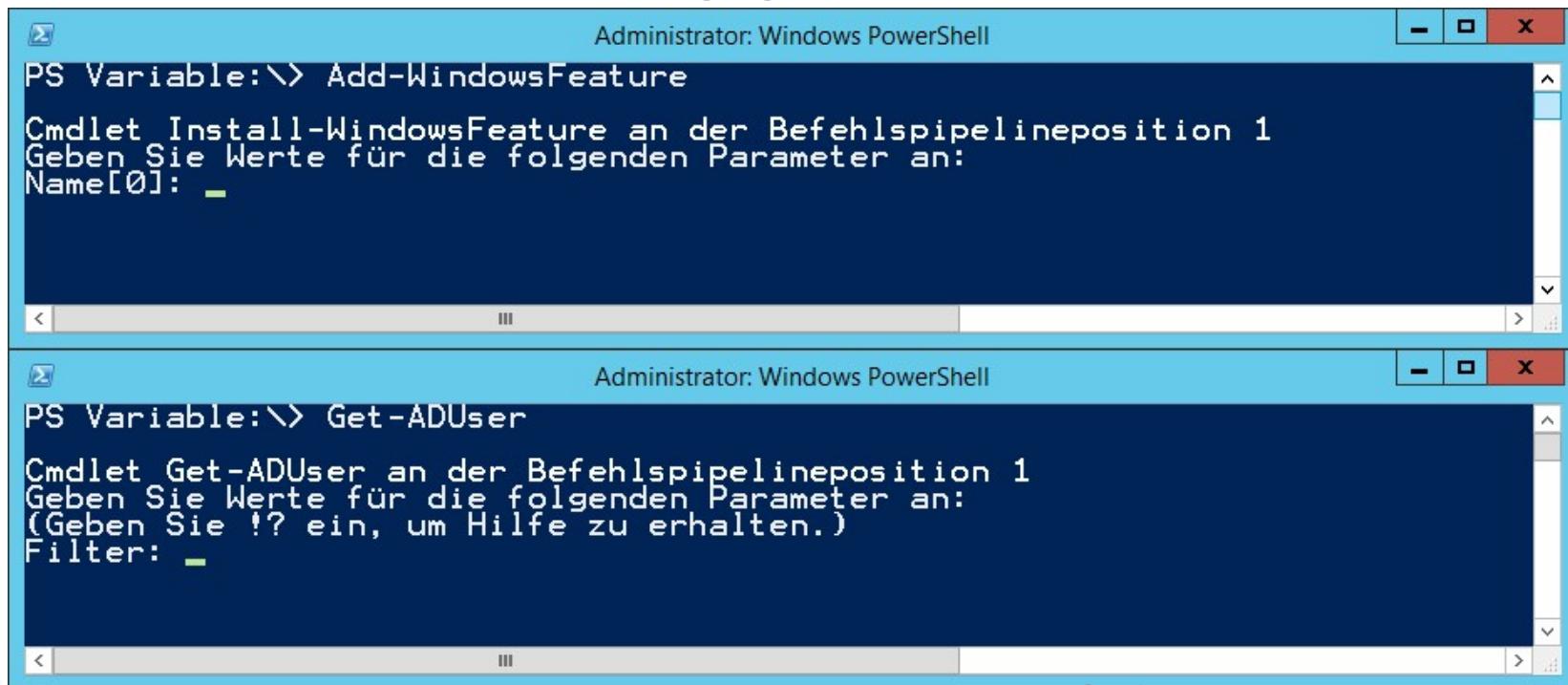
SYNTAX
    Stop-Service [-InputObject] <ServiceController[]> [-Exclude
        [-Confirm] [-WhatIf] [<CommonParameters>]
    ]
    Stop-Service [-Exclude <String[]>] [-Force] [-Include <Str
        [-WhatIf] [<CommonParameters>]
    ]
    Stop-Service [-Name] <String[]> [-Exclude <String[]>] [-Fo
        [-WhatIf] [<CommonParameters>]
```



```
Administrator: Windows PowerShell
PS C:\Windows\system32> Start-Service -Name BITS, Spooler
PS C:\Windows\system32> Start-Service -Name ("BITS", "Spooler")
PS C:\Windows\system32>
```

Ausführen von Commandlets

- Wenn ein Commandlets einen oder mehrere Parameter erwartet (Pflicht-Parameter), der nicht angegeben wurde:



The screenshot displays two separate Windows PowerShell windows, both titled "Administrator: Windows PowerShell".

The top window shows the command `PS Variable:\> Add-WindowsFeature`. The output indicates that the cmdlet `Install-WindowsFeature` was found at pipeline position 1, and it prompts for values for the following parameters: `Name[0]:` followed by a blank line.

The bottom window shows the command `PS Variable:\> Get-ADUser`. The output indicates that the cmdlet `Get-ADUser` was found at pipeline position 1, and it prompts for values for the following parameters: `(Geben Sie !? ein, um Hilfe zu erhalten.) Filter:` followed by a blank line.

Ausführen von Commandlets

- Verringern des Schreibaufwandes durch:
 - Aliase statt der vollen Cmdlet-Namen
 - Positionale Parameter
 - Verkürzte Parameter-Namen
- Verschlechtert Lesbarkeit
- Sollte in Veröffentlichungen oder Scripts vermieden werden



Ausführen von Commandlets

```
Get-Service -Name BITS -ComputerName WIN2012
```

Alias

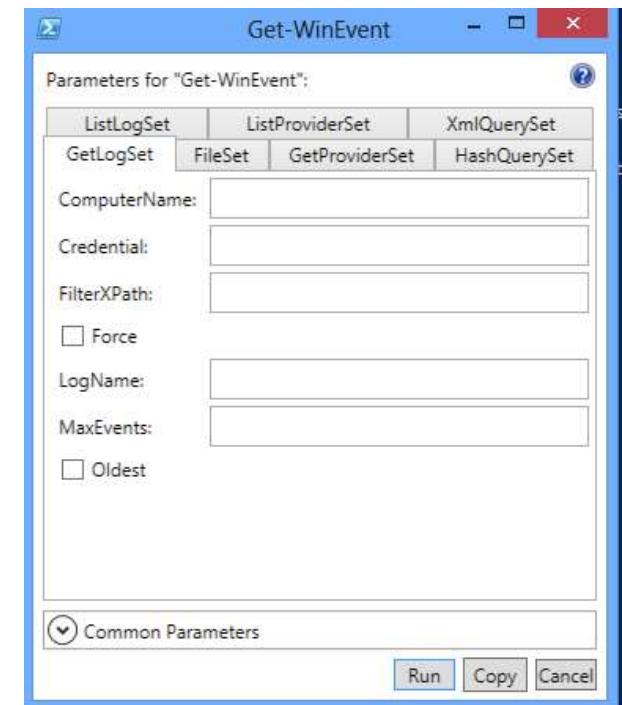
Positionaler
Parameter

Verkürzter
Parameter-
Name

```
gsv BITS -Comp WIN2012
```

Ausführen von Commandlets

- Show-Command zeigt GUI mit allen möglichen Parametern
- Werte z. T. via Dropdown-Liste
- Hauptsächlich während der Lern-Phase nützlich



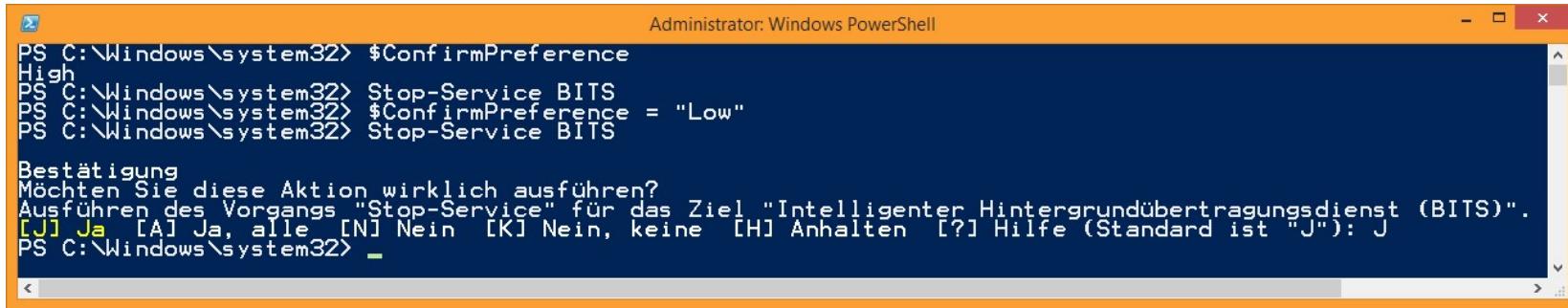
Ausführen von Commandlets

- Commandlets mit Änderungen:
 - Confirm
 - Nochmalige Abfrage vor Ausführen
 - WhatIf
 - Was WÜRDE passieren (nützlich bei unklarer Menge oder längeren Verkettungen)

Ausführen von Commandlets

- Es gibt 3 Stufen des Einflusses eines Commandlets auf das System:
 - High
 - Medium
 - Low
- **\$ConfirmPreference** regelt, wann Abfrage automatisch kommt
- Standard: „High“ -> Keine automatische Abfrage

Ausführen von Commandlets



The screenshot shows an Administrator Windows PowerShell window. The command entered is:

```
PS C:\Windows\system32> $ConfirmPreference  
High  
PS C:\Windows\system32> Stop-Service BITS  
PS C:\Windows\system32> $ConfirmPreference = "Low"  
PS C:\Windows\system32> Stop-Service BITS
```

Following this, a confirmation dialog box appears:

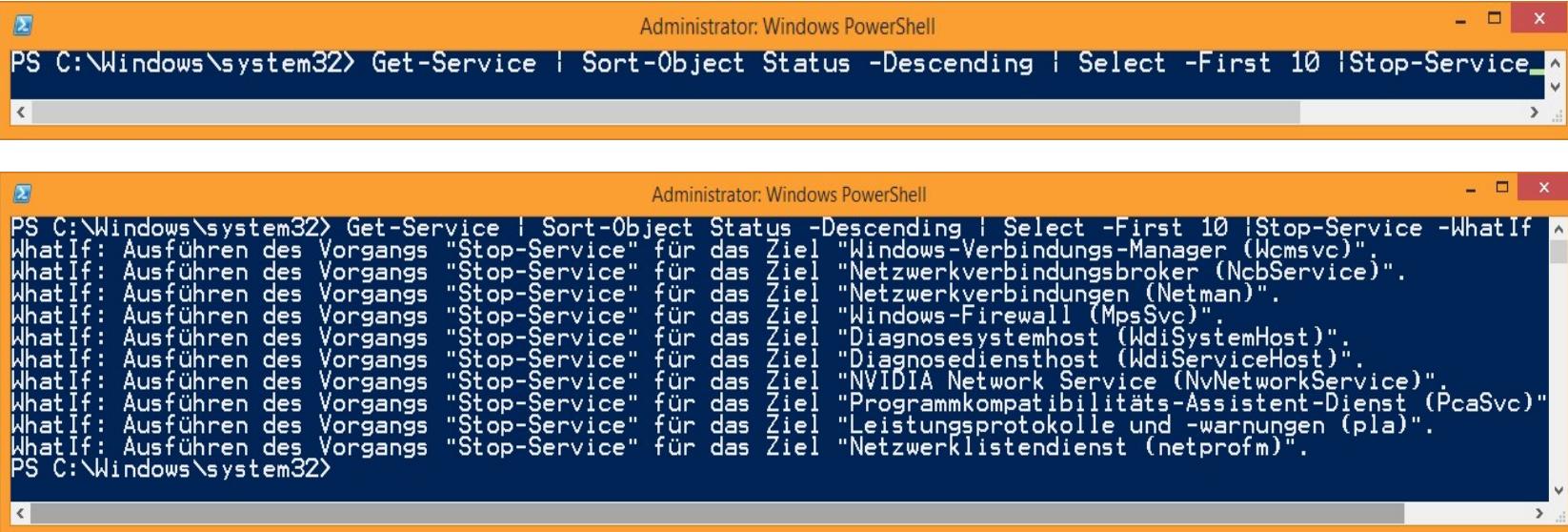
Bestätigung
Möchten Sie diese Aktion wirklich ausführen?
Ausführen des Vorgangs "Stop-Service" für das Ziel "Intelligenter Hintergrundübertragungsdienst (BITS)".
[J] Ja [A] Ja, alle [N] Nein [K] Nein, keine [H] Anhalten [?] Hilfe (Standard ist "J"): J

The user has selected 'Ja' (Yes) and pressed Enter.

- Confirm kann schädlich bei Skripten sein, die nicht-interaktiv laufen

Ausführen von Commandlets

- „WhatIf“ listet nur die Änderungen auf, ohne sie direkt auszuführen:



The image shows two side-by-side Windows PowerShell windows. Both are titled "Administrator: Windows PowerShell" and are running on the command line: PS C:\Windows\system32> Get-Service | Sort-Object Status -Descending | Select -First 10 | Stop-Service.

The top window shows the command being run without the -WhatIf parameter. It outputs a list of ten services that will be stopped: Windows-Verbindungs-Manager (Wcmsvc), Netzwerkverbindungsbroker (NcbService), Netzwerkverbindungen (Netman), Windows-Firewall (MpsSvc), Diagnosesystemhost (WdiSystemHost), Diagnosediensthost (WdiServiceHost), NVIDIA Network Service (NvNetworkService), Programmkompatibilitäts-Assistent-Dienst (PcaSvc), Leistungsprotokolle und -warnungen (pla), and Netzwerklistendienst (netprofm).

The bottom window shows the same command with the -WhatIf parameter added: PS C:\Windows\system32> Get-Service | Sort-Object Status -Descending | Select -First 10 | Stop-Service -WhatIf. This command lists the same ten services but does not actually stop them, instead displaying a series of "WhatIf" messages for each service, indicating what would happen if the service were stopped.

Ausführen von Commandlets

- **\$WhatIfPreference** regelt Verhalten
- Standard: „False“
- Bei „True“ wird jede Aktion behandelt, als wäre sie mit **-WhatIf** ausgeführt worden
- Dann kann mit **-WhatIf:\$false** ein einzelner Aufruf ohne WhatIf getätigt werden

Demo: Ausführen von Commandlets

- Anzeigen der Hilfe für ein Commandlet
- Arbeiten mit Parametern
- Arbeiten mit Aliasen
- Funktionsweise von WhatIf



Übung: Commandlets

1. Listen Sie alle Commandlets auf, die mit dem „Computer“ arbeiten
2. Lassen Sie sich die Beispiele für „Set-Service“ anzeigen
3. Welche PS-Kommandos verbergen sich hinter „dir“ und „cd“?
4. Testen Sie WhatIf an folgendem Aufruf:

```
Get-Childitem -Path C:\Testfiles\ -Filter *.txt –Recurse | Remove-Item
```



Mögliche Lösungen

1. Get-Command -Noun Computer
2. Get-Help Set-Service -Examples
Hier sollte ein Download der Hilfe nötig sein!
3. Get-Alias dir bzw. Get-Alias cd (Alternativ: Get-Help dir)
4. Get-Childitem C:\Testfiles*.txt -Recurse | Remove-Item -WhatIf

.NET-Framework 2 installieren...

Install-WindowsFeature

- Name NET-Framework-Core
- Source D:\sources\sxs

5
★

TRAINING



Module

Module

- PowerShell-Commandlets sind in der Regel in Modulen organisiert
- Module müssen geladen werden, um Cmdlets zu nutzen
- Ab PS 3.0 werden Module automatisch geladen

Module

```
Windows PowerShell
Copyright (C) 2013 Microsoft Corporation. Alle Rechte vorbehalten.
PS C:\Users\Haiko> Get-Module

ModuleType Version Name ExportedCommands
---- -- -- -----
Manifest 3.1.0.0 Microsoft.PowerShell.Management {Add-Computer, Add-Content, Che

PS C:\Users\Haiko> -
```



```
Windows PowerShell
PS C:\Users\Haiko> Get-Module -ListAvailable

Verzeichnis: C:\SkyDrive\Dokumente\WindowsPowerShell\Modules

ModuleType Version Name ExportedCommands
---- -- -- -----
Script 0.0 DotNet {Get-CommandWithType, Copy-ToZip, Get-DuplicateFile, Add-ForeachStatement, Add-IfSt
Script 1.0 FileSystem {Add-ChildControl, Add-CodeGene
Script 1.1 IsePack {New-Enum, New-PInvoke, New-Scri
Script 1.0 PowerShellPack {Add-CropFilter, Add-RotateFlip
Script 1.0 PSCodeGen {Get-Article, Get-Feed, New-Fee
Script 1.0 PSImageTools {Get-BootStatus, Get-DisplaySet
Script 1.0 PSSystemTools {Get-CurrentUser, Get-Everyone,
Script 1.0 PSUserTools {Add-TaskAction, Add-TaskTrigge
Script 1.0 TaskScheduler {Add-ChildControl, Add-CodeGene
Script 1.0 WPK {Get-AppBackgroundTaskDiagn
Get-AppLockerFileInformation, Add-AppxPackage, Get-AppxPac
Get-AssignedAccess, Get-Assi
Unlock-BitLocker, Suspend-BitL
```



```
Verzeichnis: C:\Windows\system32\WindowsPowerShell\v1.0\Modules

ModuleType Version Name ExportedCommands
---- -- -- -----
Manifest 1.0.0.0 AppBackgroundTask {Disable-AppBackgroundTaskDiagn
Manifest 2.0.0.0 AppLocker {Get-AppLockerFileInformation, Add-AppxPackage, Get-AppxPac
Manifest 2.0.0.0 Appx {Get-AppBackgroundTaskDiagn
Script 1.0.0.0 AssignedAccess {Get-AppLockerFileInformation, Add-AppxPackage, Get-AppxPac
Manifest 1.0.0.0 BitLocker {Get-AppBackgroundTaskDiagn
```

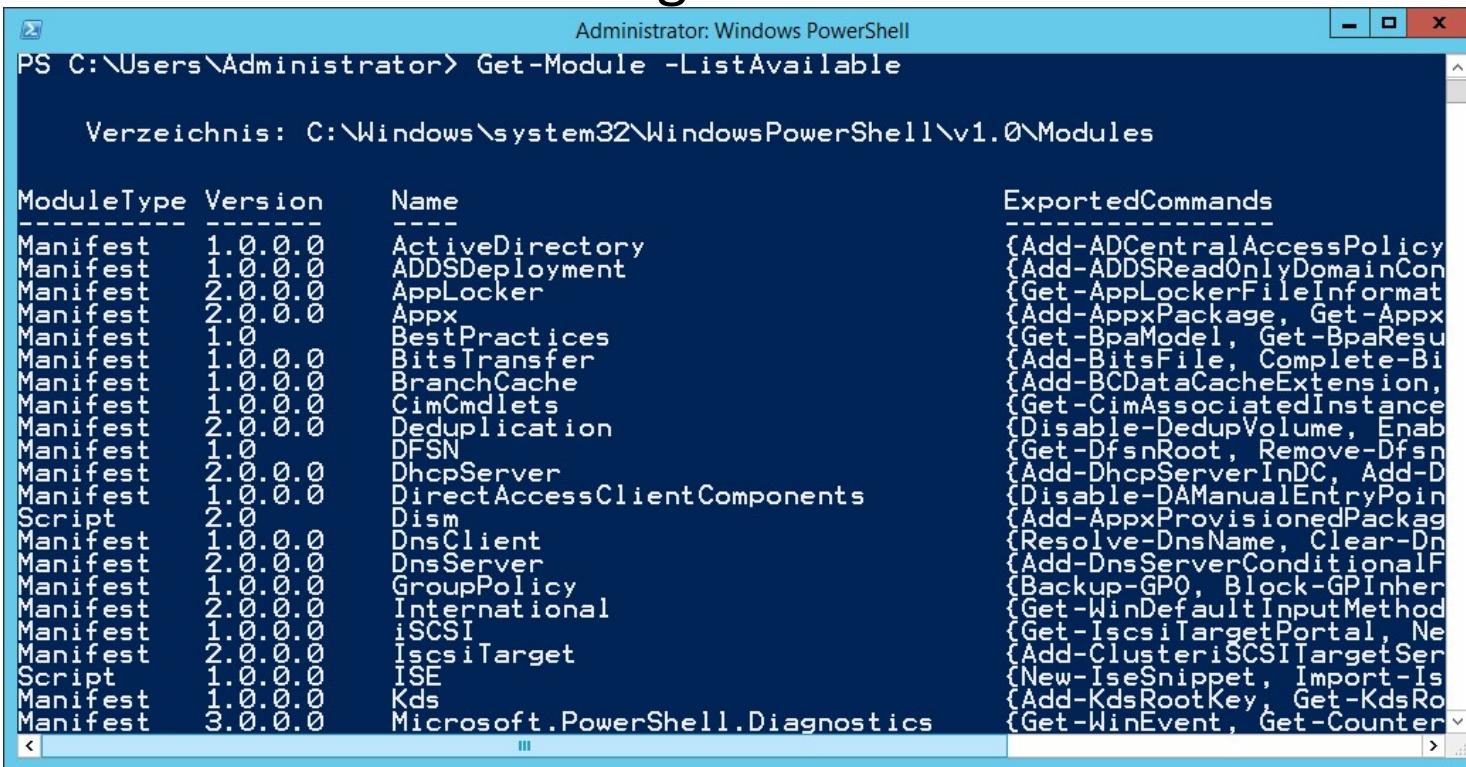
Module

- Module werden mit **Import-Module** geladen
- Alias: **ipmo**
- Auch bei neuerer PS-Version: In Scripten Module manuell laden!
(Abwärtskompatibilität)



Module

- Auf Windows Server mit einigen Rollen:



| ModuleType | Version | Name | ExportedCommands |
|------------|---------|----------------------------------|---|
| Manifest | 1.0.0.0 | ActiveDirectory | {Add-ADCentralAccessPolicy, Add-ADDSReadonlyDomainCon |
| Manifest | 1.0.0.0 | ADDSDeployment | {Get-AppLockerFileInformat |
| Manifest | 2.0.0.0 | AppLocker | {Add-AppxPackage, Get-Appx |
| Manifest | 2.0.0.0 | Appx | {Get-BpaModel, Get-BpaResu |
| Manifest | 1.0 | BestPractices | {Add-BitsFile, Complete-Bi |
| Manifest | 1.0.0.0 | BitsTransfer | {Add-BCDataCacheExtension, |
| Manifest | 1.0.0.0 | BranchCache | {Get-CimAssociatedInstance |
| Manifest | 1.0.0.0 | CimCmdlets | {Disable-DedupVolume, Enab |
| Manifest | 2.0.0.0 | Deduplication | {Get-DfsnRoot, Remove-Dfsn |
| Manifest | 1.0 | DFSN | {Add-DhcpServerInDC, Add-D |
| Manifest | 2.0.0.0 | DhcpServer | {Disable-DAManualEntryPoin |
| Manifest | 1.0.0.0 | DirectAccessClientComponents | {Add-AppxProvisionedPackag |
| Script | 2.0 | Dism | {Resolve-DnsName, Clear-Dn |
| Manifest | 1.0.0.0 | DnsClient | {Add-DnsServerConditionalF |
| Manifest | 2.0.0.0 | DnsServer | {Backup-GPO, Block-GPIher |
| Manifest | 1.0.0.0 | GroupPolicy | {Get-WinDefaultInputMethod |
| Manifest | 2.0.0.0 | International | {Get-IscsiTargetPortal, Ne |
| Manifest | 1.0.0.0 | iSCSI | {Add-ClusteriSCSITargetSer |
| Manifest | 2.0.0.0 | IscsiTarget | {New-IseSnippet, Import-Is |
| Script | 1.0.0.0 | ISE | {Add-KdsRootKey, Get-KdsRo |
| Manifest | 1.0.0.0 | Kds | {Get-WinEvent, Get-Counter |
| Manifest | 3.0.0.0 | Microsoft.PowerShell.Diagnostics | |

Übung: Module

1. Lassen Sie sich alle geladenen Module auflisten
2. Führen Sie „Get-ADUser -Filter *“ aus
3. Lassen Sie sich erneut alle geladenen Module auflisten
4. Lassen Sie sich alle verfügbaren Module auflisten



Mögliche Lösungen

1. Get-Module
2. Get-ADUser -Filter *
3. Get-Module
4. Get-Module -ListAvailable

Die Pipeline

Die Pipeline

- PowerShell Kommandos werden in einer *Pipeline* ausgeführt
- Pipeline kann ein oder mehrere Commandlets enthalten
- Mehrere Commandlets werden durch „|“ verknüpft
- Commandlets werden von links nach rechts ausgeführt
- Auf der Konsole wird die Ausgabe des letzten Commandlets gezeigt

Die Pipeline

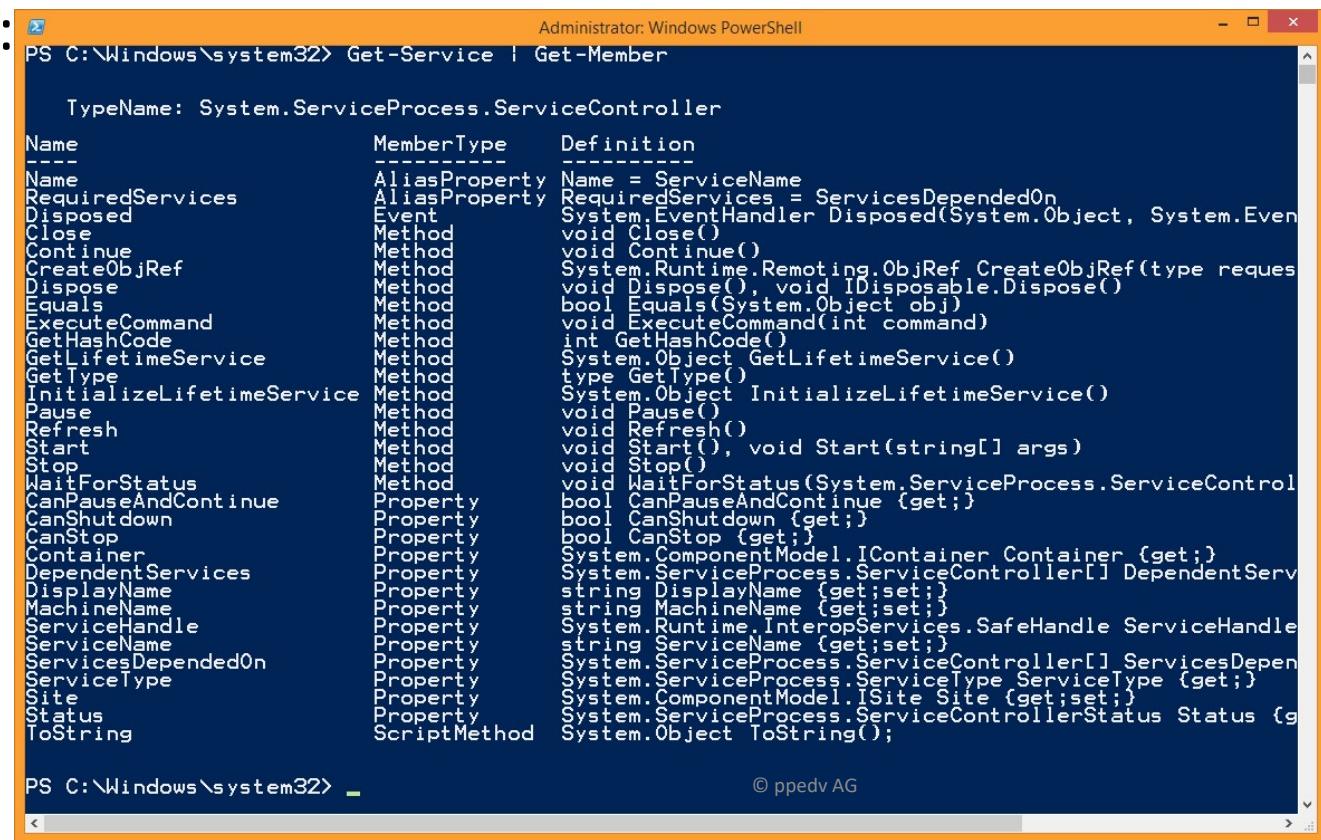
- Bsp.: **Get-Service | Out-File Services.txt**
- Das ist also die Kombination aus **Get-Service** und **Out-File**
(Hier wird dann auch keine Ausgabe erzeugt!)
- In diesem Fall entspricht das etwa dem „> Services.txt“ aus DOS
- Funktioniert aber anders!

Die Pipeline

- Windows PowerShell ist objektorientiert
- Ausgabe von Commandlets erzeugt in der Regel Objekte
- Diese werden meist in Textform dargestellt
- Objekte können enthalten:
 - Eigenschaften („Properties“)
 - Methoden („Methods“)
 - Ereignisse („Events“)

Die Pipeline

- Commandlets, die Objekte generieren, können an **Get-Member** übergeben werden:



Administrator: Windows PowerShell

```
PS C:\Windows\system32> Get-Service | Get-Member
```

| Name | MemberType | Definition |
|---------------------------|---------------|---|
| --- | ----- | ----- |
| Name | AliasProperty | Name = ServiceName |
| RequiredServices | AliasProperty | RequiredServices = ServicesDependedOn |
| Disposed | Event | System.EventHandler Disposed(System.Object, System.EventArgs) |
| Close | Method | void Close() |
| Continue | Method | void Continue() |
| CreateObjRef | Method | System.Runtime.Remoting.ObjRef CreateObjRef(type requires) |
| Dispose | Method | void Dispose(), void IDisposable.Dispose() |
| Equals | Method | bool Equals(System.Object obj) |
| ExecuteCommand | Method | void ExecuteCommand(int command) |
| GetHashCode | Method | int GetHashCode() |
| GetLifetimeService | Method | System.Object GetLifetimeService() |
| GetType | Method | type GetType() |
| InitializeLifetimeService | Method | System.Object InitializeLifetimeService() |
| Pause | Method | void Pause() |
| Refresh | Method | void Refresh() |
| Start | Method | void Start(), void Start(string[] args) |
| Stop | Method | void Stop() |
| WaitForStatus | Method | void WaitForStatus(System.ServiceProcess.ServiceControl) |
| CanPauseAndContinue | Property | bool CanPauseAndContinue {get;} |
| CanShutdown | Property | bool CanShutdown {get;} |
| CanStop | Property | bool CanStop {get;} |
| Container | Property | System.ComponentModel.IContainer Container {get;} |
| DependentServices | Property | System.ServiceProcess.ServiceController[] DependentServices |
| DisplayName | Property | string DisplayName {get;set;} |
| MachineName | Property | string MachineName {get;set;} |
| ServiceHandle | Property | System.Runtime.InteropServices.SafeHandle ServiceHandle |
| ServiceName | Property | string ServiceName {get;set;} |
| ServicesDependedOn | Property | System.ServiceProcess.ServiceController[] ServicesDependedOn |
| ServiceType | Property | System.ServiceProcess.ServiceType ServiceType {get;} |
| Site | Property | System.ComponentModel.ISite Site {get;set;} |
| Status | Property | System.ServiceProcess.ServiceControllerStatus Status {get;} |
| ToString | ScriptMethod | System.Object ToString(); |

```
PS C:\Windows\system32>
```

© ppedv AG

Die Pipeline

- Pipeline ermöglicht
 - komplexe Aufgaben in sehr kurzen Aufrufen (dazu später mehr)
 - Formatierung der Ausgabe
 - Filterung
 - Sortierung
 - uvm.

Die Pipeline – Formatierung der Ausgabe

- Verschiedene Cmdlets, die zur Formatierung der Ausgabe genutzt werden können:
 - **Format-Table** (Standard) („ft“)
 - **Format-List** („fl“)
 - **Format-Wide** („fw“)

Die Pipeline – Formatierung der Ausgabe

```
PS C:\Users\Haiko> Get-Service
```

| Status | Name | DisplayName |
|---------|------------------|-----------------------------------|
| Stopped | Adobe LM Service | Adobe LM Service |
| Running | AdobeARMservice | Adobe Acrobat Update Service |
| Stopped | AeLookupSvc | Anwendungserfahrung |
| Running | ALG | Gatewaydienst auf Anwendungsebene |
| Stopped | AppIDSvc | Anwendungsidentität |
| Stopped | Appinfo | Anwendungsinformationen |


```
PS C:\Users\Haiko> Get-Service | Format-List
```

| Name | DisplayName | Status | DependentServices | ServicesDependedOn | CanPauseAndContinue | CanShutdown | CanStop | ServiceType |
|------|--------------------|-----------|-------------------|--------------------|---------------------|-------------|---------|-------------------|
| | : Adobe LM Service | : Stopped | : {} | : {} | : False | : False | : False | : Win32OwnProcess |
| | : AdobeARMservice | : Running | | | | | | |


```
PS C:\Users\Haiko> Get-Service | Format-Table Name,Status
```

| Name | Status |
|------------------|---------|
| Adobe LM Service | Stopped |
| AdobeARMservice | Running |
| AeLookupSvc | Stopped |
| ALG | Running |



| Status | Name | DisplayName |
|---------|----------------------|-------------------------------------|
| Stopped | Adobe LM Service | Adobe LM Service |
| Running | AdobeARMservice | Adobe Acrobat Update Service |
| Stopped | AeLookupSvc | Anwendungserfahrung |
| Running | ALG | Gatewaydienst auf Anwendungsebene |
| Stopped | AppIDSvc | Anwendungsidentität |
| Stopped | Appinfo | Anwendungsinformationen |
| Running | Apple Mobile Device | Apple Mobile Device |
| Stopped | AppMgmt | Anwendungsverwaltung |
| Stopped | AppReadiness | App-Vorbereitung |
| Stopped | AppXSvc | AppX-Bereitstellungsdiens (AppXSVC) |
| Running | AudioEndpointBuilder | Windows-Audio-Endpunktsteuerung |
| Running | Audiosrv | Windows-Audio |
| Stopped | AxInstSV | ActiveX-Installer (AxInstSV) |

Demo: Gezielte Ausgabe mittels Format-Table



```
Administrator: Windows PowerShell
PS C:\Users\Administrator> Get-ADUser -Filter * |Format-Table Surname,GivenName,SamAccountName -Auto
Surname GivenName SamAccountName
----- -----------
Administrator Gast
Gast krbtgt
krbtgt

Administrator: Windows PowerShell
PS C:\Users\Administrator> Get-ADUser -Filter *

DistinguishedName : CN=Administrator,CN=Users,DC=kurs,DC=intern
Enabled           : True
GivenName         :
Name               :
ObjectClass       : user
ObjectGUID        : dfd51090-9ef6-4758-a6a1-7d8cd8f6726d
SamAccountName   : Administrator
SID               : S-1-5-21-1530042005-1223566000-148687676-500
Surname           :
UserPrincipalName :

DistinguishedName : CN=Gast,CN=Users,DC=kurs,DC=intern
Enabled           : False
GivenName         : Gast
Name               :
ObjectClass       : user
ObjectGUID        : 53c92b89-8250-4492-9466-4bf884d419e0
SamAccountName   : Gast
SID               : S-1-5-21-1530042005-1223566000-148687676-501
Surname           :
UserPrincipalName :

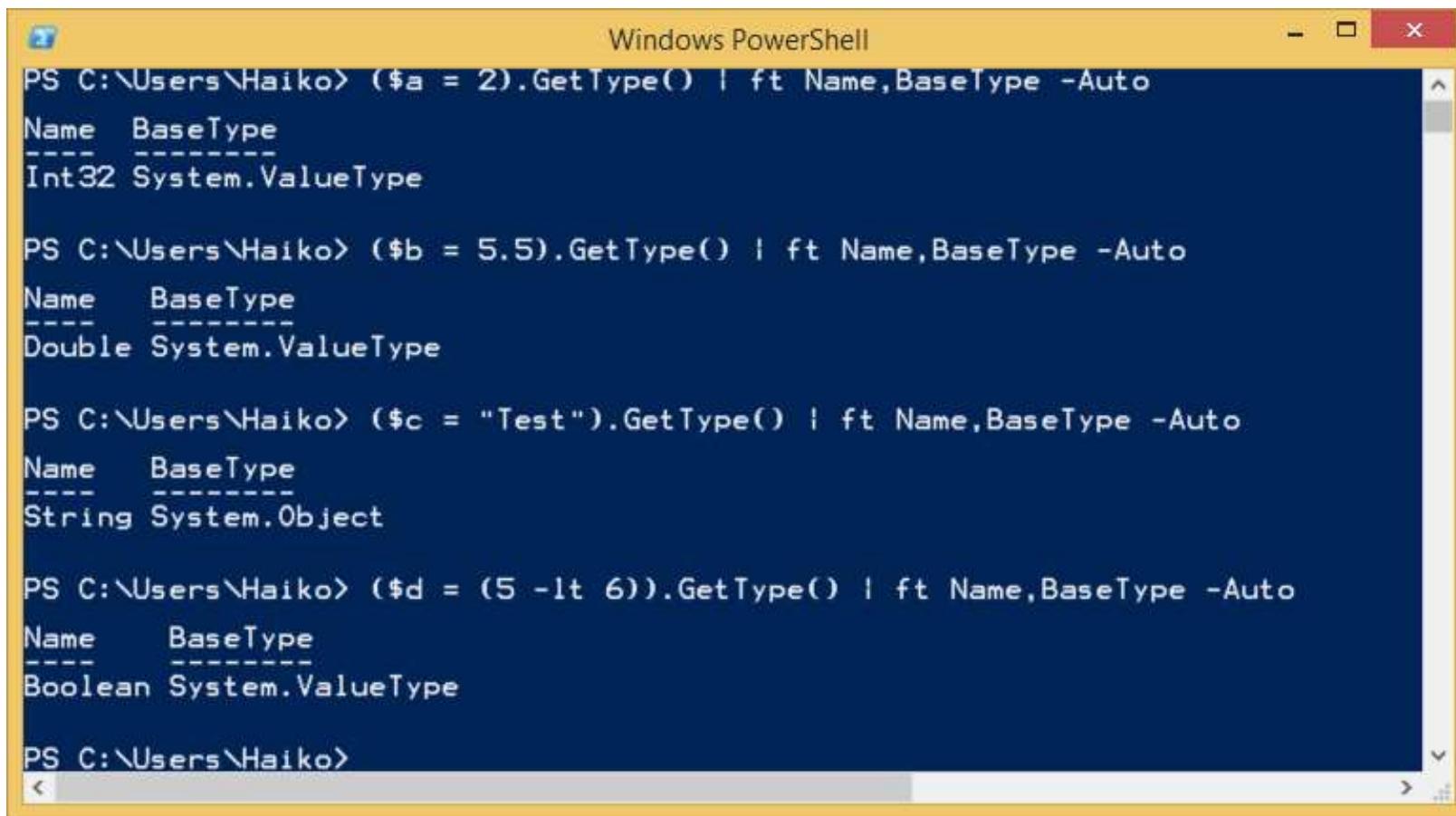
DistinguishedName : CN=krbtgt,CN=Users,DC=kurs,DC=intern
Enabled           : False
GivenName         :
Name               :
ObjectClass       : user
ObjectGUID        : 02a92887-b302-4326-a757-e9168e80deb7
SamAccountName   : krbtgt
SID               : S-1-5-21-1530042005-1223566000-148687676-502
```

Variablen I

Variablen

- Variablen speichern Werte, einzelne Objekte oder auch Listen von Werten oder Objekten
- Variablen beginnen i.d.R. mit \$
- Es gibt:
 - Benutzerdefinierte Variablen
 - Preference Variables und CommonParameter
 - Automatic Variables
- Typen müssen i.d.R. nicht definiert werden...

Variablen



A screenshot of a Windows PowerShell window titled "Windows PowerShell". The window shows five lines of PowerShell code demonstrating variable types. Each line uses the command `($variable).GetType() | ft Name, BaseType -Auto` to format the output. The results show:

- For variable `$a = 2`, the output is:

| Name | BaseType |
|-------|------------------|
| Int32 | System.ValueType |
- For variable `$b = 5.5`, the output is:

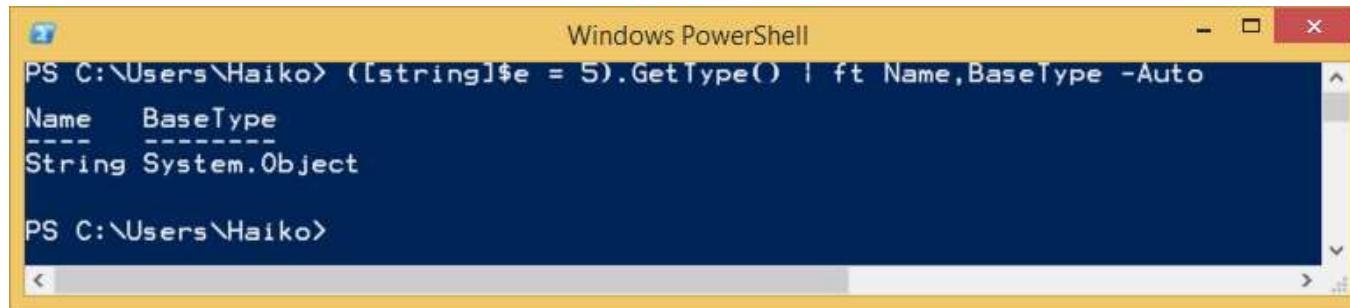
| Name | BaseType |
|--------|------------------|
| Double | System.ValueType |
- For variable `$c = "Test"`, the output is:

| Name | BaseType |
|--------|---------------|
| String | System.Object |
- For variable `$d = (5 -lt 6)`, the output is:

| Name | BaseType |
|---------|------------------|
| Boolean | System.ValueType |
- The final prompt shows the user's directory: `PS C:\Users\Haiko>`

Variablen

- ... können aber:

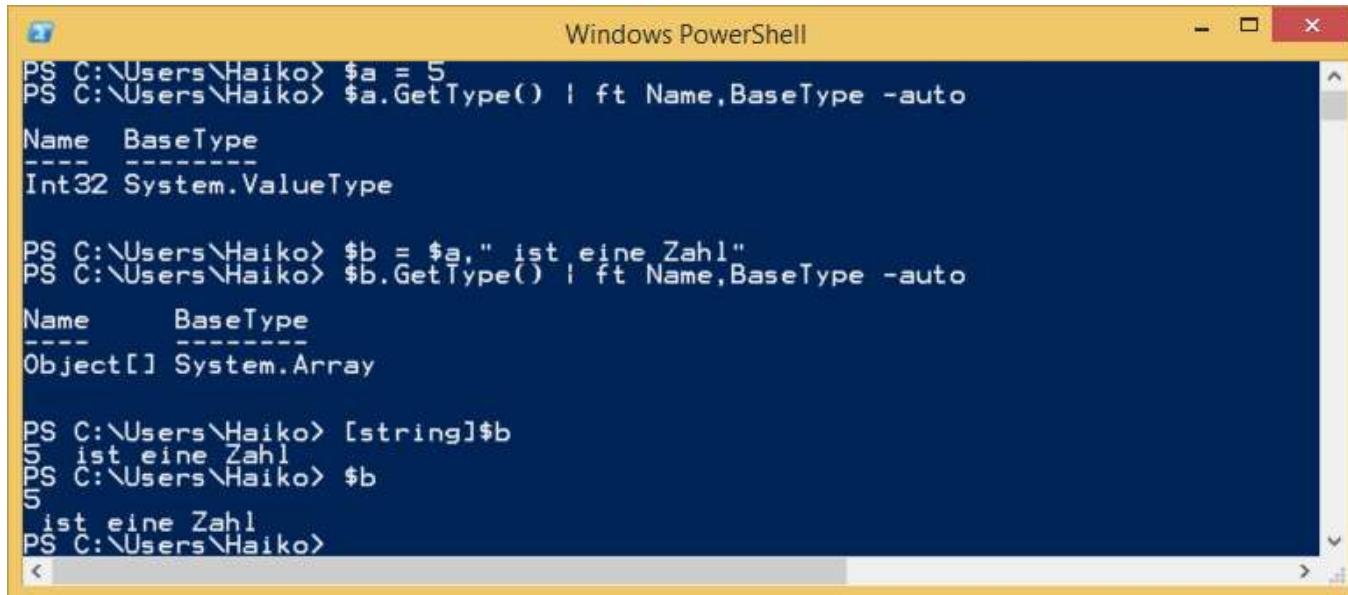


A screenshot of a Windows PowerShell window titled "Windows PowerShell". The command entered is `([string]$e = 5).GetType() | ft Name, BaseType -Auto`. The output shows a table with two columns: "Name" and "BaseType". There is one row with the value "String" under "Name" and "System.Object" under "BaseType". The PowerShell window has a yellow title bar and a dark blue background.

| Name | BaseType |
|--------|---------------|
| String | System.Object |

Variablen

- Falls nötig: Automatisches Typecasting



The screenshot shows a Windows PowerShell window with the following session transcript:

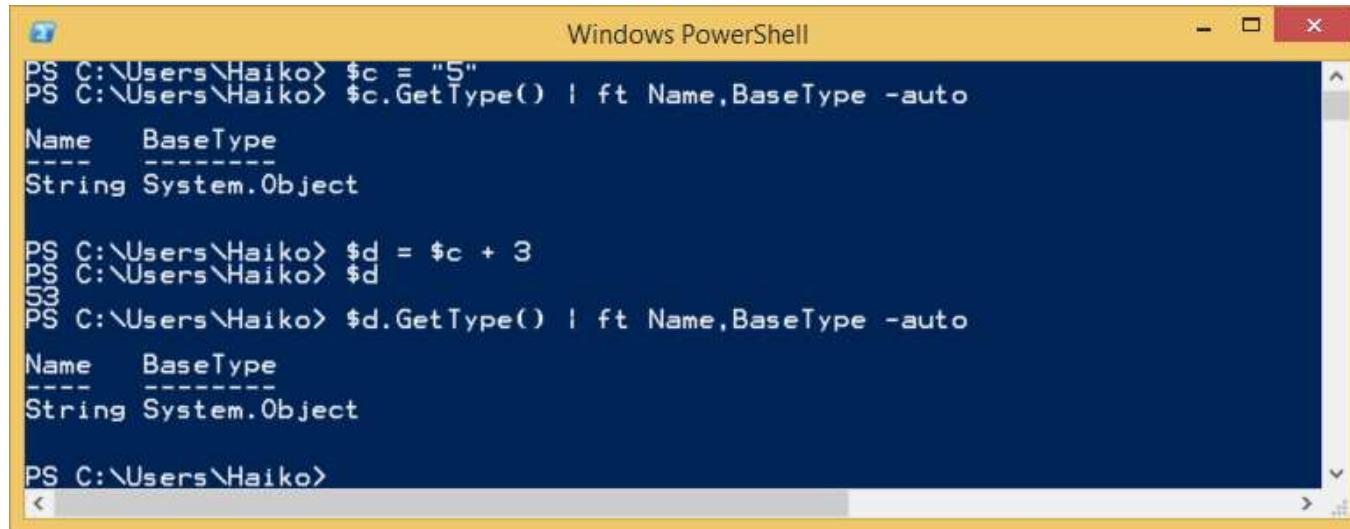
```
Windows PowerShell
PS C:\Users\Haiko> $a = 5
PS C:\Users\Haiko> $a.GetType() | ft Name,BaseType -auto
Name      BaseType
----      -----
Int32    System.ValueType

PS C:\Users\Haiko> $b = $a," ist eine Zahl"
PS C:\Users\Haiko> $b.GetType() | ft Name,BaseType -auto
Name      BaseType
----      -----
Object[]  System.Array

PS C:\Users\Haiko> [string]$b
5 ist eine Zahl
PS C:\Users\Haiko> $b
5
ist eine Zahl
PS C:\Users\Haiko>
```

Variablen

- Evtl. nicht immer wie erwartet:



A screenshot of a Windows PowerShell window titled "Windows PowerShell". The window shows the following command and its output:

```
PS C:\Users\Haiko> $c = "5"
PS C:\Users\Haiko> $c.GetType() | ft Name,BaseType -auto
Name   BaseType
----   -----
String System.Object

PS C:\Users\Haiko> $d = $c + 3
PS C:\Users\Haiko> $d
53
PS C:\Users\Haiko> $d.GetType() | ft Name,BaseType -auto
Name   BaseType
----   -----
String System.Object

PS C:\Users\Haiko>
```

Variablen

- Preference Variables:

| Variable | Standardwert |
|------------------------------|---|
| \$ConfirmPreference | High |
| \$DebugPreference | SilentlyContinue |
| \$ErrorActionPreference | Continue |
| \$ErrorView | NormalView |
| \$FormatEnumerationLimit | 4 |
| \$LogCommandHealthEvent | False (nicht protokolliert) |
| \$LogCommandLifecycleEvent | False (nicht protokolliert) |
| \$LogEngineHealthEvent | True (protokolliert) |
| \$LogEngineLifecycleEvent | True (protokolliert) |
| \$LogProviderLifecycleEvent | True (protokolliert) |
| \$LogProviderHealthEvent | True (protokolliert) |
| \$MaximumAliasCount | 4096 |
| \$MaximumDriveCount | 4096 |
| \$MaximumErrorCount | 256 |
| \$MaximumFunctionCount | 4096 |
| \$MaximumHistoryCount | 64 |
| \$MaximumVariableCount | 4096 |
| \$OFS | (Leerzeichen (" ")) |
| \$OutputEncoding | ASCIIEncoding-Objekt |
| \$ProgressPreference | Continue |
| \$PSEmailServer | (None) |
| \$PSSessionApplicationName | WSMAN |
| \$PSSessionConfigurationName | http://schemas.microsoft.com/powershell/microsoft.powershell |
| \$PSSessionOption | (Siehe unten) |
| \$VerbosePreference | SilentlyContinue |
| \$WarningPreference | Continue |
| \$WhatIfPreference | 0 |

Variablen

- Automatic Variables:

```
$$
Enthält das letzte Token in der letzten Zeile, die von der
Sitzung empfangen wurde.

$?
Enthält den Ausführungsstatus des letzten Vorgangs. Enthält
TRUE, wenn der letzte Vorgang erfolgreich war; FALSE, wenn
beim letzten Vorgang ein Fehler aufgetreten ist.

$^
Enthält das erste Token in der letzten Zeile, die von der
Sitzung empfangen wurde.

$_
Enthält das aktuelle Objekt im Pipelineobjekt. Diese Variable
kann in Befehlen verwendet werden, mit denen eine Aktion für
alle Objekte oder für ausgewählte Objekte in einer Pipeline
ausgeführt wird.

$args
Enthält ein Array der nicht deklarierten Parameter und/oder
der Parameterwerte, die an eine Funktion, ein Skript oder
einen Skriptblock übergeben werden.
Beim Erstellen einer Funktion können Sie die Parameter mit dem
Schlüsselwort "param" deklarieren, oder indem Sie eine durch
Trennzeichen getrennte Liste von Parametern in Klammern nach
dem Funktionsnamen hinzufügen.
```

\$ConsoleFileName

Enthält den Pfad der Konsolendatei (.psc1), die in der Sitzung zuletzt verwendet wurde. Diese Variable wird aufgefüllt, wenn Windows PowerShell mit dem PSConsoleFile-Parameter gestartet wird oder wenn mit dem Cmdlet "Export-Console" Snap-In-Namen in eine Konsolendatei exportiert werden.

Wenn Sie das Cmdlet "Export-Console" ohne Parameter verwenden, aktualisiert das Cmdlet automatisch die in der Sitzung zuletzt verwendete Konsolendatei. Mithilfe dieser automatischen Variablen können Sie bestimmen, welche Datei aktualisiert wird.

\$Error

Enthält ein Array von Fehlerobjekten, die die zuletzt aufgetretenen Fehler darstellen. Der letzte aufgetretene Fehler stellt das erste Fehlerobjekt im Array ("\$Error[0]") dar.

\$Event

Enthält ein PSEEventArgs-Objekt, das das zu verarbeitende Ereignis darstellt. Diese Variable wird nur im Action-Block eines Ereignisregistrierungsbefehls, z. B. "Register-ObjectEvent", aufgefüllt. Der Wert der Variablen ist mit dem Objekt identisch, das vom Cmdlet "Get-Event" zurückgegeben wird. Daher können Sie die Eigenschaften der Variablen "\$Event", z. B. \$Event.TimeGenerated, in einem Action-Scriptblock verwenden.

\$EventSubscriber

Enthält ein PSEventSubscriber-Objekt, das den Ereignisabonnenten des zu verarbeitenden Ereignisses darstellt. Diese Variable wird nur im Action-Block eines Ereignisregistrierungsbefehls aufgefüllt. Der Wert der Variablen ist mit dem Objekt identisch, das vom Cmdlet "Get-EventSubscriber" zurückgegeben wird.

Weitere: <http://technet.microsoft.com/de-de/library/dd347675.aspx>

Übung: Variablen

- Speichern Sie das heutige Datum in einer Variable „Datum“
- Ermitteln Sie den Typ von „Datum“
- Welches Datum haben wir in 100 Tagen?

-Hint: \$Variable.Methode(Argument)



Mögliche Lösungen

- \$Datum = Get-Date
- \$Datum.GetType()
 - Hint: "\$Datum | Get-Member"
- \$Datum.AddDays(100)
 - Alternativ auch: (Get-Date).AddDays(100)

5
★

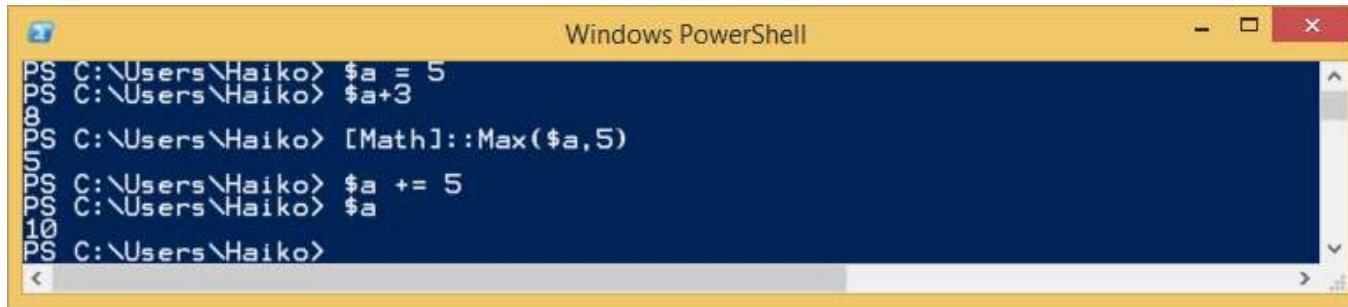
TRAINING



Operationen

Operationen

- Mit Zahlen und Variablen, die Zahlen enthalten:



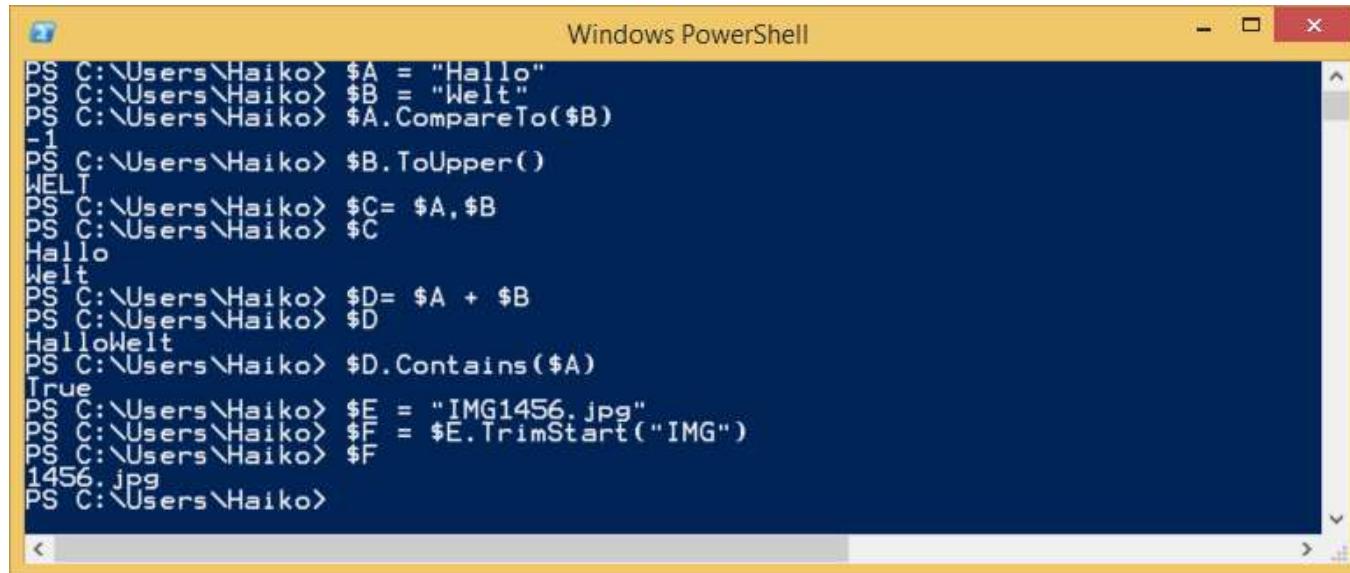
A screenshot of a Windows PowerShell window titled "Windows PowerShell". The window shows the following command history:

```
PS C:\Users\Haiko> $a = 5
PS C:\Users\Haiko> $a+3
8
PS C:\Users\Haiko> [Math]::Max($a,5)
5
PS C:\Users\Haiko> $a += 5
PS C:\Users\Haiko> $a
10
PS C:\Users\Haiko>
```

<http://technet.microsoft.com/en-us/library/hh848303.aspx>

Operationen

- Mit Strings und Variablen, die Strings enthalten:

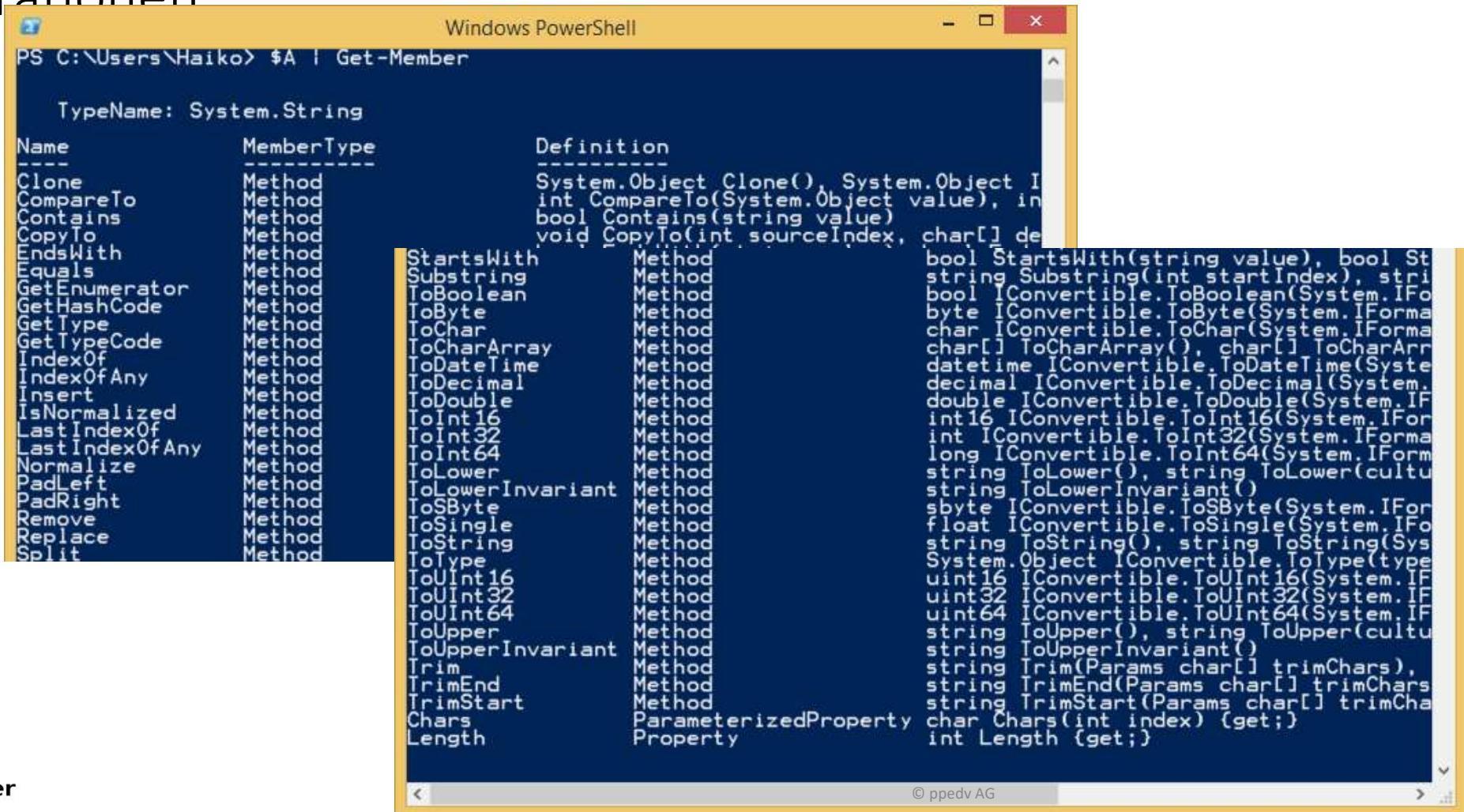


The screenshot shows a Windows PowerShell window titled "Windows PowerShell". It displays the following command-line session:

```
PS C:\Users\Haiko> $A = "Hallo"
PS C:\Users\Haiko> $B = "Welt"
PS C:\Users\Haiko> $A.CompareTo($B)
-1
PS C:\Users\Haiko> $B.ToUpper()
WELT
PS C:\Users\Haiko> $C= $A,$B
PS C:\Users\Haiko> $C
Hallo
Welt
PS C:\Users\Haiko> $D= $A + $B
PS C:\Users\Haiko> $D
HalloWelt
PS C:\Users\Haiko> $D.Contains($A)
True
PS C:\Users\Haiko> $E = "IMG1456.jpg"
PS C:\Users\Haiko> $F = $E.TrimStart("IMG")
PS C:\Users\Haiko> $F
1456.jpg
PS C:\Users\Haiko>
```

<http://technet.microsoft.com/en-us/library/ee692804.aspx>

Operationen



Windows PowerShell

```
PS C:\Users\Haiko> $A | Get-Member
```

| Name | MemberType | Definition |
|----------------|------------|--|
| Clone | Method | System.Object Clone(), System.Object I |
| CompareTo | Method | int CompareTo(System.Object value), in |
| Contains | Method | bool Contains(string value) |
| CopyTo | Method | void CopyTo(int sourceIndex, char[] de |
| EndsWith | Method | bool EndsWith(string value), bool St |
| Equals | Method | string Substring(int startIndex), stri |
| GetEnumerator | Method | bool IConvertible.ToBoolean(System.IFo |
| GetHashCode | Method | byte IConvertible.ToByte(System.IFor |
| GetType | Method | char IConvertible.ToChar(System.IFor |
| GetTypeCode | Method | char[] ToCharArray(), char[] ToCharArr |
| IndexOf | Method | datetime IConvertible.ToDateTime(Syste |
| IndexOfAny | Method | decimal IConvertible.ToDecimal(System. |
| Insert | Method | double IConvertible.ToDouble(System.IF |
| IsNormalized | Method | int16 IConvertible.ToInt16(System.IFor |
| LastIndexOf | Method | int IConvertible.ToInt32(System.IForm |
| LastIndexOfAny | Method | long IConvertible.ToInt64(System.IForm |
| Normalize | Method | string ToLower(), string ToLower(cultu |
| PadLeft | Method | string ToLowerInvariant() |
| PadRight | Method | sbyte IConvertible.ToSByte(System.IFor |
| Remove | Method | float IConvertible.ToSingle(System.IFo |
| Replace | Method | string ToString(), string ToString(Sys |
| Split | Method | System.Object IConvertible.ToType(type |
| | | uint16 IConvertible.ToInt16(System.IF |
| | | uint32 IConvertible.ToInt32(System.IF |
| | | uint64 IConvertible.ToInt64(System.IF |
| | | string ToUpper(), string ToUpper(cultu |
| | | string ToUpperInvariant() |
| | | string Trim(Params char[] trimChars), |
| | | string TrimEnd(Params char[] trimChars |
| | | string TrimStart(Params char[] trimCha |
| | | char Chars(int index) {get;} |
| | | int Length {get;} |

Übung: Operatoren

- Erzeugen Sie zwei Variablen a und b mit dem Inhalt „Hallo“ und „Welt“ und lassen Sie sich damit „Hallo Welt“ ausgeben
- Lassen Sie sich die Länge von a ausgeben
- Ersetzen Sie das „l“ in b durch ein „r“

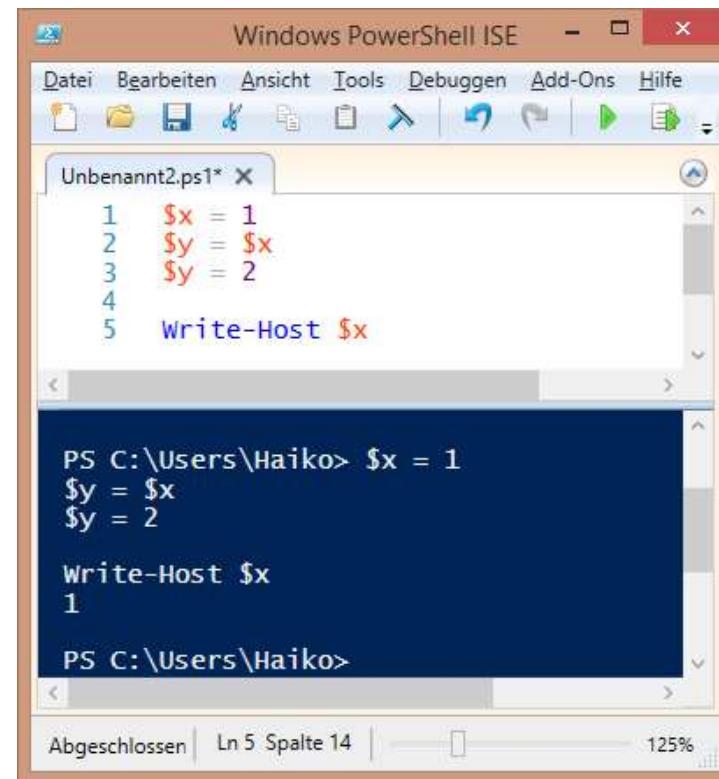


Mögliche Lösungen

- \$a = „Hallo“; \$b = „Welt“; \$a + „ “ + \$b
- \$a.Length
- \$b.Replace(„l“, „r“)

Operatoren

- \$x = 1
- \$y = \$x
- \$y = 2
- Welchen Wert hat x jetzt?



The screenshot shows the Windows PowerShell ISE interface. The script pane contains the following code:

```
1 $x = 1
2 $y = $x
3 $y = 2
4
5 Write-Host $x
```

The output pane shows the execution of the script:

```
PS C:\Users\Haiko> $x = 1
$y = $x
$y = 2

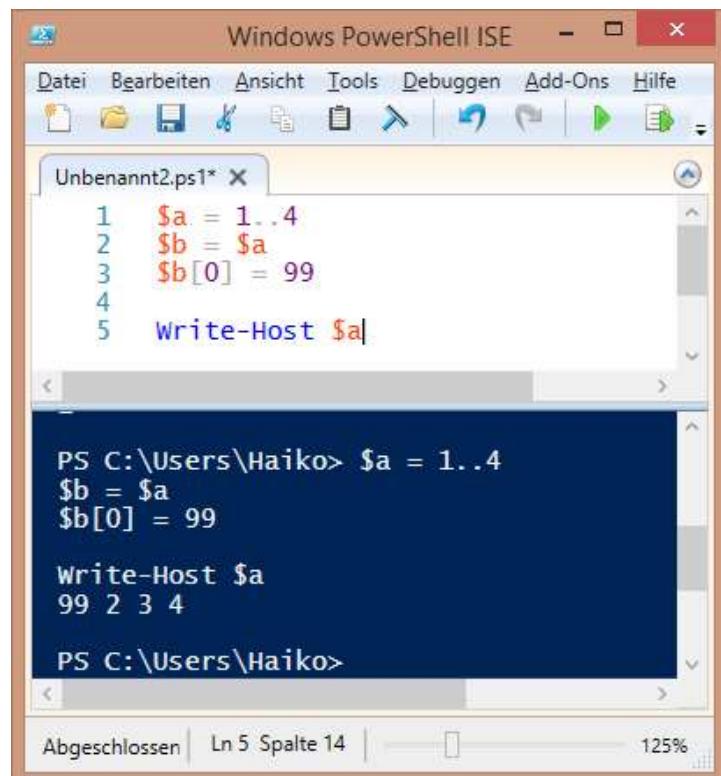
Write-Host $x
1

PS C:\Users\Haiko>
```

The status bar at the bottom indicates "Abgeschlossen" (Completed), "Ln 5 Spalte 14", and a zoom level of "125%".

Operatoren

- \$a = 1..4
- \$b = \$a
- \$b[0] = 99
- Welchen Wert hat a jetzt?



The screenshot shows a Windows PowerShell ISE window with a script file named "Unbenannt2.ps1". The script contains the following code:

```
1 $a = 1..4
2 $b = $a
3 $b[0] = 99
4
5 Write-Host $a
```

In the bottom pane, the PowerShell command PS C:\Users\Haiko> is followed by the execution of the script. The output shows the assignment of \$a to 1..4, the assignment of \$b to \$a, the assignment of \$b[0] to 99, and finally the output of the Write-Host command, which prints the value 99 followed by the sequence 2 3 4.

Formatierung der Ausgabe

Formatierung

- Einfache Varianten:

- Get-Service | **Format-Wide**
- Get-Process | Format-Wide -Property ID
- Get-Process | Format-Wide -Col 5
- Get-Process | FW -AutoSize

Formatierung

- Einfache Varianten:

- Get-Service | **Format-List -Property ***
- Get-Process | FL -Prop Name, ID
- Get-Service | FL Name, Status, DisplayName

Formatierung

- Einfache Varianten:

- Get-Process | **Format-Table** -Property ID,Name
- Get-Process | FT *
- Get-Service | FT Name,Status -AutoSize

Formatierung

- Angepasste Tabellen-Ausgabe:

```
1 Get-Process |  
2 Format-Table -Property Name,Id,@{n='VM(MB)';  
3 e={$PSItem.VM / 1MB};  
4 formatString='N2';  
5 align='right'} -AutoSize  
6
```

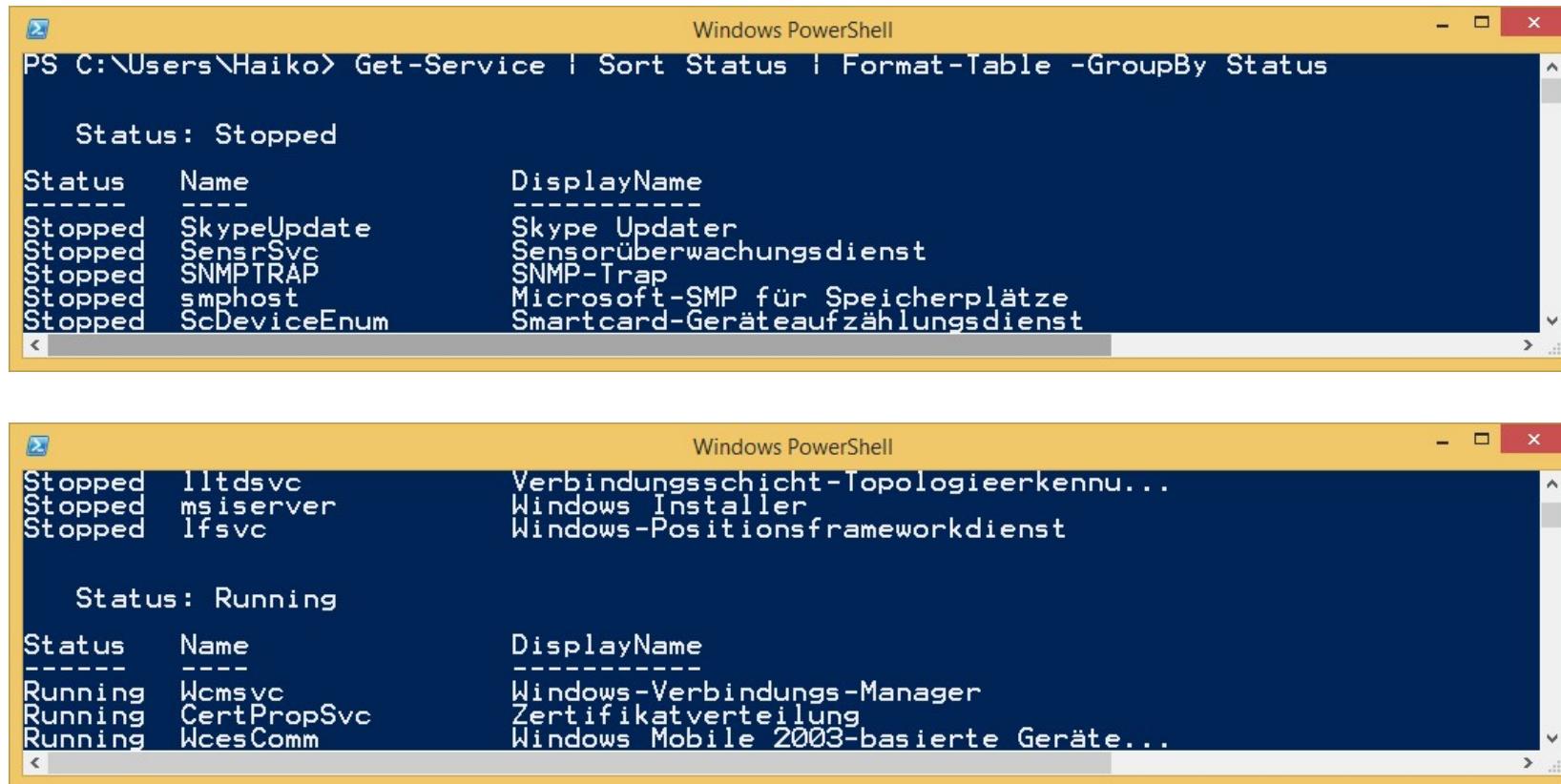
| Name | Id | VM(MB) |
|--------------------------|------|--------|
| AdobeARM | 7808 | 104,40 |
| alg | 2956 | 26,71 |
| AppleMobileDeviceService | 2060 | 93,04 |
| armsvc | 1572 | 44,36 |
| chrome | 564 | 237,84 |
| chrome | 1296 | 273,19 |
| chrome | 3764 | 282,43 |
| chrome | 4104 | 391,14 |
| chrome | 4548 | 221,05 |
| chrome | 5600 | 265,36 |
| chrome | 6340 | 512,67 |

-Autosize sorgt dafür, dass Spalten auf optimaler Breite dargestellt werden

Formatierung

- **-GroupBy** gruppiert eine Liste
- Dazu ist ein Attribut anzugeben, nach dem gruppiert werden soll
- Immer, wenn sich der Wert des Attributes ändert -> neuer Tabellenkopf
- Sinnvollerweise sollte vorher sortiert werden (nach dem selben Attribut)

Formatierung



The image shows two side-by-side Windows PowerShell windows demonstrating the use of the `Format-Table` cmdlet for displaying service status.

Top Window (Without Format-Table):

```
PS C:\Users\Haiko> Get-Service | Sort Status | Format-Table -GroupBy Status
```

| Status | Name | DisplayName |
|---------|--------------|-----------------------------------|
| Stopped | SkypeUpdate | Skype Updater |
| Stopped | SensrSvc | Sensorüberwachungsdienst |
| Stopped | SNMPTRAP | SNMP-Trap |
| Stopped | smphost | Microsoft-SMP für Speicherplätze |
| Stopped | ScDeviceEnum | Smartcard-Geräteaufzählungsdienst |

Bottom Window (With Format-Table):

```
PS C:\Users\Haiko> Get-Service | Sort Status | Format-Table -GroupBy Status
```

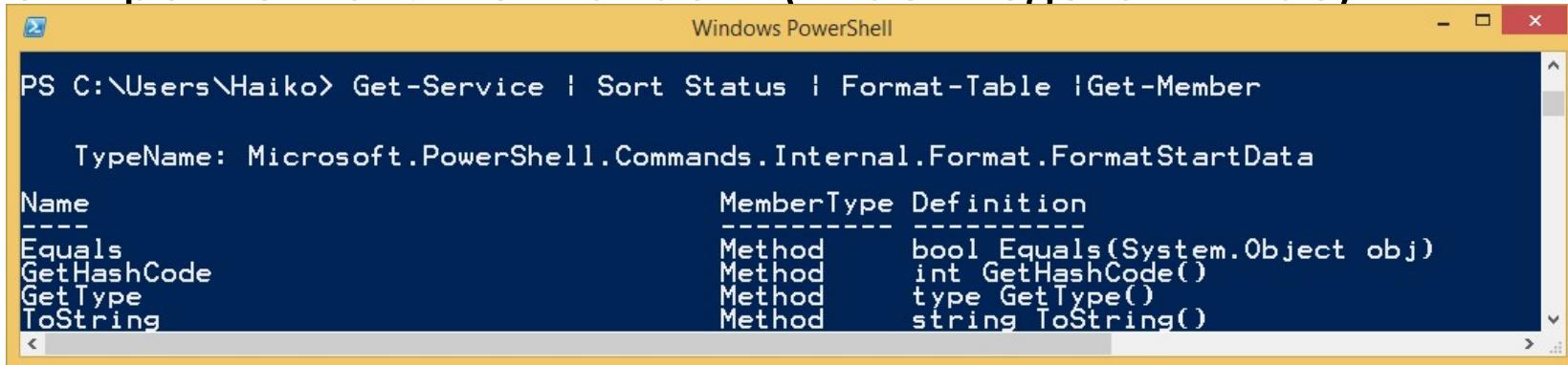
| Status | Name | DisplayName |
|---------|-----------|--|
| Stopped | l1tdsvc | Verbindungsschicht-Topologieerkennu... |
| Stopped | msiserver | Windows Installer |
| Stopped | lfsvc | Windows-Positionsframeworkdienst |

Status: Running

| Status | Name | DisplayName |
|---------|-------------|--|
| Running | Wcmsvc | Windows-Verbindungs-Manager |
| Running | CertPropSvc | Zertifikatverteilung |
| Running | WcesComm | Windows Mobile 2003-basierte Geräte... |

Formatierung

- Formatierung sollte erst angewendet werden, wenn sich die Objekte in der Pipeline nicht mehr ändern (in der Regel am Ende)!



The screenshot shows a Windows PowerShell window titled "Windows PowerShell". The command entered is:

```
PS C:\Users\Haiko> Get-Service | Sort Status | Format-Table |Get-Member
```

The output displays the members of the `FormatStartData` type, which is a subtype of `Microsoft.PowerShell.Commands.Internal.Format.FormatStartData`. The table shows the following members:

| Name | MemberType | Definition |
|-------------|------------|--------------------------------|
| Equals | Method | bool Equals(System.Object obj) |
| GetHashCode | Method | int GetHashCode() |
| GetType | Method | type GetType() |
| ToString | Method | string ToString() |

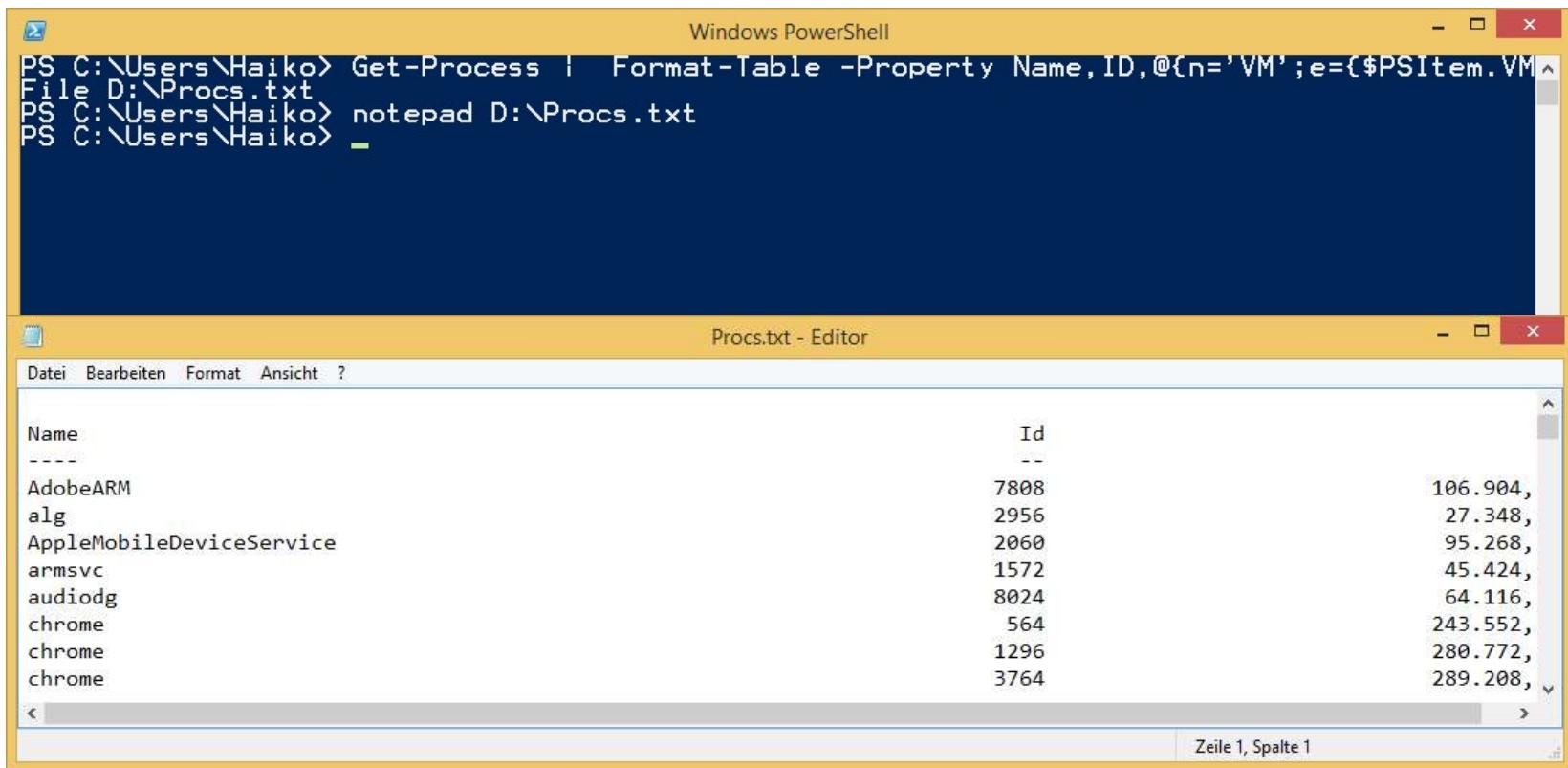
- Nur eine geringe Anzahl von Commandlets akzeptiert die sehr speziellen Objekte, die von Format-Kommandos erzeugt werden

Formatierung

- Folgende Commandlets können nach der Formatierung folgen:
 - **Out-Host** – Ausgabe auf den Bildschirm
 - **Out-File** – Ausgabe in eine Text-Datei
 - **Out-Printer** – Ausgabe auf dem Drucker
- Der umgeleitete Inhalt wird genau so aussehen, wie er auf dem Monitor aussehen würde

Formatierung

- Bsp.:

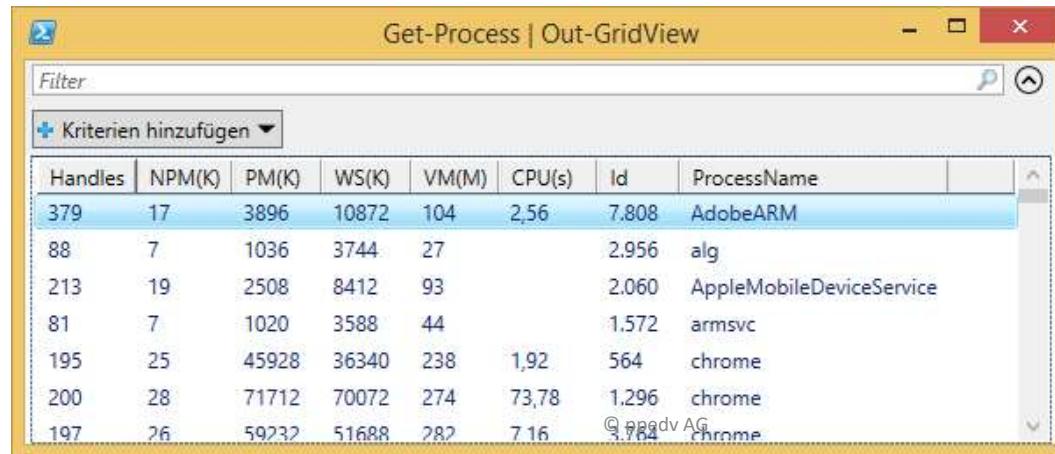


The screenshot shows a Windows PowerShell window and a Notepad window. The PowerShell window displays the command: PS C:\Users\Haiko> Get-Process | Format-Table -Property Name, ID, @{n='VM';e={\$PSItem.VM}} | File D:\Procs.txt. The Notepad window shows the contents of the file 'Procs.txt' which is a table with columns 'Name' and 'Id'. The table data is as follows:

| Name | Id |
|--------------------------|------|
| AdobeARM | 7808 |
| alg | 2956 |
| AppleMobileDeviceService | 2060 |
| armsvc | 1572 |
| audiodg | 8024 |
| chrome | 564 |
| chrome | 1296 |
| chrome | 3764 |

Formatierung

- Out-GridView gibt Objekte in einer sortierbaren, filterbaren Tabellenansicht aus
- Akzeptiert keine formatierte Ausgabe
- Nur verfügbar, wenn die PowerShell ISE installiert ist



The screenshot shows a PowerShell ISE window titled "Get-Process | Out-GridView". The window displays a grid of process information. The columns are labeled: Handles, NPM(K), PM(K), WS(K), VM(M), CPU(s), Id, and ProcessName. The data rows are:

| Handles | NPM(K) | PM(K) | WS(K) | VM(M) | CPU(s) | Id | ProcessName |
|---------|--------|-------|-------|-------|--------|-------|--------------------------|
| 379 | 17 | 3896 | 10872 | 104 | 2,56 | 7.808 | AdobeARM |
| 88 | 7 | 1036 | 3744 | 27 | | 2.956 | alg |
| 213 | 19 | 2508 | 8412 | 93 | | 2.060 | AppleMobileDeviceService |
| 81 | 7 | 1020 | 3588 | 44 | | 1.572 | armsvc |
| 195 | 25 | 45928 | 36340 | 238 | 1,92 | 564 | chrome |
| 200 | 28 | 71712 | 70072 | 274 | 73,78 | 1.296 | chrome |
| 197 | 26 | 59232 | 51688 | 282 | 7,16 | 3.784 | chrome |

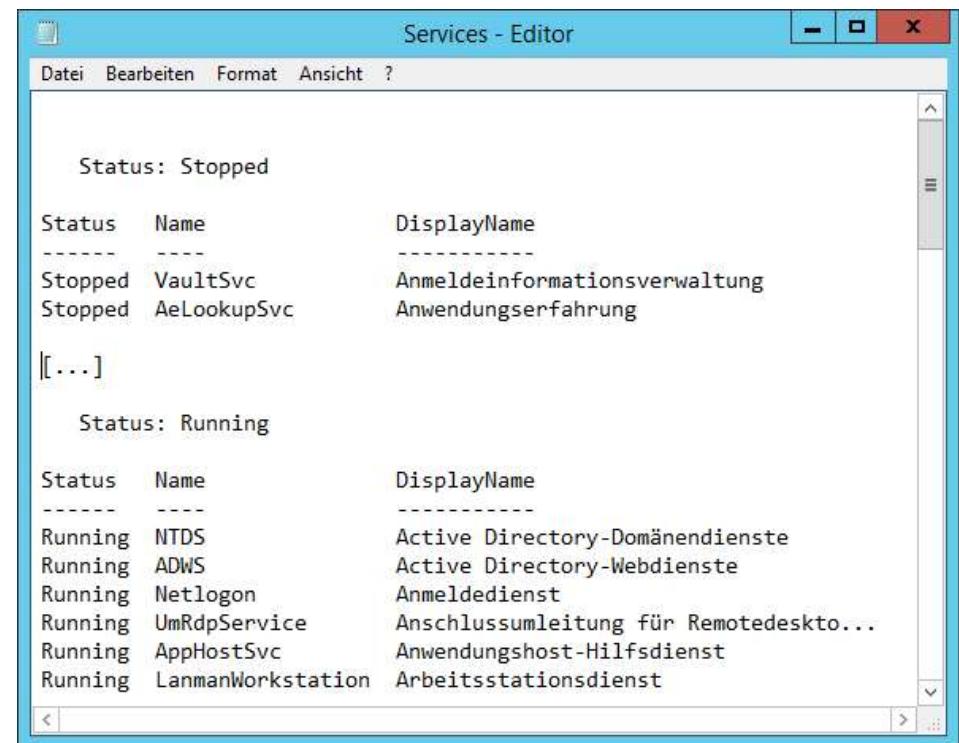
Übung: Formatierung

- Geben Sie eine Liste der Dienste
 - Sortiert nach dem Status und danach nach dem deutschen Namen
 - Gruppiert nach Status
 - In einer Datei auf Laufwerk C:\ aus



Mögliche Lösungen

- Get-Service | Sort-Object Status, DisplayName
| Format-Table -GroupBy
Status |
Out-File C:\Services.txt



The screenshot shows the Windows Services Editor window with two main sections: "Status: Stopped" and "Status: Running".

Status: Stopped

| Status | Name | DisplayName |
|---------|-------------|-------------------------------|
| Stopped | VaultSvc | Anmeldeinformationsverwaltung |
| Stopped | AeLookupSvc | Anwendungserfahrung |

[...]

Status: Running

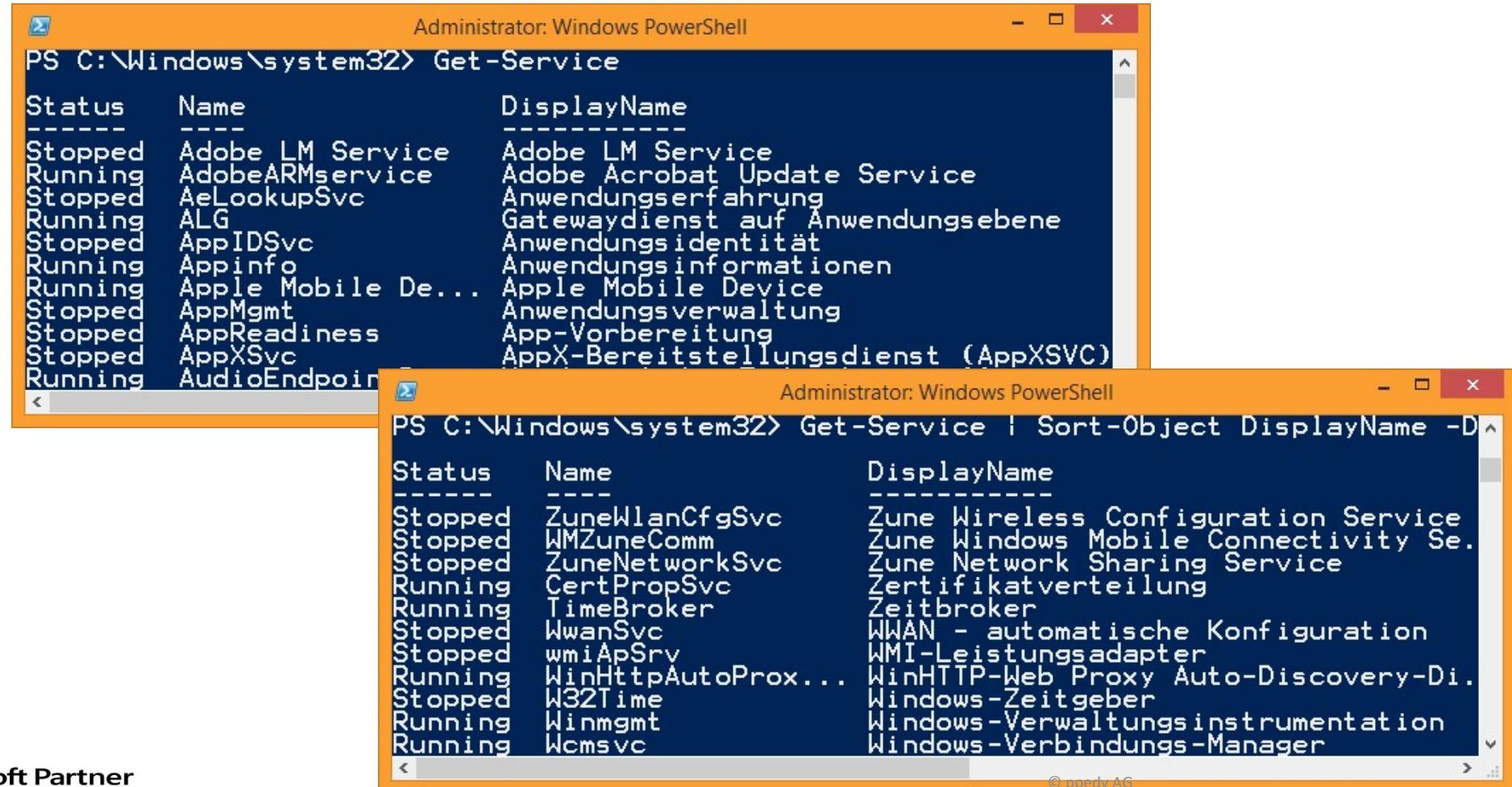
| Status | Name | DisplayName |
|---------|-------------------|--|
| Running | NTDS | Active Directory-Domänendienste |
| Running | ADWS | Active Directory-Webdienste |
| Running | Netlogon | Anmeldedienst |
| Running | UmRdpService | Anschlussumleitung für Remotedeskt... o |
| Running | AppHostSvc | Anwendungshost-Hilfsdienst |
| Running | LanmanWorkstation | Arbeitsstationsdienst |

Sortieren, Messen, Selektieren

Die Pipeline - Sortierung

- Commandlets besitzen meist eine Standard-Sortierung (oft nach Namen)
- **Sort-Object** kann Objekte in der Pipeline umsortieren
- Dabei kann sowohl das Attribut als auch die Richtung angegeben werden
- Kurzform: **Sort**
- Sortieren verändert die Ausgabe-Reihenfolge, aber die Objekte bleiben alle erhalten

Die Pipeline - Sortierung



The image shows two side-by-side Windows PowerShell windows. Both are titled "Administrator: Windows PowerShell".

The top window displays the command "Get-Service" and its output:

| Status | Name | DisplayName |
|---------|--------------------|-------------------------------------|
| Stopped | Adobe LM Service | Adobe LM Service |
| Running | AdobeARMservice | Adobe Acrobat Update Service |
| Stopped | AeLookupSvc | Anwendungserfahrung |
| Running | ALG | Gatewaydienst auf Anwendungsebene |
| Stopped | AppIDSvc | Anwendungsidentität |
| Running | Appinfo | Anwendungsinformationen |
| Running | Apple Mobile De... | Apple Mobile Device |
| Stopped | AppMgmt | Anwendungsverwaltung |
| Stopped | AppReadiness | App-Vorbereitung |
| Stopped | AppXSvc | AppX-Bereitstellungsdiens (AppXSVC) |
| Running | AudioEndpoint | |

The bottom window displays the command "Get-Service | Sort-Object DisplayName -D" and its output, which is identical to the top window but sorted by DisplayName in descending order:

| Status | Name | DisplayName |
|---------|--------------------|--------------------------------------|
| Stopped | ZuneWlanCfgSvc | Zune Wireless Configuration Service |
| Stopped | WMZuneComm | Zune Windows Mobile Connectivity Se. |
| Stopped | ZuneNetworkSvc | Zune Network Sharing Service |
| Running | CertPropSvc | Zertifikatverteilung |
| Running | TimeBroker | Zeitbroker |
| Stopped | WwanSvc | WWAN - automatische Konfiguration |
| Stopped | wmiApSrv | WMI-Leistungsadapter |
| Running | WinHttpAutoProx... | WinHTTP-Web Proxy Auto-Discovery-Di. |
| Stopped | W32Time | Windows-Zeitgeber |
| Running | Winmgmt | Windows-Verwaltungsinstrumentation |
| Running | WcmSvc | Windows-Verbindungs-Manager |

Die Pipeline - Sortierung

```
Get-Service | Sort-Object -Property Name
```

```
Get-Process | Sort Name, ID
```

```
Get-Process | Sort VM -Descending
```

Die Pipeline - Messen

- **Measure-Object** nimmt eine Sammlung von Objekten entgegen und zählt diese
- Fügen Sie **-Property** und den Namen eines Attributes sowie folgende Parameter hinzu, um mit
 - **-Average** den Durchschnitt zu berechnen
 - **-Minimum** den kleinsten Wert anzuzeigen
 - **-Maximum** den größten Wert anzuzeigen
 - **-Sum** die Summe auszurechnen
- *Ausgegeben wird ein Measurement-Objekt, nicht die Objekte, die hineingegeben wurden!*

Die Pipeline - Messen

Get-Service | Measure-Object

Get-Process |
Measure-Object -Prop PM -Sum -Average

Die Pipeline - Messen

```
Windows PowerShell
PS C:\Users\Haiko> Get-Process "chrome" | Measure-Object -Property PM -Sum -Average

Count      : 30
Average    : 81839718,4
Sum        : 2455191552
Maximum    :
Minimum    :
Property   : PM

PS C:\Users\Haiko> Get-Process
Handles  NPM(K)    PM(K)      WS(K)  VM(M) CPU(s)  Id  ProcessName
----  -----  -----  -----  -----  -----  --  -----
     88      7    1124      3684     27  3028  alg
    213     19    2708      8656     93  2096 AppleMobileDeviceService
     81      7    1028      3568     44  2056 arm svc
    231     11    9836     11400     70  8864 audi o dg
    117     28    7428     13836    108  7848 calc
    201     26    60292     50352    257  212 chrome
    199     27    58396     58720    271  564 chrome
  1999     162   187632    225104    497  1.605,39 1292 chrome
    197     26    52132     51616    243  1.98  1668 chrome
    225     30    59968     63600    401  11,23  4632 chrome
    243     34   114912    121228    415  71,81  4796 chrome
    395     46   261748    264844    672  886,48  5152 chrome
    199     25    46168     44268    234  1,02  5224 chrome
    185     30   114000     94640    295  558,58  5280 chrome
    184     23   31888      21272    211  2,59  5304 chrome
    200     27   65800      61884    260  2,61  5672 chrome
    197     28   67412      62420    264  2,64  5732 chrome
    198     27   72284      61692    278  3,52  5752 chrome
```

Die Pipeline – Teile auswählen

- **Select-Object** kennt zwei Verwendungen
- Wählt einen Teil der übergebenen Objekte
- Mit diversen Parametern kann angegeben werden, welcher Teil zurückgegeben werden soll
 - **-First x** – Die ersten x Elemente werden übergeben
 - **-Last x** – Die letzten x Elemente werden übergeben
 - **-Skip x** – Die ersten x Elemente werden übersprungen, Rest übergeben
- *Keine weitere Filterung oder Kriterien möglich*

Die Pipeline – Teile auswählen

```
Get-Service |  
Sort-Object -Property Status |  
Select-Object -First 10
```

```
Get-Process |  
Sort VM -Descending |  
Select -First 10
```

Die Pipeline – Teile auswählen

- Zweite Verwendung: Liefert gewünschte Attribute zurück
- Nach **-Property** können gewünschte Attribute aufgelistet werden
- Kann mit **-First**, **-Last** und **-Skip** kombiniert werden

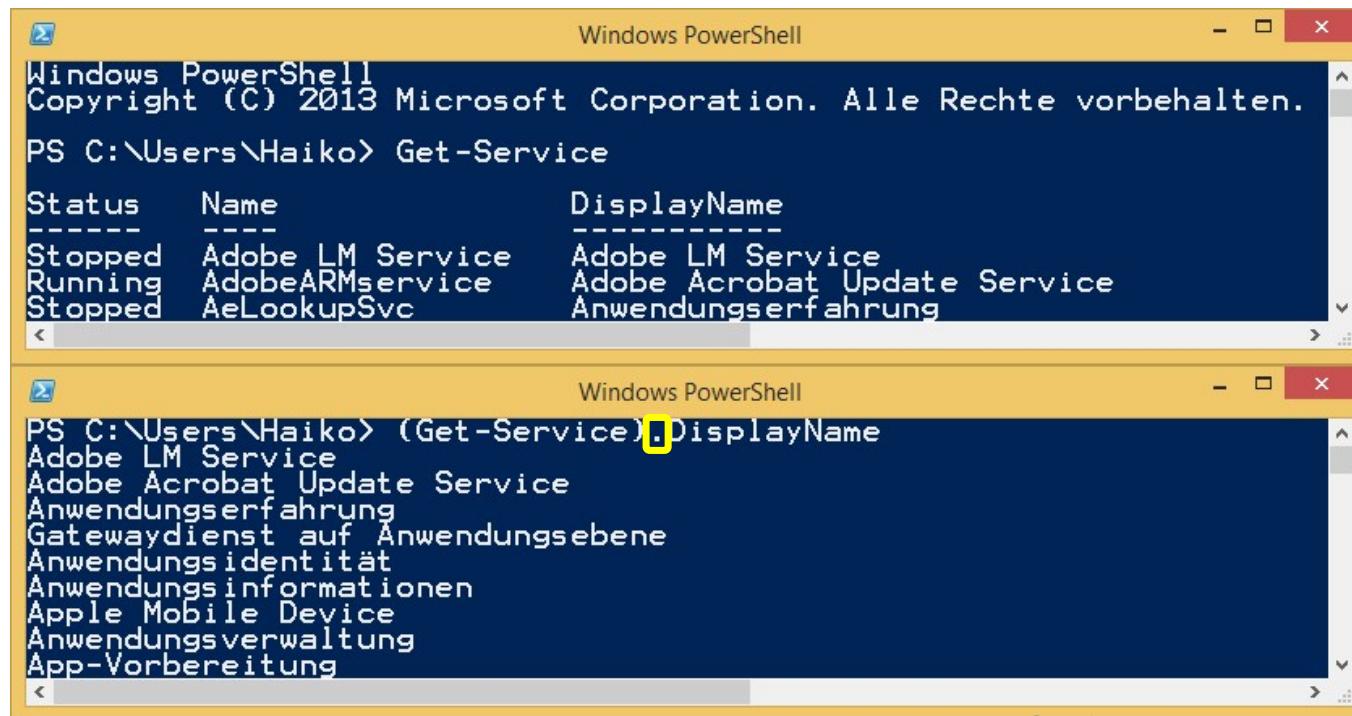
Die Pipeline – Teile auswählen

```
Get-Service |  
Select-Object -Property Name,Status
```

```
Get-Process |  
Sort PM -Descending |  
Select -Property Name, ID, PM, VM -First 10
```

Die Pipeline – Teile auswählen

- Wenn Rückgabe ein oder mehrere ganze Objekte: Auswahl einzelner Attribute möglich



The screenshot shows two separate Windows PowerShell windows. The top window displays the output of the command `Get-Service`, which lists several system services with their status, names, and display names. The bottom window shows the result of selecting the `DisplayName` attribute from the pipeline, resulting in a list of service names.

```
Windows PowerShell
Copyright (C) 2013 Microsoft Corporation. Alle Rechte vorbehalten.

PS C:\Users\Haiko> Get-Service

Status     Name             DisplayName
-----     --   -----
Stopped    Adobe_LM_Service  Adobe_LM_Service
Running    AdobeARMservice   Adobe Acrobat Update Service
Stopped    AeLookupSvc       Anwendungserfahrung

Windows PowerShell
PS C:\Users\Haiko> (Get-Service).DisplayName
Adobe_LM_Service
Adobe Acrobat Update Service
Anwendungserfahrung
Gatewaydienst auf Anwendungsebene
Anwendungsidentität
Anwendungsinformationen
Apple Mobile Device
Anwendungsverwaltung
App-Vorbereitung
```

Übung: Pipeline I

1. Lassen Sie sich die Nachnamen aller Benutzer im AD nebeneinander ausgeben
2. Lassen Sie sich alle AD-Benutzer sortiert nach ihrem Vornamen in einer Tabelle ausgeben
3. Zählen Sie die Anzahl aller AD-Gruppen
4. Listen Sie die letzten 3 AD-Benutzer auf



Mögliche Lösungen

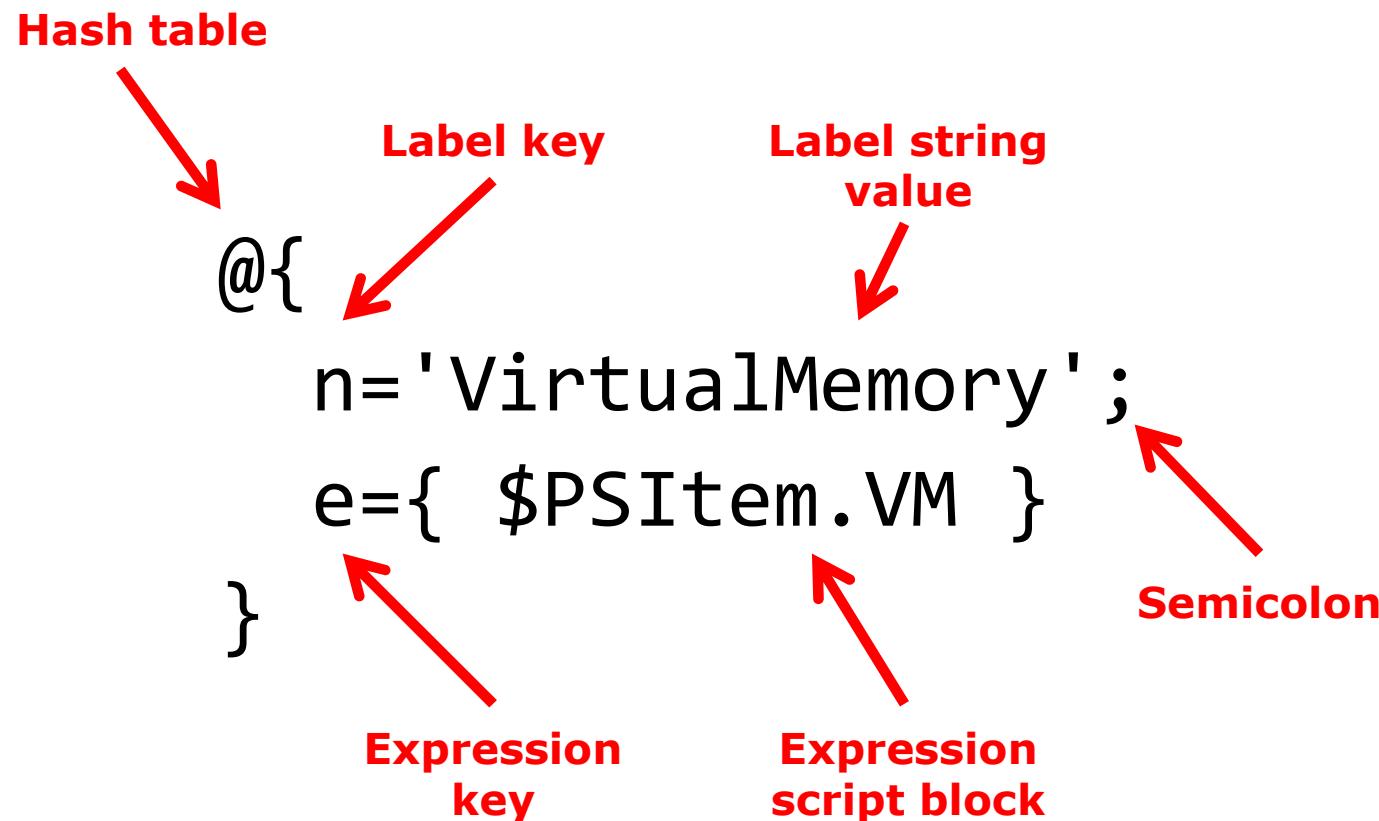
1. Get-ADUser -Filter * | Format-Wide Surname
2. Get-ADUser -Filter * | Sort-Object Givenname | Format-Table
3. (Get-ADGroup -Filter * | Measure-Object).Count
 1. Get-ADGroup -Filter * | Measure-Object
 2. (Get-ADGroup -Filter *).Count
4. Get-ADUser -Filter * | Select-Object -Last 3

Übung: Pipeline I

1. Lassen Sie sich die Prozessnamen untereinander ausgeben
2. Lassen Sie sich alle Prozesse sortiert nach ihrem Displaynamen in einer Tabelle ausgeben
3. Zählen Sie die Anzahl aller svchost Prozesse
4. Listen Sie die 3 Prozesse mit dem höchstem PM auf



Die Pipeline – Eigene Werte berechnen



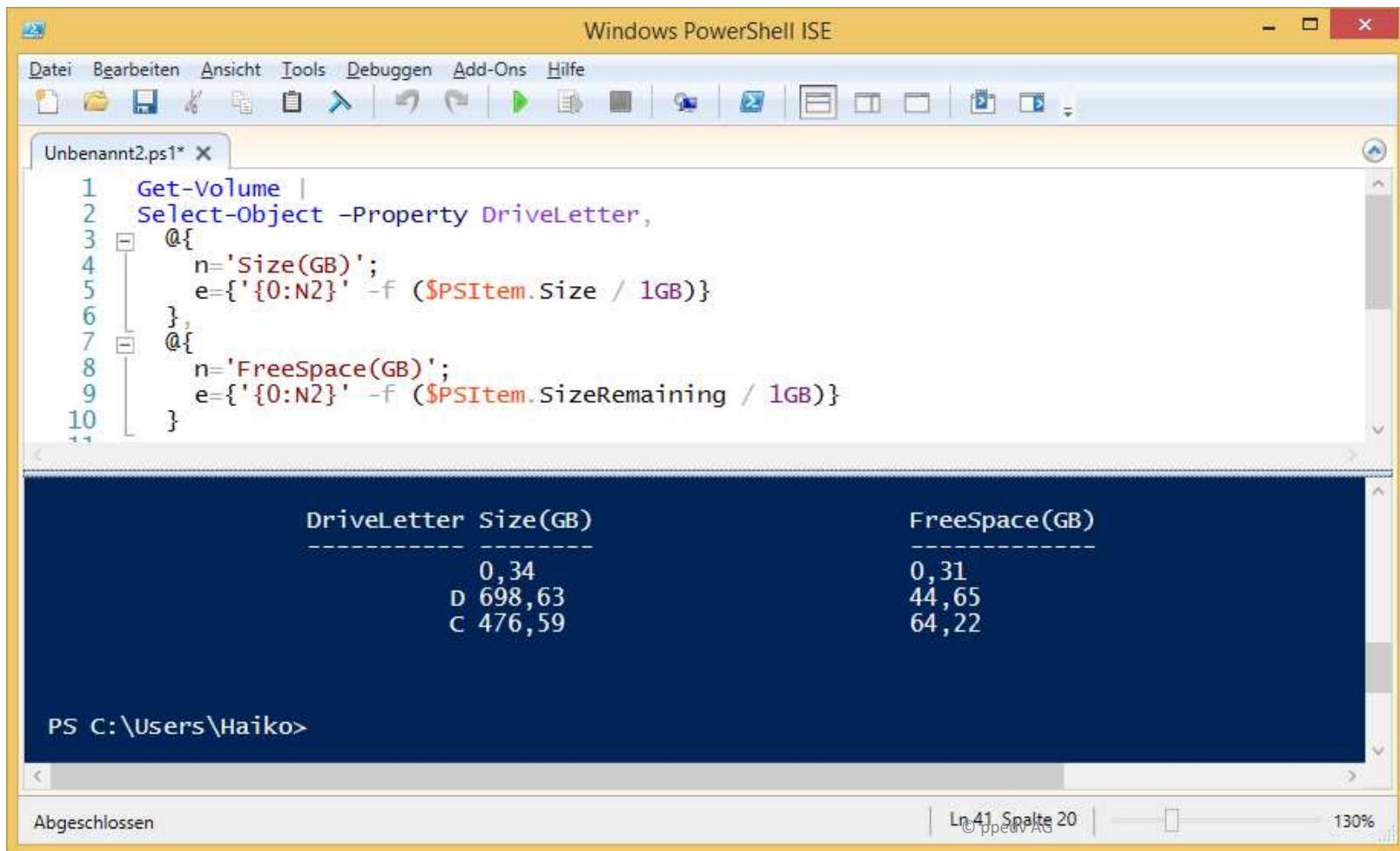
Die Pipeline – Eigene Werte berechnen

- Zum Formatieren können Abkürzungen verwendet werden:
 - **KB** kilobyte
 - **MB** megabyte
 - **GB** gigabyte
 - **TB** terabyte
 - **PB** petabyte

Die Pipeline – Eigene Werte berechnen

```
Get-Volume |  
Select-Object -Property DriveLetter,  
@{  
    n='Size(GB)';  
    e={ '{0:N2}' -f ($PSItem.Size / 1GB)}  
},  
@{  
    n='FreeSpace(GB)';  
    e={ '{0:N2}' -f ($PSItem.SizeRemaining / 1GB)}  
}
```

Die Pipeline – Eigene Werte berechnen



The screenshot shows the Windows PowerShell ISE interface. The script file 'Unbenannt2.ps1' contains the following code:

```
1 Get-Volume |
2 Select-Object -Property DriveLetter,
3 @{
4     n='Size(GB)';
5     e='{0:N2}' -f ($PSItem.Size / 1GB)
6 }
7 @{
8     n='FreeSpace(GB)';
9     e='{0:N2}' -f ($PSItem.SizeRemaining / 1GB)
10}
```

The output window displays the results for three drives:

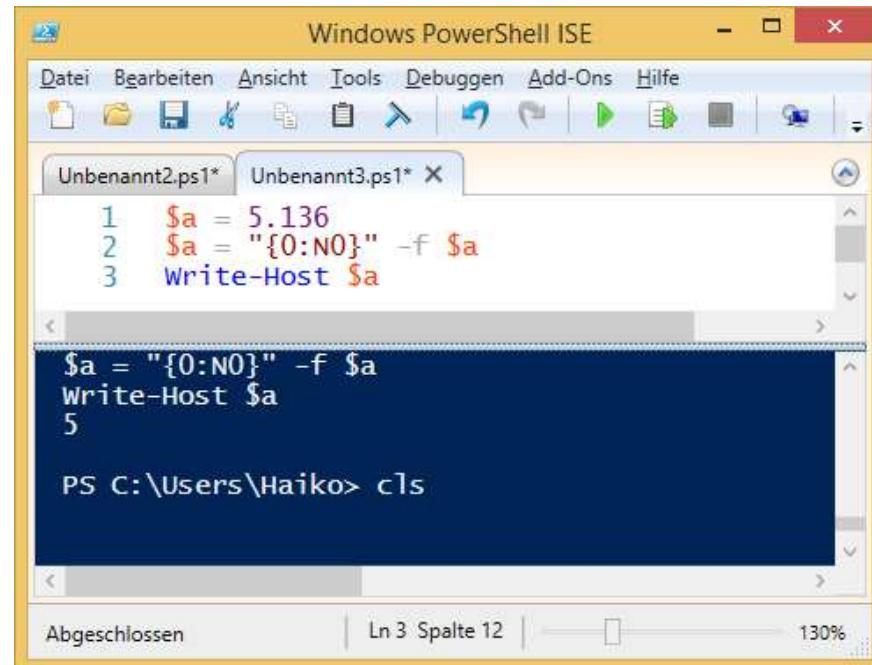
| DriveLetter | Size(GB) | FreeSpace(GB) |
|-------------|----------|---------------|
| | ----- | ----- |
| | 0,34 | 0,31 |
| D | 698,63 | 44,65 |
| C | 476,59 | 64,22 |

PS C:\Users\Haiko>

Abgeschlossen | Ln 41 Spalte 20 | 130% | 102

Zusatz: Formatieren von Zahlen

- “{0:N#}“ gibt eine Zahl (daher das „N“) mit # Nachkommastellen aus



The screenshot shows the Windows PowerShell ISE interface. In the top-left corner, there are two tabs: "Unbenannt2.ps1*" and "Unbenannt3.ps1*". The "Unbenannt3.ps1*" tab is active, displaying the following PowerShell script:

```
1 $a = 5.136
2 $a = "{0:N0}" -f $a
3 Write-Host $a
```

Below the script, the output pane shows the result of running the script:

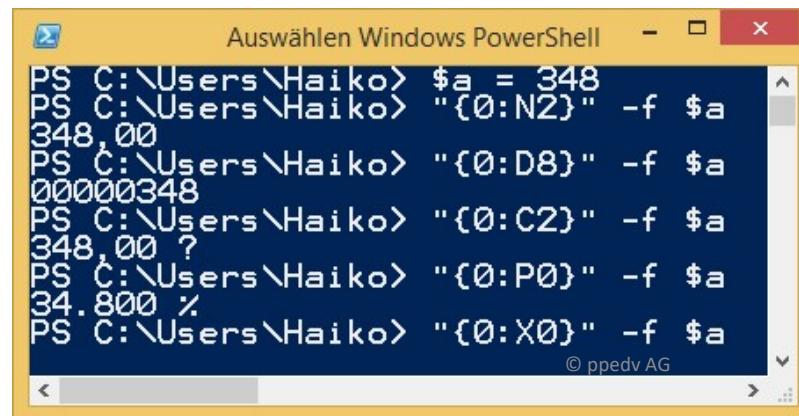
```
$a = "{0:N0}" -f $a
Write-Host $a
5
```

At the bottom of the window, status information is displayed: "Abgeschlossen" (Completed), "Ln 3 Spalte 12", and a zoom level of "130%".

Zusatz: Formatieren von Zahlen

- An Stelle von N kann auch verwendet werden:

| Name | Specifier | Description |
|-------------|-----------|--|
| Currency | C | The value is displayed as currency, using the precision specifier to indicate the number of decimal places to be displayed. |
| Decimal | D | The value is displayed using the number of digits in the precision specifier; if needed, leading zeroes are added to the beginning of the number. |
| Percentage | P | The value is multiplied by 100 and displayed as a percentage. The precision specifier indicates the number of decimal places to be displayed. |
| Hexadecimal | X | The value is displayed as a hexadecimal number. The precision specifier indicates the number of characters to be displayed, with leading zeros added to the beginning as needed. |



```
Auswählen Windows PowerShell
PS C:\Users\Haiko> $a = 348
PS C:\Users\Haiko> "{0:N2}" -f $a
348,00
PS C:\Users\Haiko> "{0:D8}" -f $a
00000348
PS C:\Users\Haiko> "{0:C2}" -f $a
348,00 ?
PS C:\Users\Haiko> "{0:P0}" -f $a
34.800 %
PS C:\Users\Haiko> "{0:X0}" -f $a
```

Übung: Pipeline II

- Aufgabe: Berechnen Sie die Summe der RAM-Auslastung aller „svchost“-Prozesse



Mögliche Lösungen

- `Get-Process -Name svchost | Measure-Object -Property PM -Sum`
- `'{0:N2}' -f ((Get-Process -Name svchost | Measure-Object -Property PM -Sum).Sum/1MB)`
- `Get-Process -Name svchost | Measure-Object -Property PM -Sum | Select-Object -Property @{n='Ram(MB)' ; e ='{0:N2}' -f ($PSItem.Sum / 1MB) } }`

Konvertieren, Importieren, Exportieren

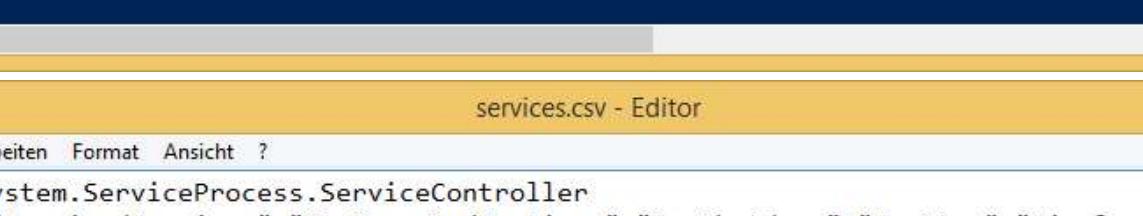
Konvertieren / Exportieren

- Mittels Konvertierung lässt sich die Ausgabe von Daten anders darstellen
- Mögliche Ausgabe-Formate sind u. A. CSV, HTML und XML
- Zwei Möglichkeiten:
 - **ConvertTo**: verändert die Form
 - **Export**: verändert die Form und schreibt die Daten extern
- **Get-Command -Verb ConvertTo,Export**

Konvertieren / Exportieren

| | |
|--------|----------------------------|
| Cmdlet | ConvertTo-Csv |
| Cmdlet | ConvertTo-Html |
| Cmdlet | ConvertTo-Json |
| Cmdlet | ConvertTo-SecureString |
| Cmdlet | ConvertTo-TpmOwnerAuth |
| Cmdlet | ConvertTo-Xml |
| Cmdlet | Export-Alias |
| Cmdlet | Export-BinaryMiLog |
| Cmdlet | Export-Certificate |
| Cmdlet | Export-Clixml |
| Cmdlet | Export-Console |
| Cmdlet | Export-Counter |
| Cmdlet | Export-Csv |
| Cmdlet | Export-FormatData |
| Cmdlet | Export-ModuleMember |
| Cmdlet | Export-PfxCertificate |
| Cmdlet | Export-PSSession |
| Cmdlet | Export-StartLayout |
| Cmdlet | Export-TlsSessionTicketKey |
| Cmdlet | Export-VM |
| Cmdlet | Export-VMSnapshot |
| Cmdlet | Export-WindowsDriver |
| Cmdlet | Export-WindowsImage |

Konvertieren / Exportieren



```
PS C:\Users\Haiko> Get-Service | Export-Csv -Path D:\services.csv
```

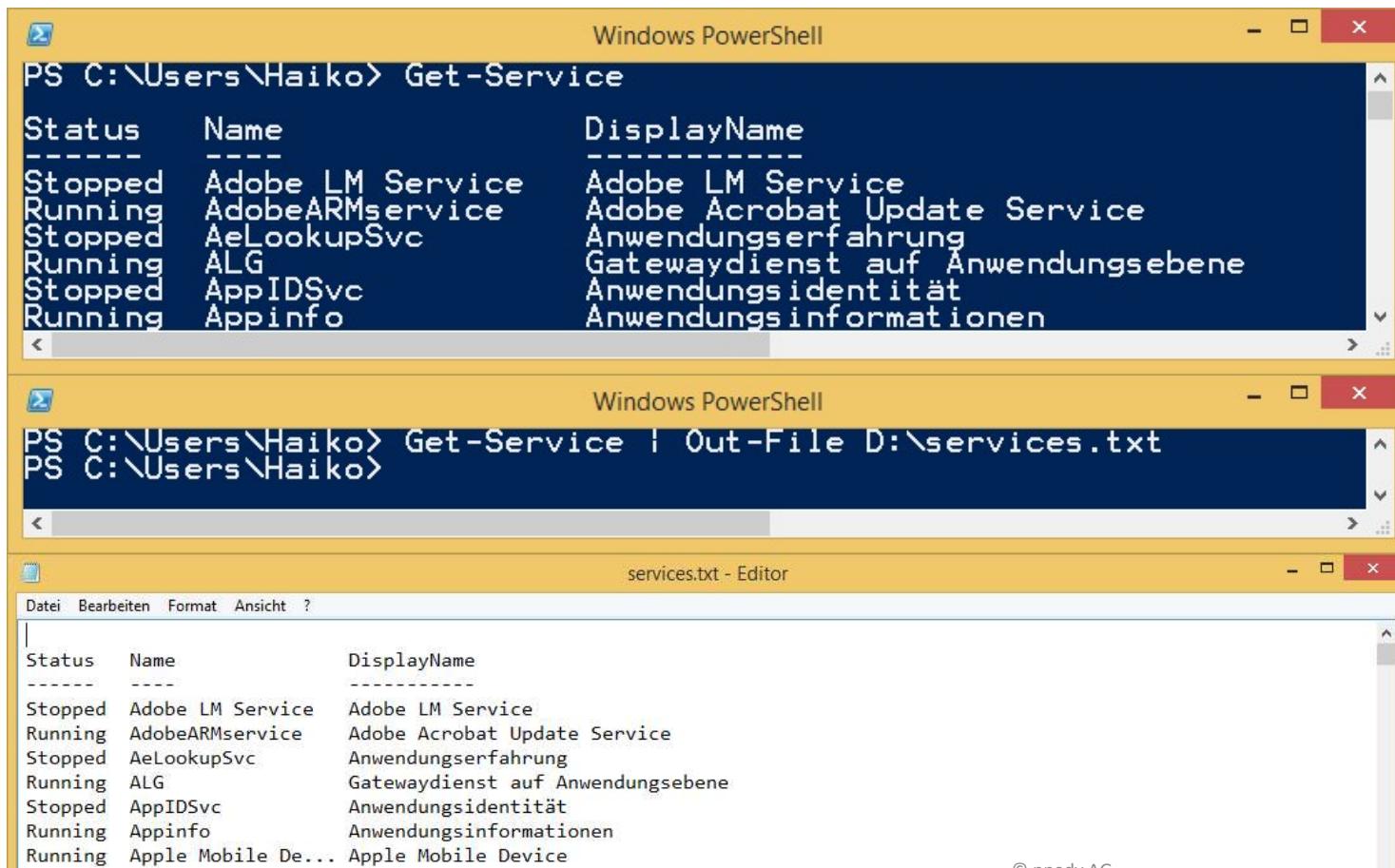
services.csv - Editor

| #TYPE | System.ServiceProcess.ServiceController |
|------------------------|---|
| "Name", | "RequiredServices", "CanPauseAndContinue", "CanShutdown", "CanStop", "DisplayName", "D |
| "Adobe LM Service", | "System.ServiceProcess.ServiceController[]", "False", "False", "False", " |
| "AdobeARMservice", | "System.ServiceProcess.ServiceController[]", "False", "False", "True", "Ad |
| "AeLookupSvc", | "System.ServiceProcess.ServiceController[]", "False", "False", "False", "Anwen |
| "ALG", | "System.ServiceProcess.ServiceController[]", "False", "False", "True", "Gatewaydienst . |
| "AppIDSvc", | "System.ServiceProcess.ServiceController[]", "False", "False", "False", "Anwendun |
| "Appinfo", | "System.ServiceProcess.ServiceController[]", "False", "False", "True", "Anwendungs |
| "Apple Mobile Device", | "System.ServiceProcess.ServiceController[]", "False", "False", "True" |
| "AppMgmt", | "System.ServiceProcess.ServiceController[]", "False", "False", "False", "Anwendung |
| "AppReadiness", | "System.ServiceProcess.ServiceController[]", "False", "False", "False", "App' |

Konvertieren / Exportieren

- **Out-File** gibt den Inhalt der Pipeline in einer Textdatei aus
- Der Inhalt der Datei entspricht dem, was auf der Konsole ausgegeben worden wäre (keine Konvertierung)
- Die Daten lassen sich dann nur noch schwer automatisch weiterverarbeiten

Konvertieren / Exportieren



The screenshot displays three windows illustrating the conversion and export of service information.

The top window is a Windows PowerShell session showing the command `Get-Service`. The output is a table:

| Status | Name | DisplayName |
|---------|------------------|-----------------------------------|
| Stopped | Adobe LM Service | Adobe LM Service |
| Running | AdobeARMservice | Adobe Acrobat Update Service |
| Stopped | AeLookupSvc | Anwendungserfahrung |
| Running | ALG | Gatewaydienst auf Anwendungsebene |
| Stopped | AppIDSvc | Anwendungsidentität |
| Running | Appinfo | Anwendungsinformationen |

The middle window is another Windows PowerShell session showing the command `Get-Service | Out-File D:\services.txt`.

The bottom window is a Microsoft Word document titled "services.txt - Editor" containing the same table of service information.

Konvertieren / Exportieren

- Der Inhalt der Pipeline kann sich mit jedem Cmdlet ändern
- Es ist wichtig zu wissen, was die Pipeline „aktuell“ enthält

Konvertieren / Exportieren Get-Service |

```
PS C:\Users\Haiko> Get-Service | Get-Member  
  
TypeName: System.ServiceProcess.ServiceController
```

Sort-Object -Property Status |

```
PS C:\Users\Haiko> Get-Service | Sort-Object -Property Status | Get-Member  
  
TypeName: System.ServiceProcess.ServiceController
```

Select-Object -Property Name,Status |

```
PS C:\Users\Haiko> Get-Service | Sort-Object -Property Status | Select-Object -Property Name,  
  
TypeName: Selected.System.ServiceProcess.ServiceController
```

ConvertTo-CSV |

```
PS C:\Users\Haiko> Get-Service | Sort-Object -Property Status | Select-Object -Property Name,Status | ConvertTo-CSV | Ge  
t-Member  
  
TypeName: System.String
```

Out-File -FilePath ServiceList.csv

Konvertieren / Exportieren

```
... | ConvertTo-CSV |      Out-File  
          D:\output.csv
```

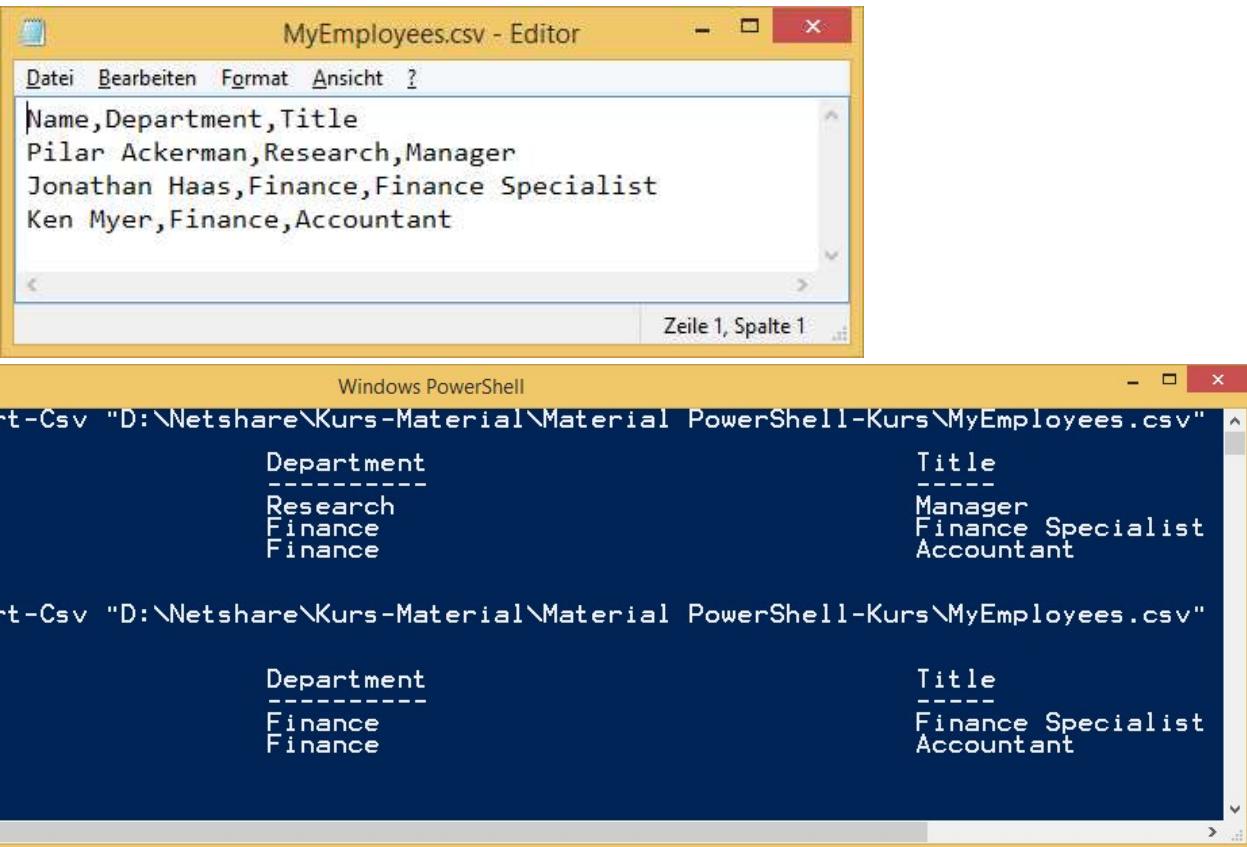
=

```
... | Export-CSV D:\output.csv
```

Importieren

- **Importieren**-Kommandos sind die Gegenteile der **Exportieren**-Kommandos
- im Allgemeinen finden zwei Schritte statt:
 - Daten raw einlesen
 - Konvertierung in passendes Format
- Bsp.: **Import-CSV**, **Import-CliXML**, ...

Importieren



The screenshot shows a Windows environment with two windows. The top window is a text editor titled "MyEmployees.csv - Editor" containing the following CSV data:

| Name | Department | Title |
|----------------|------------|--------------------|
| Pilar Ackerman | Research | Manager |
| Jonathan Haas | Finance | Finance Specialist |
| Ken Myer | Finance | Accountant |

The bottom window is a "Windows PowerShell" window with the following command and output:

```
PS C:\Users\Haiko> Import-Csv "D:\Netshare\Kurs-Material\Material PowerShell-Kurs\MyEmployees.csv"
Name          Department      Title
----          -----        -----
Pilar Ackerman    Research    Manager
Jonathan Haas     Finance    Finance Specialist
Ken Myer          Finance    Accountant

PS C:\Users\Haiko> Import-Csv "D:\Netshare\Kurs-Material\Material PowerShell-Kurs\MyEmployees.csv" | Where-Object { $_.Department -eq "Finance" }
Name          Department      Title
----          -----        -----
Jonathan Haas    Finance    Finance Specialist
Ken Myer          Finance    Accountant
```

Damit könnte man z. B. einen Massen-Import von AD-Usern durchführen!

Importieren

- **Get-Content** liest lediglich den Inhalt einer Datei
- Es findet keine Konvertierung oder Interpretierung statt
- **ConvertFrom**-Kommandos können Daten in der Pipeline von raw in ein interpretierbares Format konvertieren
- **Import**-Kommandos sind eine Art Kombination aus **Get-Content** und **ConvertFrom**

Übungen: Export / Import I

- Exportieren Sie die ersten 10 Einträge aus einem Ereignisprotokoll (z. B. „System“) in eine CSV-Datei
- Hinweis: **Get-Eventlog**

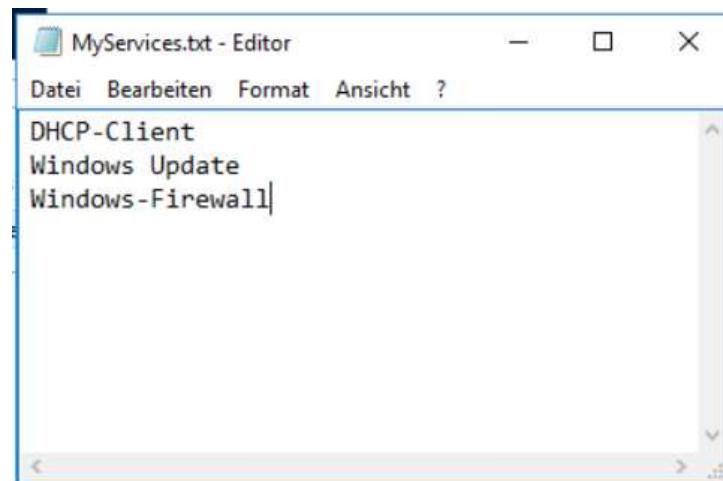


Mögliche Lösungen

- `Get-EventLog System | Select-Object -Last 10 | Export-Csv D:\Events.csv`
- `Get-EventLog System -Newest 10 | Export-Csv D:\Events.csv; Invoke-Item D:\Events.csv`

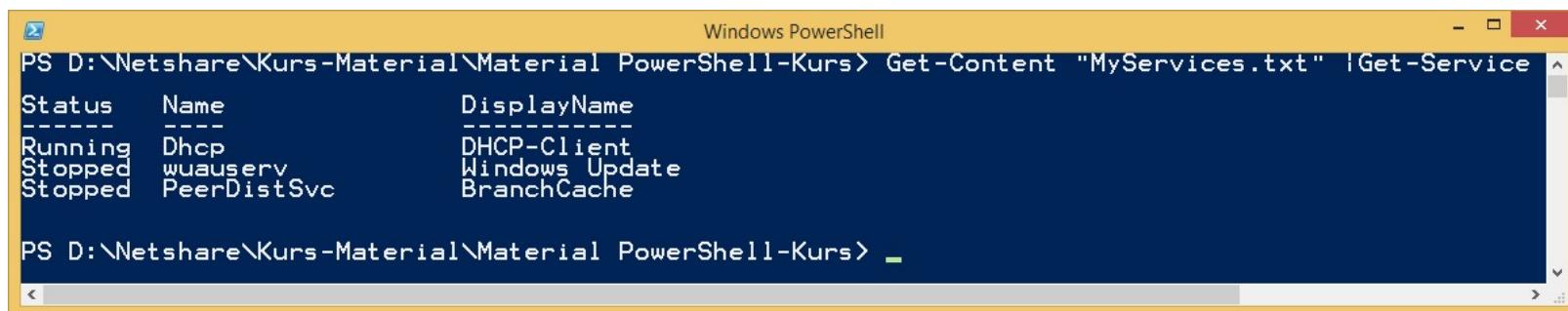
Übungen: Export / Import II

- Fragen Sie den Status von 3 Dienste, deren Namen in einer Textdatei (C:\Testfiles\MyServices.txt) angegeben sind, ab



Mögliche Lösungen

- Get-Content C:\Testfiles\MyServices.txt | Get-Service



```
Windows PowerShell
PS D:\Netshare\Kurs-Material\Material PowerShell-Kurs> Get-Content "MyServices.txt" | Get-Service
Status   Name           DisplayName
-----  --  -----
Running  Dhcp          DHCP-Client
Stopped  wuauserv      Windows Update
Stopped  PeerDistSvc  BranchCache

PS D:\Netshare\Kurs-Material\Material PowerShell-Kurs>
```

ExportTo-HTML

```

1 $a = "<style>"  

2 $a = $a + "BODY{background-color:peachpuff;}"  

3 $a = $a + "TABLE{border-width: 1px; border-style: solid; border-color: black; border-collapse: collapse;}"  

4 $a = $a + "TH{border-width: 1px; padding: 0px; border-style: solid; border-color: black; background-color:thistle}"  

5 $a = $a + "TD{border-width: 1px; padding: 0px; border-style: solid; border-color: black; background-color:PaleGoldenrod}"  

6 $a = $a + "</style>"  

7  

8 Get-Service | Select-Object Status, Name, DisplayName |  

9     ConvertTo-HTML -head $a -body "<H2>Service Information</H2>" |  

10    Out-File D:\Test.htm  

11  

12 Invoke-Expression D:\Test.htm

```

Service Information

| Status | Name | DisplayName |
|---------|----------------------|---|
| Stopped | Adobe LM Service | Adobe LM Service |
| Running | AdobeARMService | Adobe Acrobat Update Service |
| Stopped | AeLookupSvc | Anwendungserfahrung |
| Running | ALG | Gatewaydienst auf Anwendungsebene |
| Stopped | AppIDSvc | Anwendungsidentität |
| Running | Appinfo | Anwendungsinformationen |
| Running | Apple Mobile Device | Apple Mobile Device |
| Stopped | AppMgmt | Anwendungsverwaltung |
| Stopped | AppReadiness | App-Vorbereitung |
| Stopped | AppXSvc | AppX-Bereitstellungsdienst (AppXSVC) |
| Running | AudioEndpointBuilder | Windows-Audio-Endpunktterstellung |
| Running | Audiosrv | Windows-Audio |
| Stopped | AxInstSV | ActiveX-Installer (AxInstSV) |
| Stopped | BDESVC | BitLocker-Laufwerkverschlüsselungsdienst |
| Running | BFE | Basisfiltermodul |
| Running | BITS | Intelligenter Hintergrundübertragungsdienst |
| Stopped | Bonjour Service | Dienst "Bonjour" |
| Running | BrokerInfrastructure | Infrastrukturdienst für Hintergrundaufgaben |

5
★

TRAINING



Nachtrag

Starten einer Applikation

```
1 $IE = 'C:\Program Files\Internet Explorer\iexplore.exe'
2 #Invoke-Expression $IE -> Geht nicht wg. Leerzeichen
3 #Invoke-Command $IE _> Geht nicht wg. Leerzeichen
4 Invoke-Item $IE #Geht
5 &$IE #Geht
6 Start-Process $IE #Geht
7 Invoke-WmiMethod -Class Win32_Process -Name Create -ArgumentList $IE #Geht
```

5
★

TRAINING



Filtern

Filtern

- Vergleichsoperatoren:

| Vergleich | Case-Insensitive | Case-Sensitive |
|---------------------|------------------|----------------|
| Ist gleich | -eq | -ceq |
| Ist ungleich | -ne | -cne |
| Größer als | -gt | -cgt |
| Kleiner als | -lt | -clt |
| Größer oder gleich | -ge | -cge |
| Kleiner oder gleich | -le | -cle |
| Ähnlich / Wildcard | -like | -clike |

- „Traditionelle“ Operatoren wie <,>,=,==, != haben keine Gültigkeit in PowerShell!

Filtern

- Für die Filterung wird **Where-Object** (Alias: **Where**) verwendet (es gibt kein Filter-Object!)
- Basis-Syntax:
 - `Get-Service | Where Status -eq Running`
 - `Get-Process | Where CPU -gt 20`
 - Es wird nur ein einziger Vergleich unterstützt
 - Es können u.U. nicht alle Attribute abgeprüft werden
 - `Get-Service | Where Name.Length -gt 5` wird nicht funktionieren!
 - Aber so: `Get-Service | Where {$_.Name.Length -gt 20}`

Filtern

- Erweiterte Syntax:

- Erlaubt mehrere Bedingungen
- Benötigt ein „Filter-Skript“, welches True oder False zurückgibt
- Im Skript kann mit **\$PSItem** oder **\$_** auf das jeweilige Objekt der Pipeline verwiesen werden

- Beispiele:

- `Get-Service | Where-Object -Filter { $PSItem.Status -eq 'Running' }`
- `Get-Service | Where { $_.Status -eq 'Running' }`
- `Get-Service | ? { $PSItem.Status -eq 'Running' }`

Filtern

- Anmerkung:
 - `$_` und `$PSItem` haben die selbe Funktion
 - `$PSItem` wurde erst in PS3.0 eingeführt (besser zu lesen)
 - Für Abwärtskompatibilität: Besser `$_`

Übung: Filtern

- Fragen Sie alle Dienste ab, deren Namen mehr als 15 Zeichen enthält
- Hinweis: **.Length** liefert die Länge eines Strings



Mögliche Lösungen

```
Get-Service | Where { $_.Name.Length -gt 15 }
```



The screenshot shows a Windows PowerShell window titled "Windows PowerShell". The command entered is "PS C:\Users\Haiko> Get-Service | Where { \$_.Name.Length -gt 15 }". The output is a table listing services whose names are longer than 15 characters:

| Status | Name | DisplayName |
|---------|--------------------|--|
| Stopped | Adobe LM Service | Adobe LM Service |
| Running | Apple Mobile De... | Apple Mobile Device |
| Running | AudioEndpointBu... | Windows-Audio-Endpunktterstellung |
| Running | BrokerInfrastru... | Infrastrukturdienst für Hintergrund... |
| Running | DeviceAssociati... | Gerätezuordnungsdienst |
| Running | DisplayLinkService | DisplayLinkManager |
| Stopped | FontCache3.0.0.0 | Windows Presentation Foundation-Sch... |

Filtern II

- Filterung auch mit mehreren Kriterien möglich
- Verknüpfung durch boolsche Operatoren **-and** bzw. **-or**
- Bsp.:

```
Get-Volume | Where-Object -Filter {  
    $PSItem.HealthStatus -ne 'Healthy'  
    -or  
    $PSItem.SizeRemaining -lt 100MB  
}
```

Filtern III

- Zur Optimierung der Geschwindigkeit:
 - Filterung so früh wie möglich in der Pipeline (reduziert Anzahl der weiterverarbeiteten Objekte)
 - Wenn Commandlets eigene Filter-Parameter haben, sollten diese an Stelle von **Where-Object** genutzt werden

Übung: Filtern

- Lassen Sie alle AD-Benutzer ausgeben, deren Vorname „Paul“ lautet!



Mögliche Lösungen

- `Get-ADUser -Filter {GivenName -eq "Paul"}`
- `Get-ADUser -Filter * | Where-Object GivenName -eq "Paul"`

Aufzählen

Aufzählen

- Stellen Sie sich eine Sammlung von Objekten vor...
- ...bei denen Sie eine bestimmte Aktion mit jedem Objekt einzeln durchführen wollen.
- Kommando: **ForEach-Object** (Aliase: **ForEach** und **%**)
- u. A. nützlich, wenn die Objekte eigene Methode(n) haben

Aufzählen

```
Get-ChildItem -Path C:\Example -File |  
ForEach-Object -MemberType Encrypt
```

```
Get-ChildItem -Path C:\Example -File |  
ForEach Encrypt
```

```
Get-ChildItem -Path C:\Example -File |  
% -MemberType Encrypt
```

Aufzählen

- Auch hier gibt es eine erweiterte Syntax:
- `Get-ChildItem C:\Test -File | ForEach-Object { $PSItem.Encrypt() }`
- Dadurch entstehen mehr Möglichkeiten, u. A. wenn die Objekte keine eigene Methoden haben
- Verweis auf Objekte mittels **\$PSItem** oder **\$_**

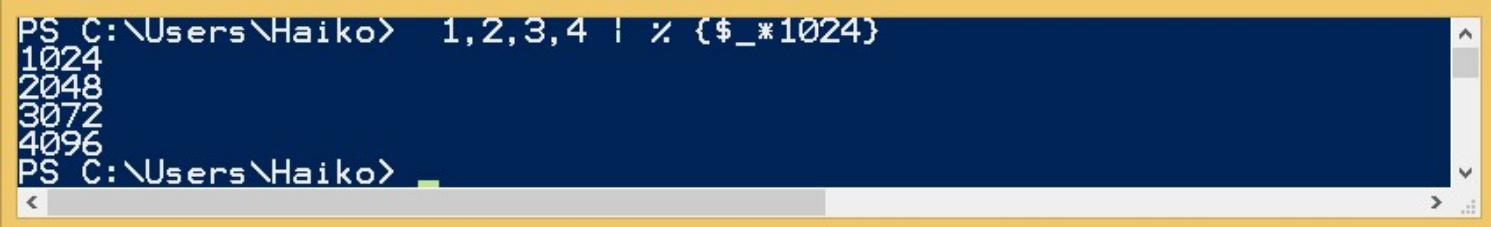
Aufzählen

So (oder ähnlich) löst man das „üblicherweise“:

```
$array = 1,2,3,4
$array.Length
for ($i=0;$i -lt $array.length; $i++)
{
    Write-Host $i
    $array[$i] * 1024
}
```

Aufzählen

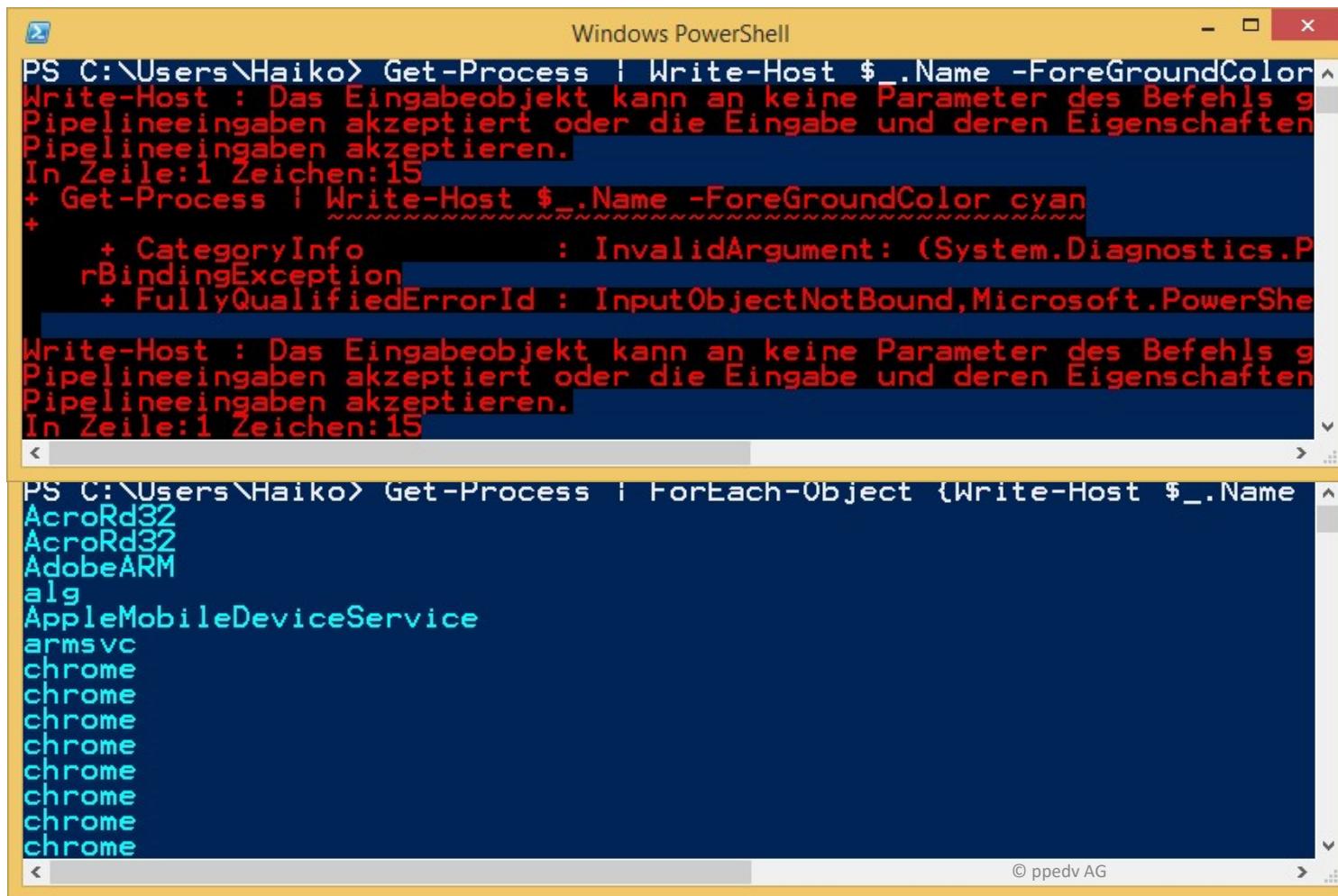
- 1,2,3,4 | % {\$_* * 1024}



```
PS C:\Users\Haiko> 1,2,3,4 | % {$_* * 1024}
1024
2048
3072
4096
PS C:\Users\Haiko>
```

- Geht nicht:
 - Get-Process | Write-Host \$_.Name -ForegroundColor cyan
- Geht:
 - Get-Process | ForEach-Object {Write-Host \$_.Name -ForegroundColor cyan}

Aufzählen



The screenshot shows two separate sessions in a Windows PowerShell window.

The top session demonstrates an error when trying to pipe the output of `Get-Process` directly to `Write-Host`. The command is:

```
PS C:\Users\Haiko> Get-Process | Write-Host $_.Name -ForegroundColor
```

The output is:

```
Write-Host : Das Eingabeobjekt kann an keine Parameter des Befehls g  
Pipelineeingaben akzeptiert oder die Eingabe und deren Eigenschaften  
Pipelineeingaben akzeptieren.  
In Zeile:1 Zeichen:15
```

The bottom session shows the correct way to pipe the output of `Get-Process` to `ForEach-Object`, which then calls `Write-Host` on each process object. The command is:

```
PS C:\Users\Haiko> Get-Process | ForEach-Object {Write-Host $_.Name}
```

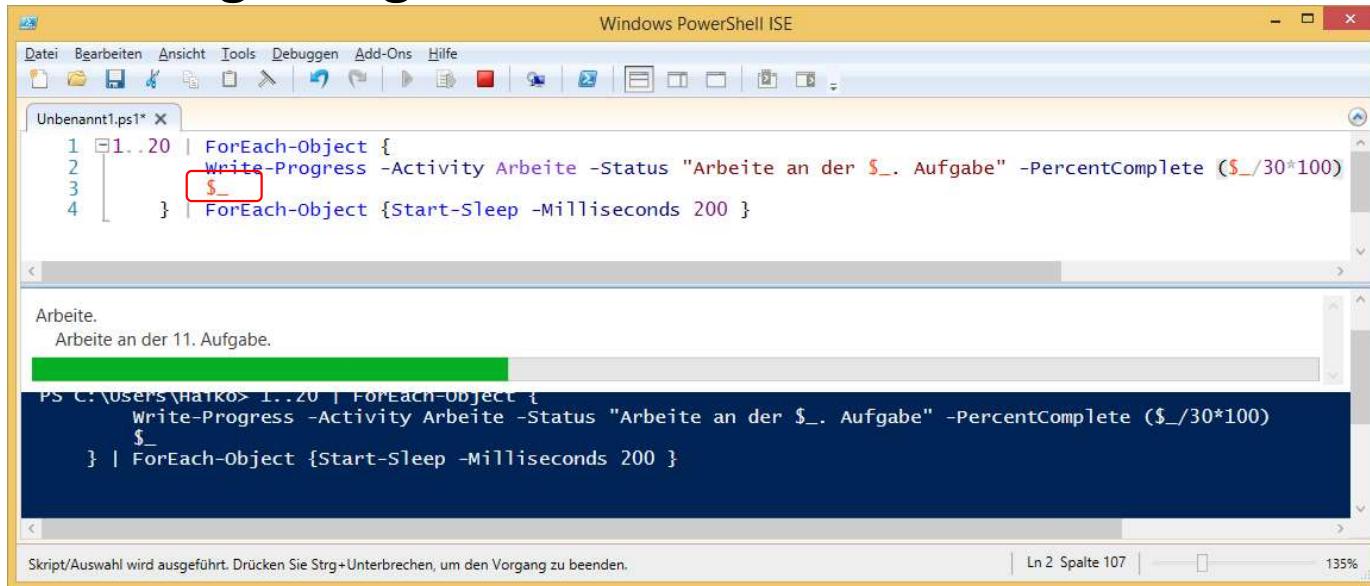
The output is:

```
AcroRd32  
AcroRd32  
AdobeARM  
alg  
AppleMobileDeviceService  
armsvc  
chrome  
chrome  
chrome  
chrome  
chrome  
chrome  
chrome  
chrome
```

At the bottom right of the window, it says `© ppdev AG`.

Aufzählen

- Fortschrittsanzeige möglich



The screenshot shows the Windows PowerShell ISE interface. A script named 'Unbenannt1.ps1' is open in the editor. The code uses a ForEach-Object loop to process 1..20 items. For each item, it writes a progress message and then sleeps for 200 milliseconds. A red box highlights the status parameter '\$_. Aufgabe' in the Write-Progress command. The PowerShell window below shows the progress bar filling up, indicating the script is running. The status bar at the bottom of the window displays the message 'Skript/Auswahl wird ausgeführt. Drücken Sie Strg+Unterbrechen, um den Vorgang zu beenden.'

```
Windows PowerShell ISE
Datei Bearbeiten Ansicht Tools Debuggen Add-Ons Hilfe
Unbenannt1.ps1*
1 .. 20 | ForEach-Object {
2     Write-Progress -Activity Arbeit -Status "Arbeit an der $_. Aufgabe" -PercentComplete ($_/30*100)
3     $_
4 } | ForEach-Object {Start-Sleep -Milliseconds 200 }

Arbeit.
Arbeit an der 11. Aufgabe.

PS C:\Users\HATKO> 1..20 | ForEach-Object {
    Write-Progress -Activity Arbeit -Status "Arbeit an der $_. Aufgabe" -PercentComplete ($_/30*100)
    $_
} | ForEach-Object {Start-Sleep -Milliseconds 200 }

Skript/Auswahl wird ausgeführt. Drücken Sie Strg+Unterbrechen, um den Vorgang zu beenden.
Ln 2 Spalte 107 | 135%
```

- „Zurücklegen in die Pipe“ wichtig!

Aufzählen

```
1..100 | ForEach-Object {  
    Write-Progress -Activity "Zähle" -Status  
    "Arbeite gerade an Zahl $_" -PercentComplete  
    $_;  
    Start-Sleep -Milliseconds 50;  
    $_  
}
```

Übung: Aufzählen und co.

- Testen Sie diese 5 Aufrufe:

- Get-Service | Select-Object -Property Name
- Gsv | Select Name
- Get-Service | ForEach Name
- Get-Service | % { \$_.Name }
- Get-Service | ft name

- Was geschieht? Warum?



Übung: Aufzählen und Co.

- In der „optischen Wirkung“ gleich
- Bei genauer Betrachtung:
 - Unterschiedliche Arbeitsweise (damit in der Regel auch Performance-Unterschiede)
 - Objekttypen sind am Ende unterschiedlich!!

5
★

TRAINING



Schleifen

Schleifen

- Mehrere Arten von Schleifen

- For-Schleife
- ForEach-Schleife
- While-Schleife
- Do-While-Schleife
- Do-Until-Schleife

Schleifen

- **For-Schleife** läuft eine bestimmte Anzahl Iterationen:
 - `for ($i = 1; $i -le 10; $i++) { $i }`
 - Läuft, bis Zähler i nicht mehr „le“ (less or equal) 10 ist
 - Schleife endet also, nachdem i den Wert 10 angenommen hat und den Schleifenkörper durchlaufen hat

Schleifen

- **ForEach-Schleife** läuft über eine Menge von Objekten („Array“) deren Anzahl in der Regel beim Programmieren noch nicht bekannt war
- Die Eigenschaften der Objekte in der Menge können auch bei der Abarbeitung geändert werden
- `$Users = Get-ADUser -Filter *`
- `ForEach ($EinUser in $Users) {`
 `$EinUser.Givenname }`
- Achtung: Nicht mit **ForEach-Object** verwechseln!

Schleifen

- **While-Schleifen** sind „kopfgeprüft“ („abweisend“)
- Schleifenkörper wird nur durchlaufen, wenn **vorher** die Bedingung wahr ist
- Bedingung kann auch ein Commandlet o.ä. sein
- `While(($Eingabe = Read-Host -Prompt „Soll ich weitermachen?“) -eq „Ja“)`
- `{ #Schleifenkörper }`

Schleifen

- **Do-While-Schleifen** sind „fußgeprüft“ („nicht-abweisend“)
- Schleifenkörper wird in jedem Fall min. 1x durchlaufen
- Bedingung für weiteren Durchlauf wird am Ende geprüft
- do { #Anweisung } while (#Bedingung)

Schleifen

- **Do-Until-Schleifen** sind „fußgeprüft“ („nicht-abweisend“)
- Schleifenkörper wird in jedem Fall min. 1x durchlaufen
- Bedingung für weiteren Durchlauf wird am Ende geprüft
- `do { #Anweisung } until (#Bedingung)`
- Quasi die „Umkehrung“ von **Do-While**

Schleifen

- Bei den meisten Schleifen: Abbruch innerhalb des Schleifenkörpers möglich
- Dazu: **break**

```
for ($i = 1; $i -le 10; $i++) {  
    $i  
    If ($i -eq 5) { break }  
}
```

Übung: Schleifen

- Erzeugen Sie eine Schleife, die alle geraden Zahlen von 2 bis 10 ausgibt!



Mögliche Lösungen

```
for ($i=1; $i -le 5; $i++)  
{ Write-Host „$($i*2)`“ }
```

```
$i = 1  
while ($i -lt 6) {  
    Write-Host "$($i*2)"; $i++ }
```

```
for ($i=2; $i -le 10; $i=$i+2)  
{ Write-Host „$i“ }
```

Mögliche Lösungen

```
for ($i=1; $i -le 10; $i++)  
{  
    if ($i%2 -eq 0)  
    {  
        $i  
    }  
}
```

Wiederholung 1. Tag

Wiederholung

- Mit welchen Kommandos kann man neue/unbekannte Commandlets und deren Funktionsweise finden?
- Was sind Module? Wie sollte man damit verfahren?
- Was genau ist die „Pipeline“ und was geschieht in ihr?
- Wie werden Variablen definiert?
- Welche Vergleichsoperatoren gibt es in PowerShell?

Wiederholung

- Womit kann man die Ausgabe formatieren? Was ist dabei zu beachten?
- Wie kann man die ausgegebenen Objekte sortieren und/oder selektieren?
- Welche Möglichkeiten gibt es bzgl. Import und Export von Daten?
- Was ist „Aufzählen“? Wozu benötigt man das?

Verzweigungen u. Ä.

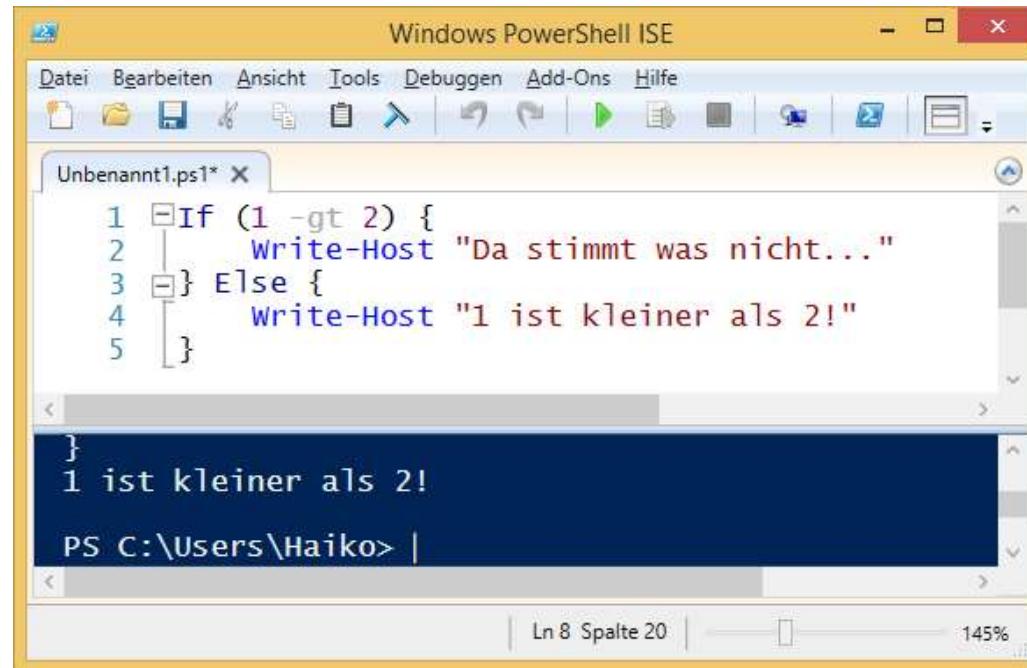
Verzweigungen

- If-Then-Else Konstruktion ähnlich anderer Programmiersprachen:

```
If ($this -eq $that) {  
    # Condition 1  
} ElseIf ($those -gt $these) {  
    # Condition 2  
} ElseIf ($this -lt $that) {  
    # Condition 3  
} Else {  
    # Condition 4  
}
```

Verzweigungen

- Demo: If-Then-Else



The screenshot shows the Windows PowerShell ISE interface. The code editor window contains a script named "Unbenannt1.ps1" with the following content:

```
1 IF (1 -gt 2) {  
2     Write-Host "Da stimmt was nicht..."  
3 } Else {  
4     Write-Host "1 ist kleiner als 2!"  
5 }
```

The output pane below shows the execution results:

```
}
```

1 ist kleiner als 2!

PS C:\Users\Haiko> |

Verzweigungen

- Switch:

```
$a = 5
switch ($a)
{
    1 {"The color is red."}
    2 {"The color is blue."}
    3 {"The color is green."}
    4 {"The color is yellow."}
    5 {"The color is orange."}
    6 {"The color is purple."}
    7 {"The color is pink."}
    8 {"The color is brown."}
    default {"The color could not be determined."}
}
```

Die Pipeline II

Die Pipeline II

- Übergabe zwischen 2 Commandlets ist im Hintergrund etwas komplexer
- Verständnis dafür oft nötig
- Commandlets akzeptieren nur Eingaben durch ihre eigenen Parameter, nicht „von außen“
- Bei Übergabe durch Pipeline ist ein „Mapping“ der Attribute des Eingangs-Objekts auf die erwarteten Attribute des Ausgangs-Objektes nötig

Die Pipeline II

- Ein kleines Beispiel:
 - Get-MailBox | Export-MailBox –PstFolderPath D:\
 - Export-MailBox erwartet einen Parameter „Identity“ zur Identifizierung, wessen Postfach exportiert werden soll
 - Get-MailBox liefert sehr viele Parameter
 - Zuordnung nötig

Die Pipeline II

- Es gibt 2 Techniken, wie Parameter in der Pipeline gebunden werden können:
 - **ByValue** – Standard, wird immer zuerst versucht
 - **ByPropertyName** – wenn **ByValue** fehlschlägt
- **Get-Help -Full** listet auf, welche Parameter eine Pipelineeingabe akzeptieren und mit welcher Technik

Die Pipeline II

```
PS C:\Users\Haiko> Get-Help Get-Service -Full

NAME
  Get-Service

ÜBERSICHT
  Gets the services on a local or remote computer.

SYNTAX
  Get-Service [[-Name] <String[]>] [-ComputerName <String[]>] [-DependentServices]
  <String[]> [-RequiredServices] [<CommonParameters>]

PARAMETER
  -ComputerName <String[]>
    Gets the services running on the specified computers. The default is the local computer, type the computer name, a dot (.), or "localhost". This parameter does not rely on Windows PowerShell remoting. You can use the Get-Service even if your computer is not configured to run remote commands.

    Erforderlich?          false
    Position?              named
    Standardwert           local computer
    Pipelineeingaben akzeptieren?true (ByPropertyName)
    Platzhalterzeichen akzeptieren?false

  -Name <String[]>
    Specifies the service names of services to be retrieved. Wildcards are permitted for all of the services on the computer.

    Erforderlich?          false
    Position?              1
    Standardwert           All services
    Pipelineeingaben akzeptieren?true (ByPropertyName, ByValue)
    Platzhalterzeichen akzeptieren?true

  -RequiredServices [<SwitchParameter>]
    Gets only the services that this service requires.

    This parameter gets the value of the ServicesDependedOn property of the service.
```

Die Pipeline II

String-Objekte in der Pipeline



"BITS", "WinRM" | Get-Service -Name



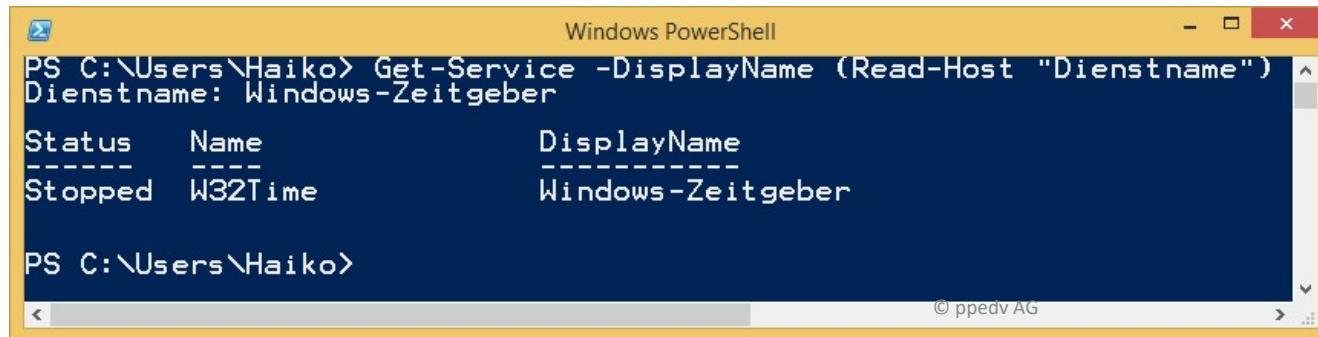
Werden unsichtbar an einen Parameter übergeben, der Strings aus der Pipeline "byValue" akzeptiert

Die Pipeline II

- **Object** und **PSObject** sind generische Objekt-Typen
- Wenn ein Parameter einen dieser Objekt-Typen akzeptiert, akzeptiert er jedes Objekt
- Wenn ein Parameter Pipeline-Input **ByValue** akzeptiert, dann kann er jedes Objekt von der Pipeline annehmen

Die Pipeline II

- An Stelle von „echten“ Werten für die Parameter können auch PowerShell-Aufrufe stehen, die entsprechende Werte liefern:
- Get-Process -Name (Get-Content Names.txt)
- Get-Service -Name (Read-Host „Dienstname“)



```
Windows PowerShell
PS C:\Users\Haiko> Get-Service -DisplayName (Read-Host "Dienstname")
Dienstname: Windows-Zeitgeber
Status     Name               DisplayName
-----   ----              -----
Stopped   W32Time           Windows-Zeitgeber

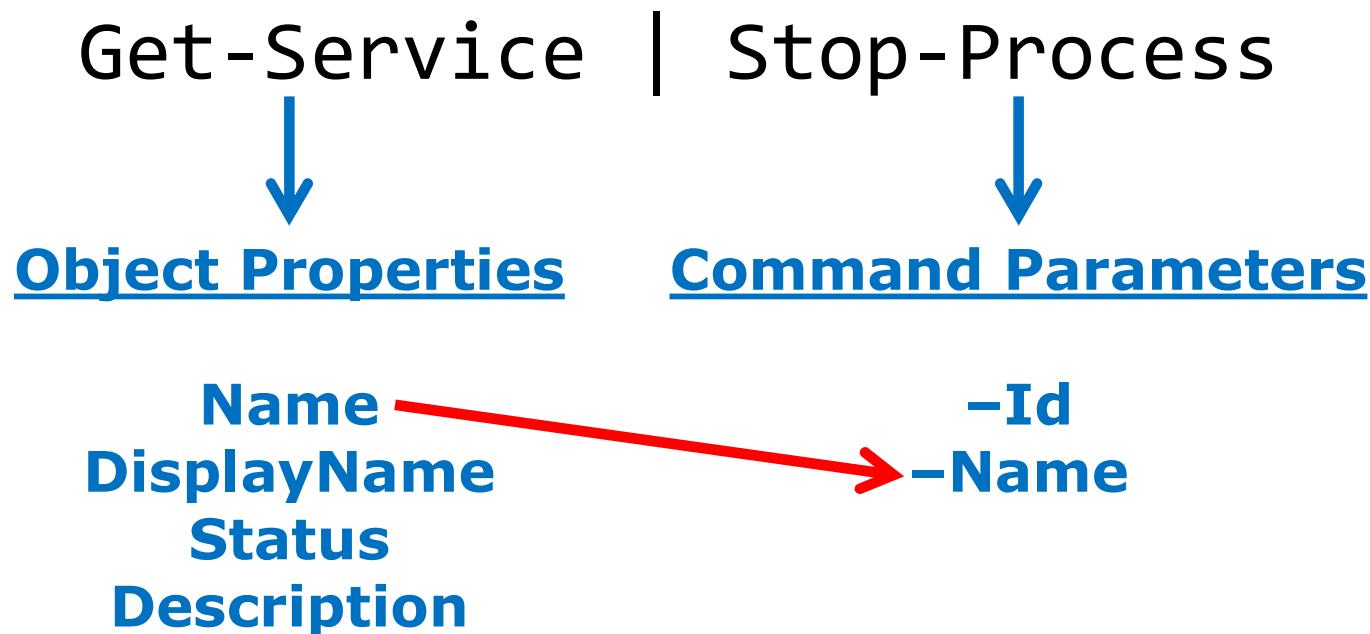
PS C:\Users\Haiko>
```

Die Pipeline II

- `Get-Service -ComputerName
c:\Computers.txt)` (Get-Content

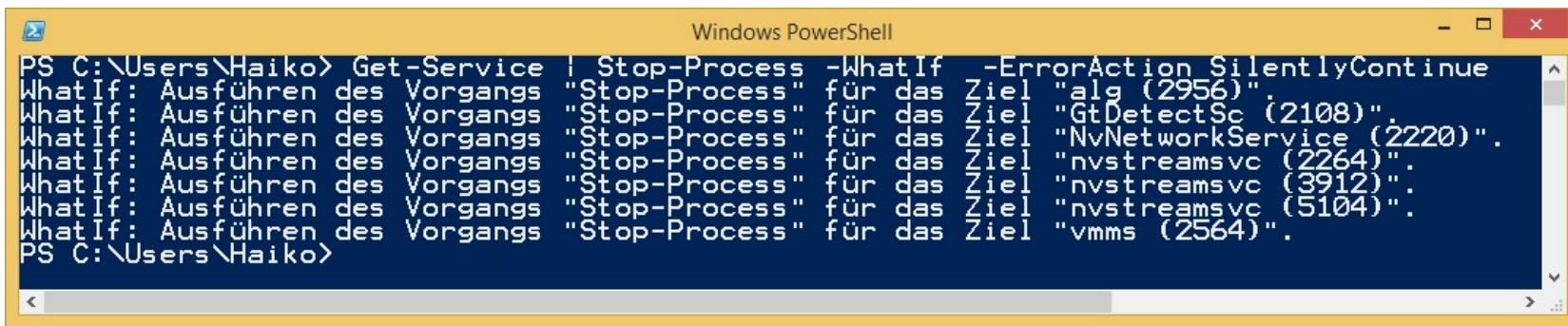
Die Pipeline II

- Wechsel zu ByPropertyName:



Die Pipeline II

- Get-Service | Stop-Process scheint wenig sinnvoll
- In der Tat wirft es viele Fehler, aber:



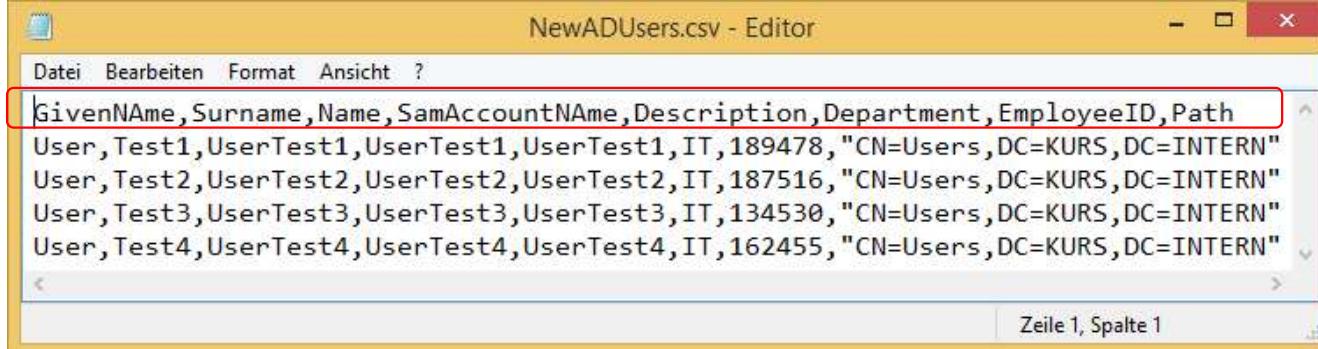
```
Windows PowerShell
PS C:\Users\Haiko> Get-Service | Stop-Process -WhatIf -ErrorAction SilentlyContinue
WhatIf: Ausführen des Vorgangs "Stop-Process" für das Ziel "alg (2956)".
WhatIf: Ausführen des Vorgangs "Stop-Process" für das Ziel "GtDetectSc (2108)".
WhatIf: Ausführen des Vorgangs "Stop-Process" für das Ziel "NvNetworkService (2220)".
WhatIf: Ausführen des Vorgangs "Stop-Process" für das Ziel "nvstreamsvc (2264)".
WhatIf: Ausführen des Vorgangs "Stop-Process" für das Ziel "nvstreamsvc (3912)".
WhatIf: Ausführen des Vorgangs "Stop-Process" für das Ziel "nvstreamsvc (5104)".
WhatIf: Ausführen des Vorgangs "Stop-Process" für das Ziel "vmms (2564)".
PS C:\Users\Haiko>
```

Die Pipeline II

- Die **-Full** Hilfe eines Commandlets listet auf, welche Parameter eine Eingabe durch **ByPropertyName** zulassen:

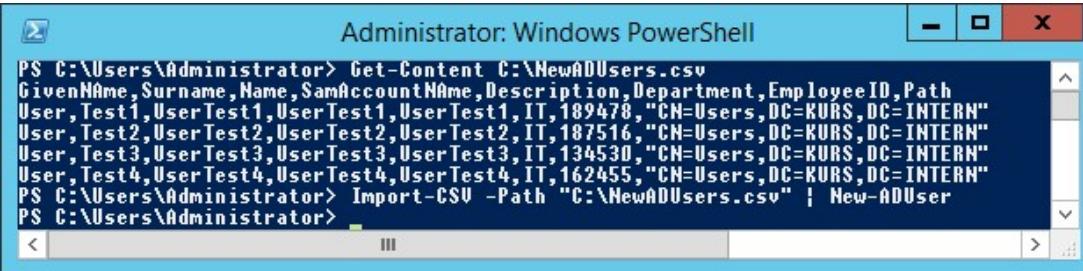
```
PARAMETER
    -ComputerName <String[]>
        Gets the services running on the specified computers. The default is the local computer. Type the NetBIOS name, an IP address, or a fully qualified domain name of a remote computer, type the computer name, a dot (.), or "localhost". This parameter does not rely on Windows PowerShell remoting. You can use the Get-Service cmdlet even if your computer is not configured to run remote commands.
        Erforderlich?           false
        Position?               named
        Standardwert            Local Computer
        Pipelineeingaben akzeptieren? true (ByPropertyName)
        Platzhalterzeichen akzeptieren? false
```

Die Pipeline II



| GivenName | Surname | Name | SamAccountName |
|-----------|---------|-----------|----------------|
| User | Test1 | UserTest1 | UserTest1 |
| User | Test2 | UserTest2 | UserTest2 |
| User | Test3 | UserTest3 | UserTest3 |
| User | Test4 | UserTest4 | UserTest4 |

```
Import-Csv -Path "C:\NewADUsers.csv" |  
    New-ADUser
```



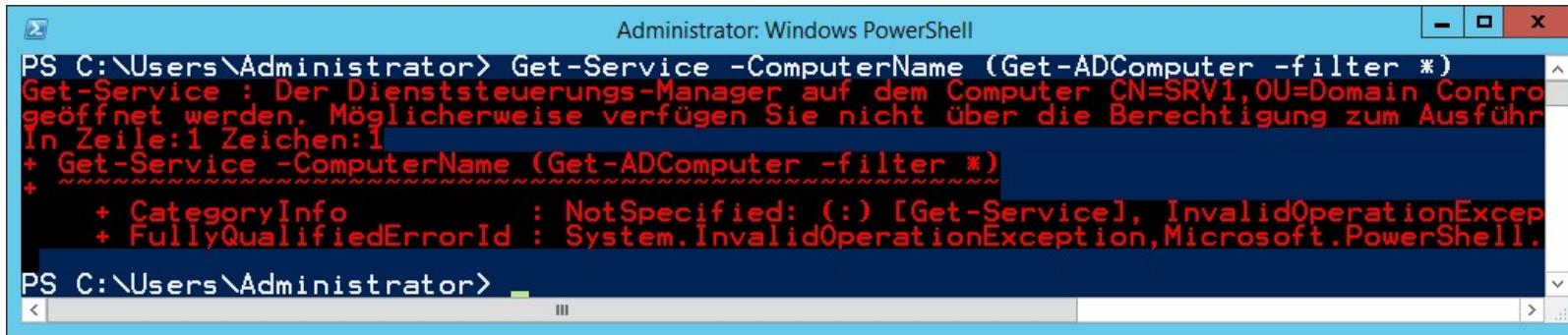
```
PS C:\Users\Administrator> Get-Content C:\NewADUsers.csv  
GivenName,Surname,Name,SamAccountName,Description,Department,EmployeeID,Path  
User,Test1,UserTest1,UserTest1,UserTest1,IT,189478,"CN=Users,DC=KURS,DC=INTERN"  
User,Test2,UserTest2,UserTest2,UserTest2,IT,187516,"CN=Users,DC=KURS,DC=INTERN"  
User,Test3,UserTest3,UserTest3,UserTest3,IT,134530,"CN=Users,DC=KURS,DC=INTERN"  
User,Test4,UserTest4,UserTest4,UserTest4,IT,162455,"CN=Users,DC=KURS,DC=INTERN"  
PS C:\Users\Administrator> Import-Csv -Path "C:\NewADUsers.csv" | New-ADUser  
PS C:\Users\Administrator>
```

Zuordnung zwischen CSV-“Spalten” und PS-Attributen
geschieht **ByPropertyName**

Die Pipeline II

- Expandieren von Attributen durch

-ExpandProperty



The screenshot shows a Windows PowerShell window titled "Administrator: Windows PowerShell". The command entered was "Get-Service -ComputerName (Get-ADComputer -filter *)". The output is in German and reads:

```
PS C:\Users\Administrator> Get-Service -ComputerName (Get-ADComputer -filter *)
Get-Service : Der Dienststeuerungs-Manager auf dem Computer CN=SRV1,OU=Domain Control geöffnet werden. Möglicherweise verfügen Sie nicht über die Berechtigung zum Ausführen.
In Zeile:1 Zeichen:1
+ Get-Service -ComputerName (Get-ADComputer -filter *)
+ ~~~~~
+ CategoryInfo          : NotSpecified: (:) [Get-Service], InvalidOperationException
+ FullyQualifiedErrorId : System.InvalidOperationException,Microsoft.PowerShell.
```

PS C:\Users\Administrator>

Die Pipeline II

- Fehler, weil **Get-Service** für **-ComputerName** einen String erwartet...



... Get-ADComputer aber ein Objekt ADComputer liefert:

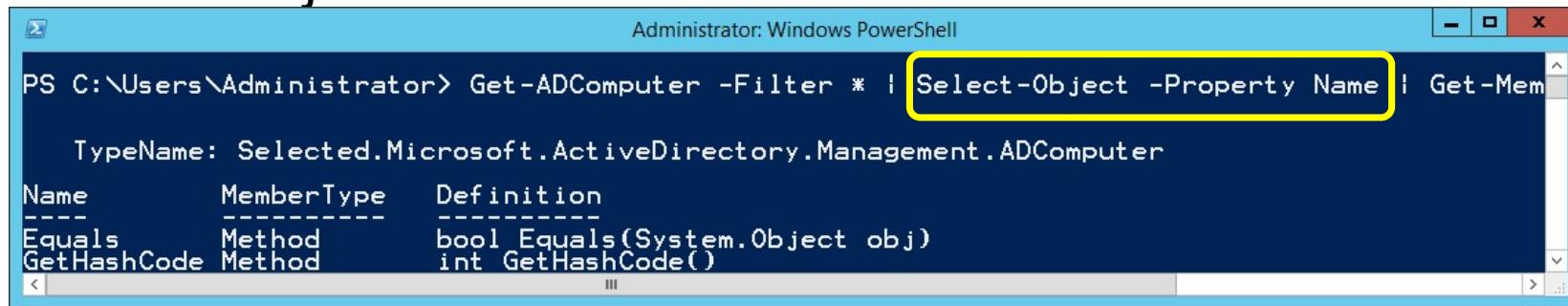


```
Administrator: Windows PowerShell
PS C:\Users\Administrator> Get-ADComputer -Filter * | Get-Member

 TypeName: Microsoft.ActiveDirectory.Management.ADComputer
Name          MemberType      Definition
----          ----          -----
Contains      Method        bool Contains(string propertyName)
Equals       Method        bool Equals(System.Object obj)
```

Die Pipeline II

- Select-Object schafft Abhilfe...

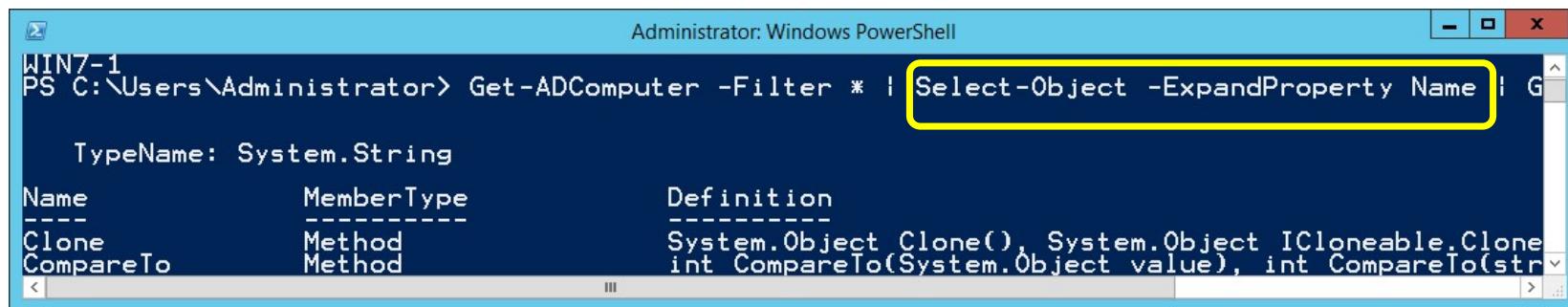


```
Administrator: Windows PowerShell
PS C:\Users\Administrator> Get-ADComputer -Filter * | Select-Object -Property Name | Get-Mem
```

The output shows the TypeName: Selected.Microsoft.ActiveDirectory.Management.ADComputer and its properties: Name, MemberType, Definition.

| Name | MemberType | Definition |
|-------------|------------|--------------------------------|
| Equals | Method | bool Equals(System.Object obj) |
| GetHashCode | Method | int GetHashCode() |

... aber nicht so ↑ , sondern so ↓:



```
Administrator: Windows PowerShell
WIN7-1
PS C:\Users\Administrator> Get-ADComputer -Filter * | Select-Object -ExpandProperty Name | G
```

The output shows the TypeName: System.String and its properties: Name, MemberType, Definition.

| Name | MemberType | Definition |
|-----------|------------|---|
| Clone | Method | System.Object Clone(), System.Object ICloneable.Clone |
| CompareTo | Method | int CompareTo(System.Object value), int CompareTo(str |

Die Pipeline II



```
Administrator: Windows PowerShell
PS C:\Users\Administrator> Get-Service -ComputerName (Get-ADComputer -Filter * | Select-Object -ExpandProperty Name)
Status    Name          DisplayName
-----  -----
Stopped  AdobeARMservice  Adobe Acrobat Update Service
Running   ADWS          Active Directory-Webdienste
Stopped  AeLookupSvc     Anwendungserfahrung
Stopped  AeLookupSvc     Anwendungserfahrung
Stopped  ALG           Gatewaydienst auf Anwendungsebene
Running   ALG           Gatewaydienst auf Anwendungsebene
```

`Get-Service -ComputerName (Get-ADComputer
Filter * | Select-Object -ExpandProperty Name)`

Oder kürzer:

`Get-Service -ComputerName (Get-ADComputer -Filter
*).Name`



```
Administrator: Windows PowerShell
PS C:\Users\Administrator> Get-Service -ComputerName (Get-ADComputer -Filter * | Select-Object -ExpandProperty Name,DisplayName,Status,MachineName)
Name          DisplayName          Status MachineName
-----  -----
AdobeARMservice  Adobe Acrobat Update Service  Stopped  SRV1
ADWS          Active Directory-Webdienste  Running   SRV1
AeLookupSvc     Anwendungserfahrung  Stopped  SRV1
AeLookupSvc     Anwendungserfahrung  Stopped  WIN8-1
ALG           Gatewaydienst auf Anwendun...  Stopped  WIN8-1
```

Übung: Pipeline II

- Erzeugen Sie mehrere AD-Benutzer in einem Rutsch mit Hilfe eines CSV-Files!
- Hinweis: Folgende Attribute sollte Ihr CSV mindestens enthalten (mehr sind auch gut):
 - GivenName
 - Surname
 - Name
 - SamAccountName
 - Enabled
 - Password



Mögliche Lösungen

- Import-Csv C:\Testfiles\MyNewADUsers.txt | New-ADUser



GivenName,Surname,Name,SamAccountName,Description,Department,EmployeeID,Path,Enabled,Password,PasswordNeverExpires
Hans,Dampf,Hans Dampf,hans.dampf,Hans Dampf,IT,189478,"OU=Benutzerkonten,DC=KURS,DC=INTERN",\$True,P@ssw0rd,\$True
Phil,O'Soph,Phil O'Soph,phil.osoph,Phil O'Soph,Marketing,08154711,"OU=Benutzerkonten,DC=KURS,DC=INTERN",\$True,P@ssw0rd,\$True

Hinweis

- Die AD-Benutzer werden alle deaktiviert angelegt. Warum?
- Grund: Die Passwörter werden nicht aus der CSV-Datei gelesen! **New-ADUser** erwartet das Passwort als „*SecureString*“
- Lösung siehe nächste Seite

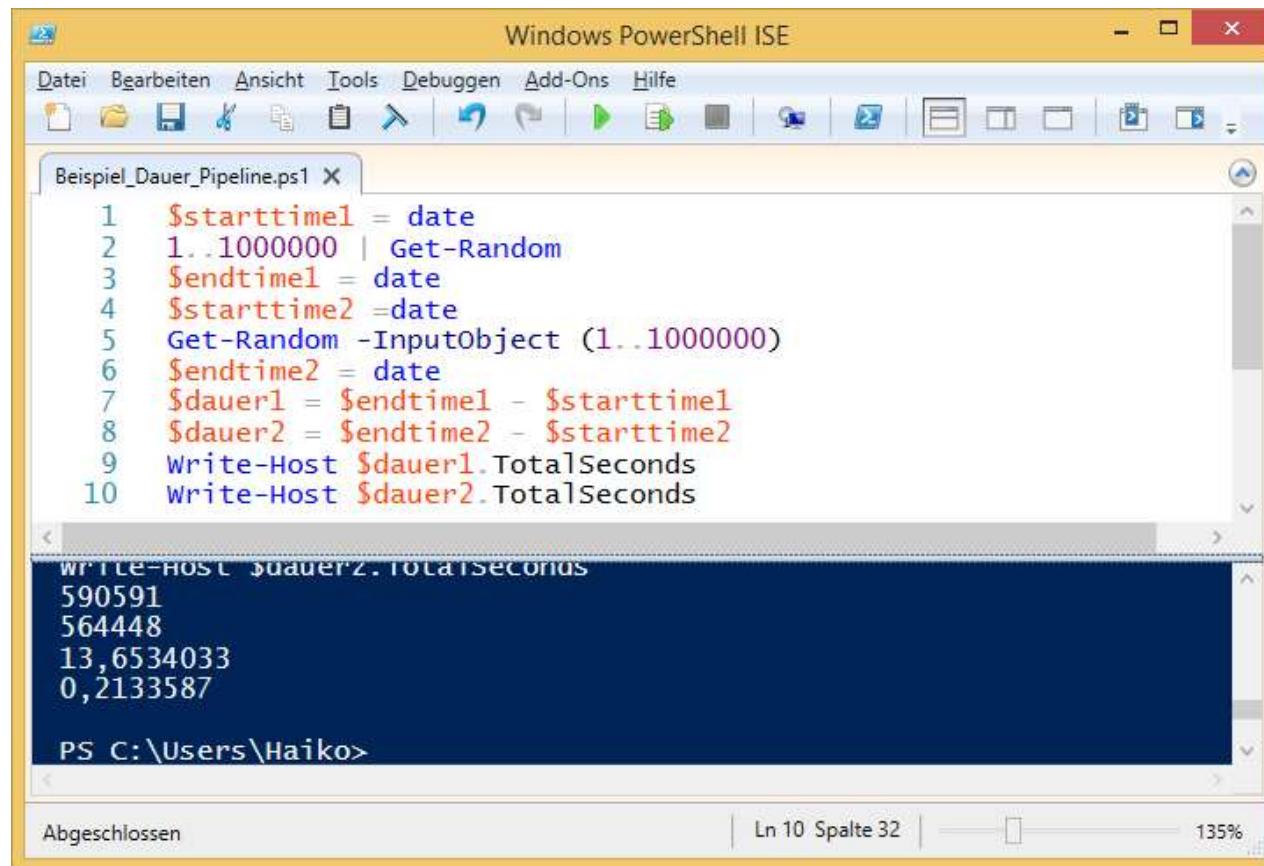
Hinweis

```
1 import-csv .\BulkAddADUsers.csv |  
2 % {  
3     New-ADUser -GivenName $_.GivenName  
4         -Surname $_.Surname  
5         -Name $_.Name  
6         -SamAccountName $_.SamAccountName  
7         -Description $_.Description  
8         -Department $_.Department  
9         -EmployeeID $_.EmployeeID  
10        -Path $_.Path  
11        -Enabled $True  
12        -AccountPassword  
13            (ConvertTo-SecureString $_.Password  
14                -AsPlainText -force)  
15        -PasswordNeverExpires $True  
16    }
```

Die Pipeline II

- Pipeline übergibt immer ein Objekt nach dem anderen
- Dies spart Speicher
- Nachteil: Langsam!
- Daher: Pipeline nur dort verwenden, wo es nötig oder sinnvoll ist!
- „Klassische“ Schleifen sind in der Regel schneller

Die Pipeline II



The screenshot shows the Windows PowerShell Integrated Scripting Environment (ISE) window. The title bar reads "Windows PowerShell ISE". The menu bar includes "Datei", "Bearbeiten", "Ansicht", "Tools", "Debuggen", "Add-Ons", and "Hilfe". The toolbar contains various icons for file operations like Open, Save, Copy, Paste, and Run. A tab titled "Beispiel_Dauer_Pipeline.ps1" is selected. The code editor displays the following PowerShell script:

```
1 $starttime1 = date
2 1..1000000 | Get-Random
3 $endtime1 = date
4 $starttime2 =date
5 Get-Random -InputObject (1..1000000)
6 $endtime2 = date
7 $dauer1 = $endtime1 - $starttime1
8 $dauer2 = $endtime2 - $starttime2
9 Write-Host $dauer1.TotalSeconds
10 Write-Host $dauer2.TotalSeconds
```

Below the code, the output pane shows the results of running the script:

```
PS C:\Users\Haiko>
write-host $dauer2.TotalSeconds
590591
564448
13,6534033
0,2133587
```

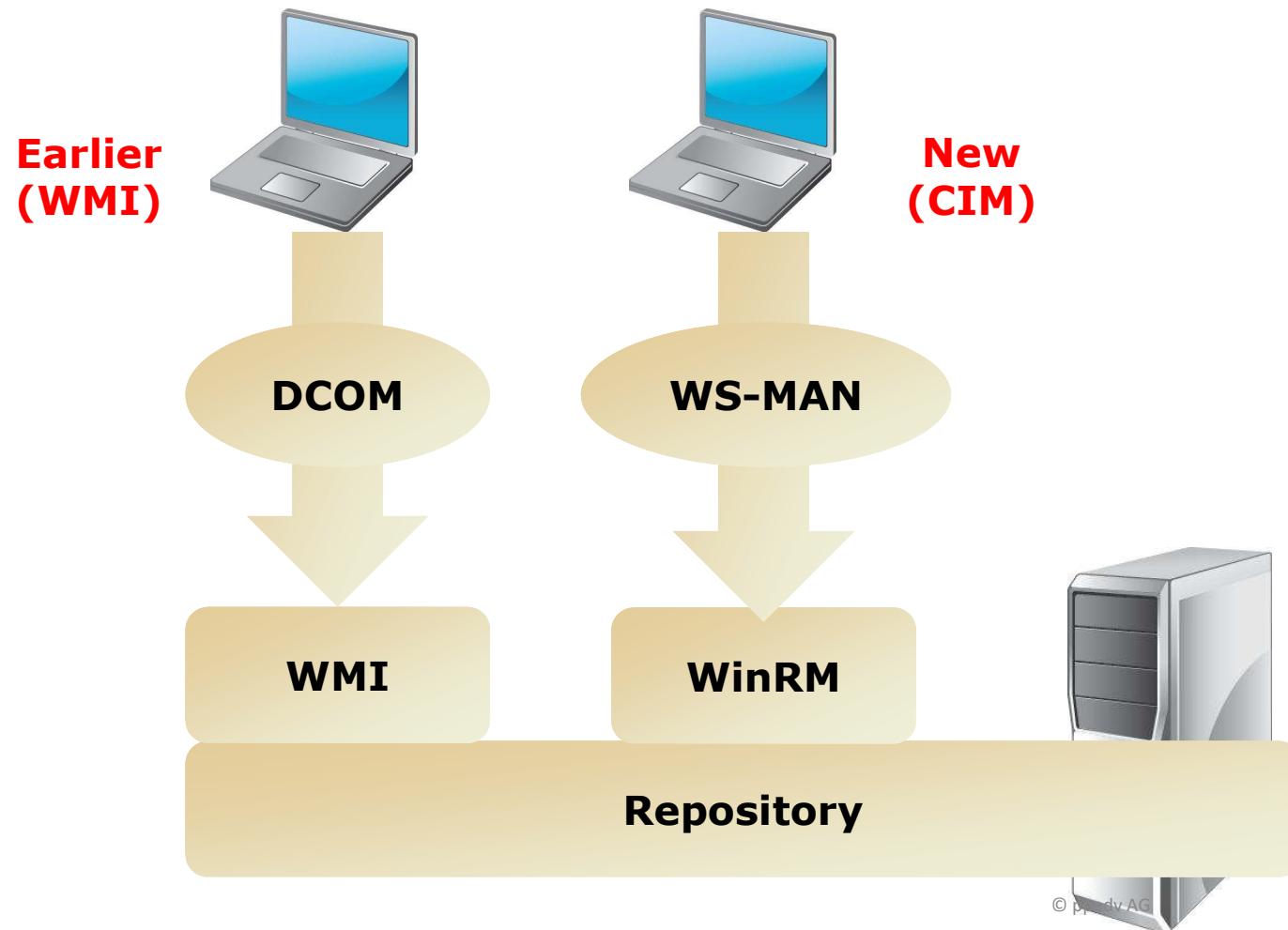
The status bar at the bottom indicates "Abgeschlossen" (Completed), "Ln 10 Spalte 32", and a zoom level of "135%".

Abfragen mit WMI und CIM

WMI / CIM

- WMI: Windows Management Instrumentation
- DCOM: Distributed Component Object Model
- CIM: Common Information Model
- WinRM: Windows Remote Management
- WS-MAN: Web Services für Management

WMI / CIM



WMI / CIM

| | CIM | WMI |
|---|-----|--------------------|
| Benötigt WMF 2.0 oder neuer | Ja* | WMF nicht benötigt |
| Benötigt aktiviertes Remoting | Ja* | Nein |
| Cross-platform Kompatibilität, i.e. Unterstützt Nicht-Windows Systeme | Ja | Nein |
| Benötigt einzelne Firewall Port Exception | Ja | Nein |
| Unterstützt session-based Verbindungen | Ja | Nein |
| Unterstützt ad-hoc-based Verbindungen | Ja | Ja |
| Unterstützt Microsoft Windows XP und neuer | Ja | Ja |
| Unterstützt Microsoft Windows Server 2003 und neuer | Ja | Ja |
| Unterstützt Microsoft Windows NT 4.0 und neuer | * | Ja |
| Benötigt Remote-Administration Firewall Exception | * | Ja |

WMI / CIM

- Grundlage beider Techniken ist das Repository
- Dieses ist in Namespaces und Klassen unterteilt
- Klassen repräsentieren die verwaltbaren Komponenten
- Instanzen („Instances“) sind die tatsächliche Abbildung einer Klasse
- Instanzen haben Eigenschaften („Properties“) und Methoden („Methods“)

WMI / CIM

- Sinnvollerweise schaut man sich die Dokumentation einer Klasse mittels Suchmaschine an
- Die Klassen in **root\CIMv2** sind gut dokumentiert
- Klassen anderer Namespaces sind z. T. schlecht dokumentiert

WMI / CIM

- Bsp. **Win32_BIOS** Klasse:
- [http://msdn.microsoft.com/en-us/library/aa394077\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/aa394077(v=vs.85).aspx)

Win32_BIOS class

20 out of 29 rated this helpful - Rate this topic

The **Win32_BIOS** WMI classrepresents the attributes of the computer system's basic input/output services (BIOS) that are installed on a computer.

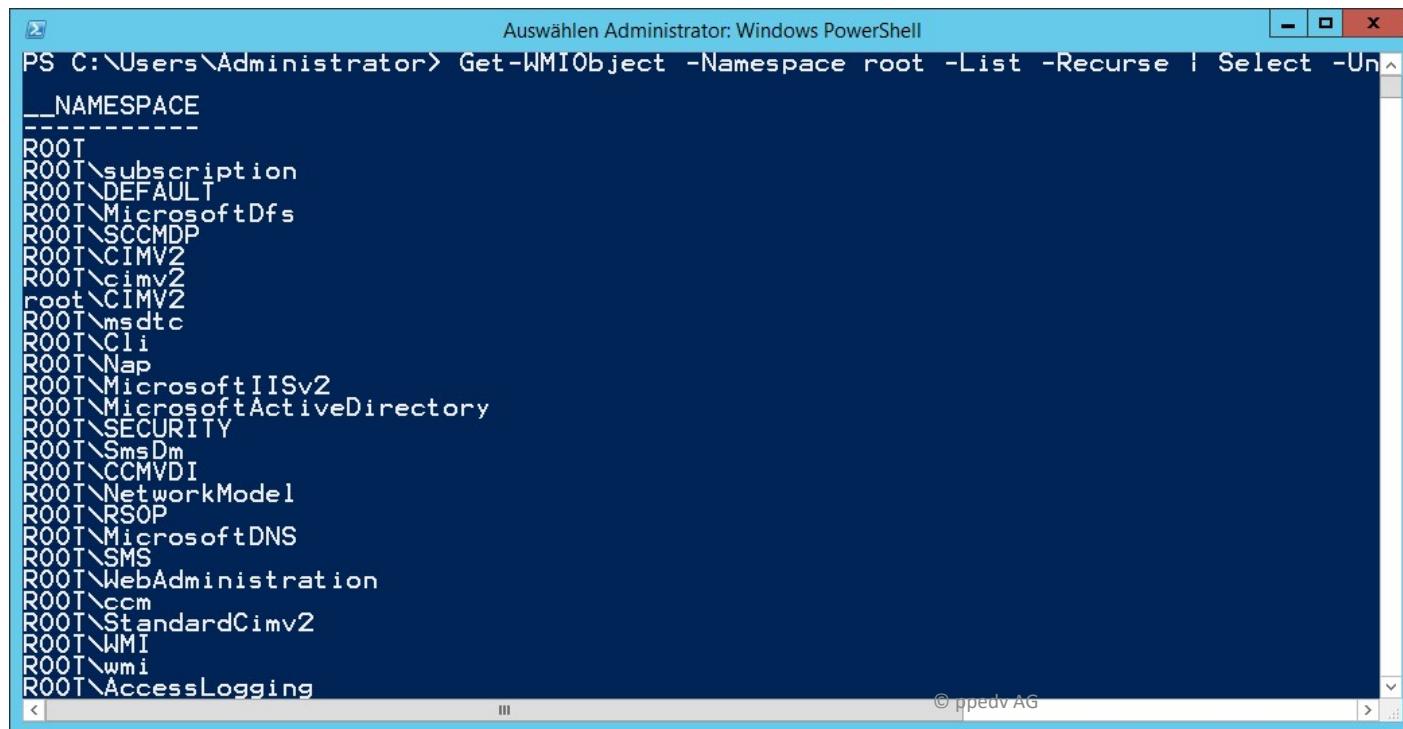
The following syntax is simplified from Managed Object Format (MOF) code and includes all of the inherited properties. Properties are listed in alphabetic order, not MOF order.

Syntax

```
[Provider("CIMWin32")]class Win32_BIOS : CIM_BIOSElement
{
    uint16  BiosCharacteristics[];
    string  BIOSVersion[];
    string  BuildNumber;
    string  Caption;
    string  CodeSet;
    string  CurrentLanguage;
    string  Description;
    string  IdentificationCode;
    uint16  InstallableLanguages;
    datetime InstallDate;
    string  LanguageEdition;
    String  ListOfLanguages[];
    string  Manufacturer;
    string  Name;
    string  OtherTargetOS;
    boolean PrimaryBIOS;
    datetime ReleaseDate;
    string  SerialNumber;
    string  SMBIOSBIOSVersion;
    uint16  SMBIOSMajorVersion;
```

WMI / CIM

- Vorhandene Namespaces können abgefragt werden:
- Get-WMIOBJECT -Namespace root -List -Recurse | Select -Unique __NAMESPACE



```
Auswählen Administrator: Windows PowerShell
PS C:\Users\Administrator> Get-WMIOBJECT -Namespace root -List -Recurse | Select -Unique __NAMESPACE
__NAMESPACE
-----
ROOT
ROOT\subscription
ROOT\DEFAULT
ROOT\MicrosoftDfs
ROOT\SCCMDP
ROOT\CIMV2
ROOT\cimv2
root\CIMV2
ROOT\msdtc
ROOT\Cl
ROOT\Nap
ROOT\MicrosoftIISv2
ROOT\MicrosoftActiveDirectory
ROOT\SECURITY
ROOT\SmsDm
ROOT\CCMVDI
ROOT\NetworkModel
ROOT\RSOP
ROOT\MicrosoftDNS
ROOT\SMS
ROOT\WebAdministration
ROOT\ccm
ROOT\StandardCimv2
ROOT\WMI
ROOT\wmi
ROOT\AccessLogging
```

WMI / CIM

- Klassen lassen sich alphabetisch anzeigen:

```
Get-WmiObject -Namespace root\cimv2 -List |  
Sort Name
```

```
Get-CimClass -Namespace root\CIMv2 |  
Sort CimClassName
```

- Raten ist (fast) die einzige Möglichkeit, die passende Klasse zu finden
- Alternativ: „WMI Explorer“ Tools

WMI / CIM

- Klassen, die mit `__` starten sind Systemklassen, können ignoriert werden:

| NameSpace: | R00T\SecurityCenter2 | |
|----------------------------|----------------------|--------------------------------------|
| Name | Methods | Properties |
| CIM_Indication | {} | {CorrelatedIndications, Indicati...} |
| CIM_ClassIndication | {} | {ClassDefinition, CorrelatedIndi...} |
| CIM_ClassDeletion | {} | {ClassDefinition, CorrelatedIndi...} |
| CIM_ClassCreation | {} | {ClassDefinition, CorrelatedIndi...} |
| CIM_ClassModification | {} | {ClassDefinition, CorrelatedIndi...} |
| CIM_InstIndication | {} | {CorrelatedIndications, Indicati...} |
| CIM_InstCreation | {} | {CorrelatedIndications, Indicati...} |
| CIM_InstModification | {} | {CorrelatedIndications, Indicati...} |
| CIM_InstDeletion | {} | {CorrelatedIndications, Indicati...} |
| _NotifyStatus | {} | {StatusCode} |
| _ExtendedStatus | {} | {Description, Operation, Paramet...} |
| CIM_Error | {} | {CIMStatusCode, CIMStatusCodeDes...} |
| MSFT_WmiError | {} | {CIMStatusCode, CIMStatusCodeDes...} |
| MSFT_ExtendedStatus | {} | {CIMStatusCode, CIMStatusCodeDes...} |
| _SecurityRelatedClass | {} | {} |
| Trustee | {} | {Domain, Name, SID, SidLength...} |
| NTLMUser9X | {} | {Authority, Flags, Mask, Name...} |
| ACE | {} | {AccessMask, AceFlags, AceType,...} |
| SecurityDescriptor | {} | {ControlFlags, DACL, Group, Own...} |
| PARAMETERS | {} | {} |
| SystemClass | {} | {} |
| ProviderRegistration | {} | {provider} |
| EventProviderRegistration | {} | {EventQueryList, provider} |
| ObjectProviderRegistration | {} | {InteractionType, provider, Quer...} |
| ClassProviderRegistration | {} | {CacheRefreshInterval, Interacti...} |

WMI / CIM

```
Windows PowerShell
PS C:\Users\Haiko> Get-CimClass -Namespace root\CIMv2 | Sort CimClassName -Descending

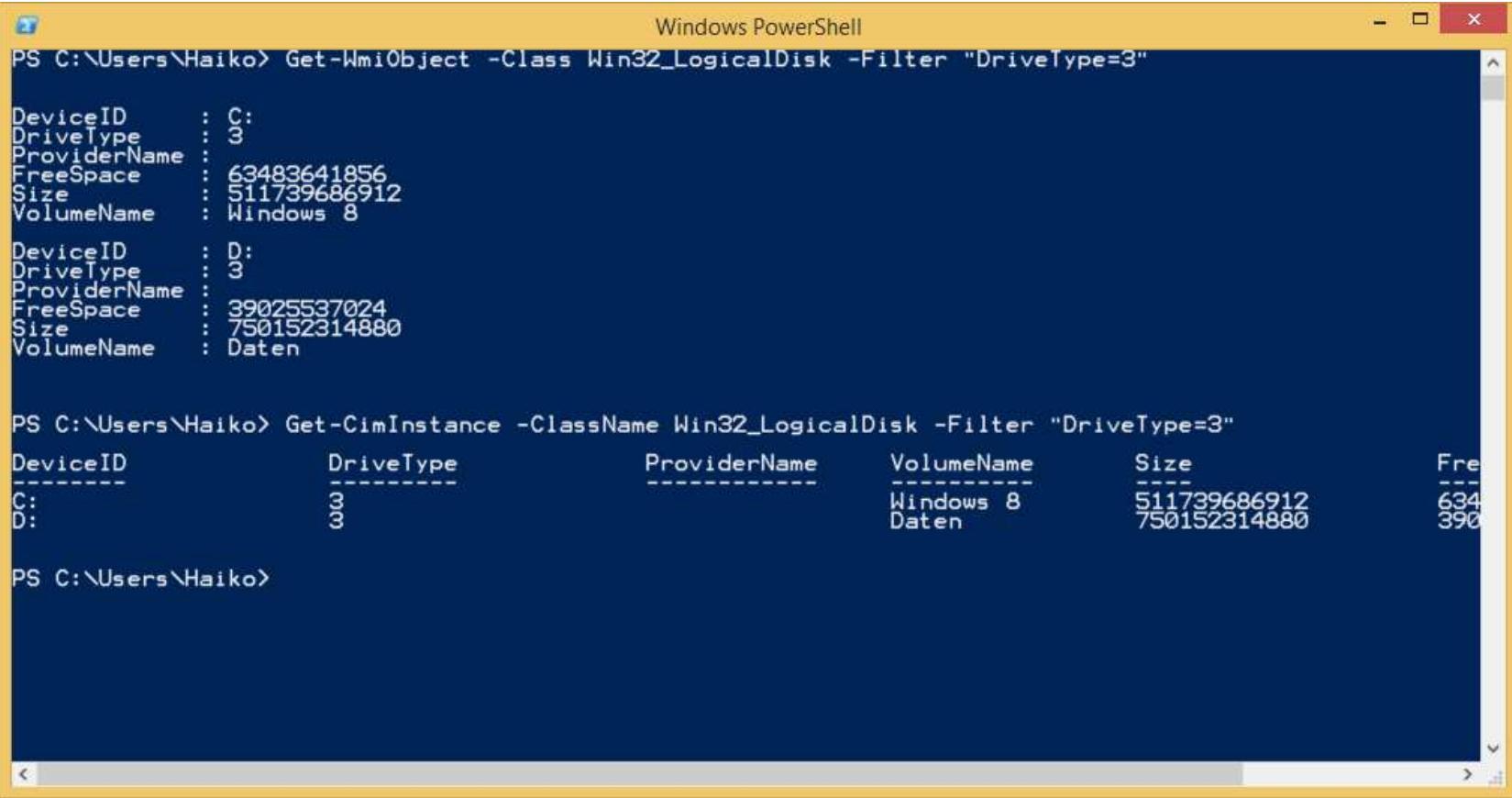
NameSpace: ROOT/CIMv2

CimClassName          CimClassMethods      CimClassProperties
-----              -----
Win32_WMISetting      {}                  {Caption, Description, SettingID}
Win32_WMIElementSetting {}                  {Element, Setting}
Win32_WinSAT           {}                  {CPUScore, D3DScore, DiskScore, Account, DiskSpaceUsed, Limit, Element, Setting}
Win32_VolumeUserQuota  {}                  {Element, Setting}
Win32_VolumeQuotaSetting {}                {Element, Setting}
Win32_VolumeQuota     {}                  {Element, Setting}
Win32_VolumeChangeEvent {}                {SECURITY_DESCRIPTOR, TIME_CREATE, SetPowerState, R...}
Win32_Volume           {}                  {Caption, Description, InstallDate, SetPowerState, R...}
Win32_VoltageProbe    {}                  {Caption, Description, InstallDate, Element, Setting}
Win32_VideoSettings   {}                  {SetPowerState, R...}
Win32_VideoController {}                {Caption, Description, InstallDate, SetPowerState, R...}
Win32_VideoConfiguration {}              {Caption, Description, SettingID, Day, DayOfWeek, Hour, Milliseconds, FolderRedirection, OfflineFiles}
Win32_UTCTime          {}                  {Caption, Description, InstallDate, SetPowerState, R...}
Win32_UserStateConfigurationCont... {}        {ChangeOwner}
Win32_UserProfile      {}                  {AppDataRoaming, Contacts, Desktop, GroupComponent, PartComponent}
Win32_UserInDomain     {}                  {Element, Setting}
Win32_UserDesktop      {}                  {Caption, Description, InstallDate, SetPowerState, R...}
Win32_UserAccount      {Rename}            {Caption, Description, InstallDate, SetPowerState, R...}
Win32_USBHub           {}                  {Caption, Description, InstallDate, Antecedent, Dependent, Negotiated}
Win32_USBControllerDevice {}                {Caption, Description, InstallDate, SetPowerState, R...}
Win32_USBController    {}                  {ActionID, Caption, Description, Domain, Name, SID, SidLength...}
Win32_TypeLibraryAction {}                {Invoke}
Win32_Trustee          {}                  {PrivilegeCount, Privileges}
Win32_TokenPrivileges {}                {GroupCount, Groups}
Win32_TokenGroups      {}                  {Caption, Description, SettingID, SECURITY_DESCRIPTOR, TIME_CREATE}
Win32_TimeZone          {}                {SECURITY_DESCRIPTOR, TIME_CREATE}
Win32_ThreadTrace      {}                {SECURITY_DESCRIPTOR, TIME_CREATE}
Win32_ThreadStopTrace  {}                {SECURITY_DESCRIPTOR, TIME_CREATE}
Win32_ThreadStartTrace {}                {Caption, Description, InstallDate, SetPowerState, R...}
Win32_Thread           {}                {Caption, Description, InstallDate}
```

WMI / CIM

- Abfragen enthalten immer den Namen der Klasse
- **-Namespace**, falls Klasse nicht in **root\CMv2** ist
- **-Filter** um die Instanzen, die zurückgegeben werden einzuschränken
- Filter-Operatoren sind nicht die üblichen in PS!

WMI / CIM



Windows PowerShell

```
PS C:\Users\Haiko> Get-WmiObject -Class Win32_LogicalDisk -Filter "DriveType=3"

DeviceID      : C:
DriveType     : 3
ProviderName  :
FreeSpace     : 63483641856
Size          : 511739686912
VolumeName    : Windows 8

DeviceID      : D:
DriveType     : 3
ProviderName  :
FreeSpace     : 39025537024
Size          : 750152314880
VolumeName    : Daten

PS C:\Users\Haiko> Get-CimInstance -ClassName Win32_LogicalDisk -Filter "DriveType=3"

DeviceID      DriveType      ProviderName   VolumeName   Size   Free
-----      -----      -----
C:           3             Windows 8       Windows 8   511739686912 634
D:           3             Daten          Daten       750152314880 390

PS C:\Users\Haiko>
```

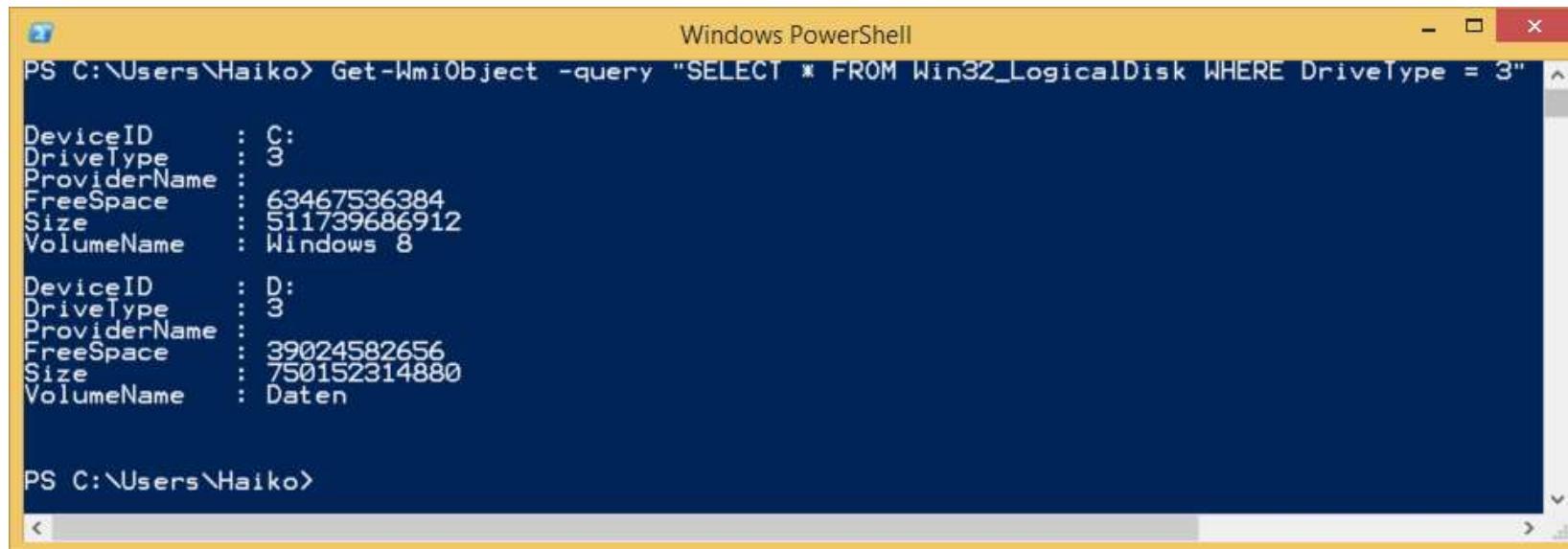
WMI / CIM

```
Get-WmiObject -Class  
Win32_LogicalDisk  
-Filter "DriveType=3"
```

```
Get-CimInstance -ClassName  
Win32_LogicalDisk  
-Filter "DriveType=3"
```

WMI / CIM

- Über WMI/CIM lassen sich auch WQL-Abfragen stellen:



The screenshot shows a Windows PowerShell window titled "Windows PowerShell". The command run is "Get-WmiObject -query "SELECT * FROM Win32_LogicalDisk WHERE DriveType = 3"". The output displays two logical disk objects (C: and D) with their properties:

| Property | Value |
|--------------|--------------|
| DeviceID | C: |
| DriveType | 3 |
| ProviderName | |
| FreeSpace | 63467536384 |
| Size | 511739686912 |
| VolumeName | Windows 8 |
| DeviceID | D: |
| DriveType | 3 |
| ProviderName | |
| FreeSpace | 39024582656 |
| Size | 750152314880 |
| VolumeName | Daten |

```
Get-WmiObject -query "SELECT * FROM Win32_LogicalDisk WHERE DriveType = 3"
```

```
Get-CimInstance -query "SELECT * FROM Win32_LogicalDisk WHERE DriveType = 3"
```

WMI / CIM

- Get-WmiObject -Class Win32_Service
- Get-CimInstance -ClassName Win32_Process
- Get-CimInstance -ClassName Win32_LogicalDisk - Filter "DriveType=3,"
- Get-CimInstance -Query "SELECT * FROM Win32_NetworkAdapter"



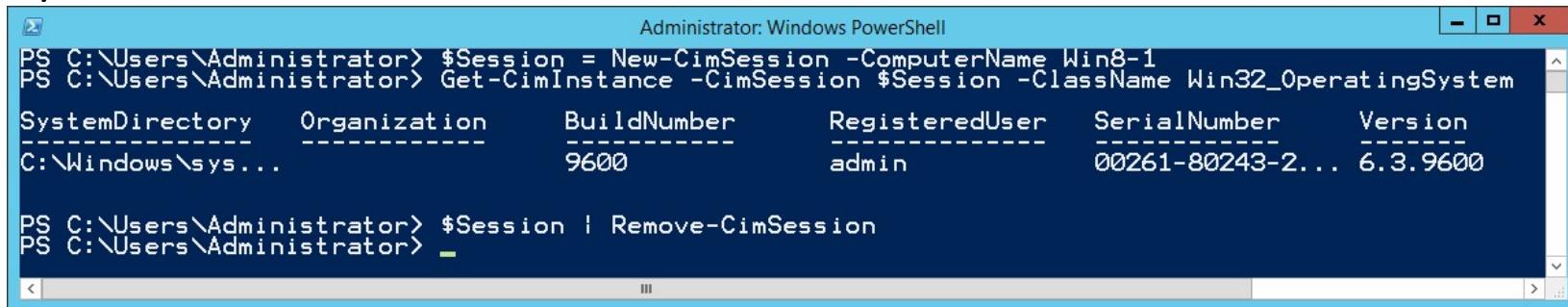
WMI / CIM

- Remote-Abfrage:
 - -ComputerName zur Angabe des Zielcomputers
 - -Credential für alternative Anmeldedaten
- Get-WmiObject
ComputerName WIN8-1
 - Credential PPEDV\Administrator
 - Class Win32_BIOS
- WMI nutzt DCOM, CIM nutzt WinRM für ad-hoc Verbindungen

WMI / CIM

- CIMSessions
 - wiederverwendbare,
 - beständige,
 - authentifizierte Verbindungen zu einem Remote-System
- Erzeugt und in einer Variable gespeichert
- Anstelle von **-ComputerName** wird dann **-CimSession** zur Übergabe verwendet
- CIMSessions haben einen Timeout
- Können auch manuell geschlossen werden

WIM / CIM



Administrator: Windows PowerShell

```
PS C:\Users\Administrator> $Session = New-CimSession -ComputerName Win8-1
PS C:\Users\Administrator> Get-CimInstance -CimSession $Session -ClassName Win32_OperatingSystem
SystemDirectory Organization BuildNumber RegisteredUser SerialNumber Version
----- ----- ----- -----
C:\Windows\sys... ... 9600 admin 00261-80243-2... 6.3.9600

PS C:\Users\Administrator> $Session | Remove-CimSession
PS C:\Users\Administrator>
```

- \$Session = New-CimSession
ComputerName Win8-1
- Get-CimInstance -CimSession \$Session -ClassName Win32_OperatingSystem



© ppedv AG

WMI / CIM

- Über WMI und CIM lassen sich auch Änderungen durchführen
- Dafür sind die Methoden der Klassen zuständig
- Mit **Get-Member** lassen sich die Methoden auflisten
- Zusätzlich ist eine Dokumentation nötig

WMI / CIM

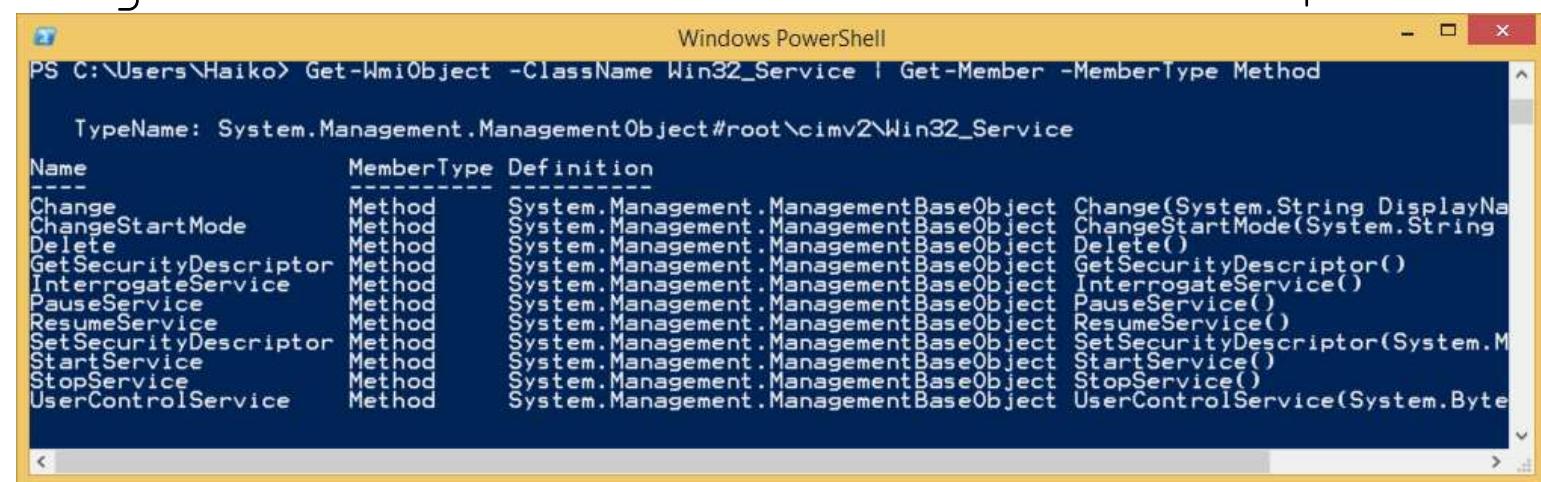
Methods

The **Win32_Service** class has these methods.

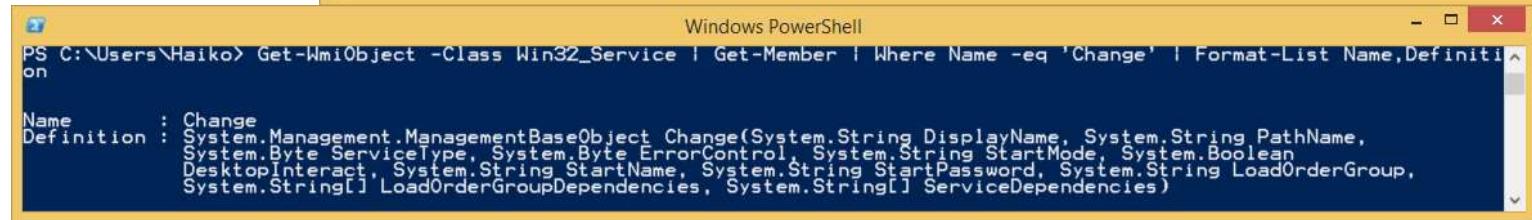
| Method | Description |
|------------------------------|--|
| Change | Modifies a service. |
| ChangeStartMode | Modifies the start mode of a service. |
| Create | Creates a new service. |
| Delete | Deletes an existing service. |
| GetSecurityDescriptor | Returns the security descriptor that controls access to the service. This method is available starting with Windows Vista. Windows Server 2003, Windows XP, Windows 2000, and Windows NT 4.0: This method is not available. |
| InterrogateService | Requests that a service update its state to the service manager. |
| PauseService | Attempts to place a service in the paused state. |
| ResumeService | Attempts to place a service in the resumed state. |
| SetSecurityDescriptor | Writes an updated version of the security descriptor that controls access to the service. This method is available starting with Windows Vista. Windows Server 2003, Windows XP, Windows 2000, and Windows NT 4.0: This method is not available. |
| StartService | Attempts to place a service into the startup state. |
| StopService | Places a service in the stopped state. |
| UserControlService | Attempts to send a user-defined control code to a service. |

WMI / CIM

- Methoden finden:
- Get-WmiObject -ClassName Win32_Service | Get-Member



```
TypeName: System.Management.ManagementObject#root\cimv2\Win32_Service
Name          MemberType Definition
----          -----
Change        Method   System.Management.ManagementBaseObject Change(System.String DisplayName, System.String PathName, System.Byte ServiceType, System.Byte ErrorControl, System.String StartMode, System.Boolean DesktopInteract, System.String StartName, System.String StartPassword, System.String LoadOrderGroup, System.String[] LoadOrderGroupDependencies, System.String[] ServiceDependencies)
```

```
Name      : Change
Definition : System.Management.ManagementBaseObject Change(System.String DisplayName, System.String PathName, System.Byte ServiceType, System.Byte ErrorControl, System.String StartMode, System.Boolean DesktopInteract, System.String StartName, System.String StartPassword, System.String LoadOrderGroup, System.String[] LoadOrderGroupDependencies, System.String[] ServiceDependencies)
```



WMI / CIM

- Zum Ausführen der Methoden eignen sich mehrere Commandlets:
 - **Invoke-WmiMethod**
 - **Invoke-CimMethod**
 - **ForEach-Object**
- Das Rückgabe-Objekt enthält für gewöhnlich ein **ReturnValue**, 0 heisst in der Regel „success“

WMI / CIM

- Methoden ausführen:
- Invoke-CimMethod -ComputerName WIN8-1 -
ClassName Win32_OperatingSystem -MethodName
Reboot
- Get-WmiObject -Class Win32_Process -Filter
"Name='mspaint.exe'" | Invoke-WmiMethod -Name
Terminate



© ppedv AG

Übung: WMI/CIM

1. Lassen Sie sich die Klassen aus root\CIMv2 ausgeben
2. Schauen Sie sich einige Klassen näher an
3. Fragen Sie ALLE Werte aus Win32_OperatingSystem ab
4. Bauen Sie eine CIM-Session zu Member1 auf und Fragen Sie die Werte der logischen Festplatte von Member1 ab



Übung: WMI/CIM

1. Lassen Sie sich die Klassen aus root\CIMv2 ausgeben
2. Schauen Sie sich einige Klassen näher an
3. Fragen Sie ALLE Werte aus Win32_OperatingSystem ab



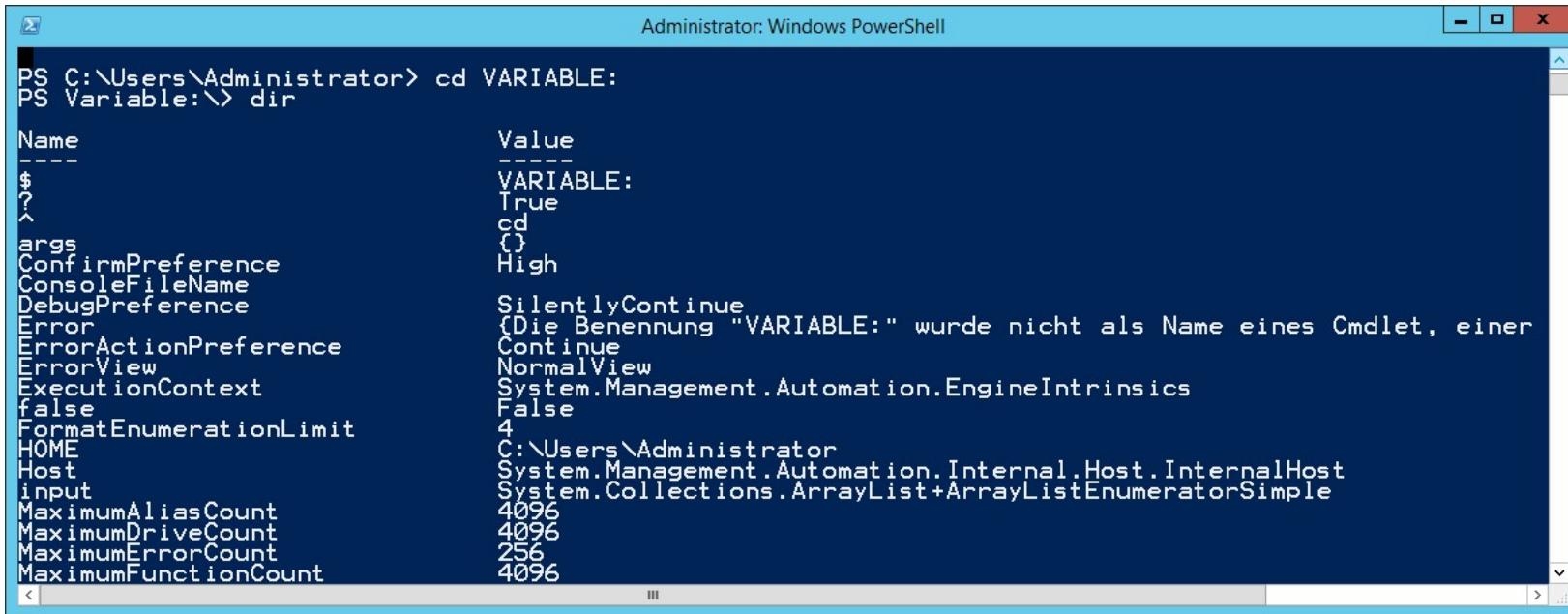
Mögliche Lösungen

1. Get-WmiObject -Namespace root\CMV2 -List
2. Get-WmiObject -Class KLASSE bzw. Get-CIMInstance -Classname KLASSE
3. Get-CimInstance Win32_OperatingSystem | fl *
4. \$Session = New-CimSession -ComputerName MEMBER1; Get-CimInstance -CimSession \$Session -ClassName Win32_LogicalDisk

Variablen II

Variablen II

- Variablen sind in einem eigenen Laufwerk **VARIABLE:** gespeichert:

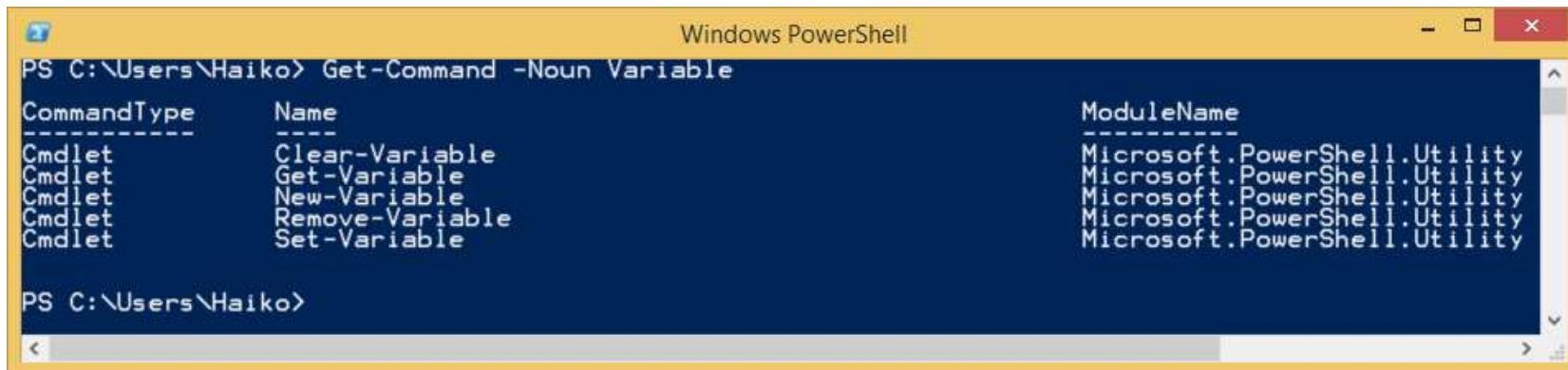


The screenshot shows a Windows PowerShell window titled "Administrator: Windows PowerShell". The command entered is "PS C:\Users\Administrator> cd VARIABLE:" followed by "PS Variable:> dir". The output displays a table of variables and their values:

| Name | Value |
|------------------------|--|
| --- | ---- |
| \$ | VARIABLE: |
| ? | True |
| ^ | cd |
| args | {} |
| ConfirmPreference | High |
| ConsoleFileName | SilentlyContinue |
| DebugPreference | {Die Benennung "VARIABLE:" wurde nicht als Name eines Cmdlet, einer Continue |
| Error | NormalView |
| ErrorActionPreference | System.Management.Automation.EngineIntrinsics |
| ErrorView | False |
| ExecutionContext | 4 |
| false | C:\Users\Administrator |
| FormatEnumerationLimit | System.Management.Automation.Internal.Host.InternalHost |
| HOME | System.Collections.ArrayList+ArrayListEnumeratorSimple |
| Host | 4096 |
| input | 4096 |
| MaximumAliasCount | 256 |
| MaximumDriveCount | 4096 |
| MaximumErrorCount | |
| MaximumFunctionCount | |

Variablen II

- Wenn man mehr als ein Objekt in einer Variable abspeichert entsteht ein **Array**
- Es existieren einige Commandlets, mit denen sich Variablen verwalten lassen:



The screenshot shows a Windows PowerShell window titled "Windows PowerShell". The command `Get-Command -Noun Variable` is run, and the output displays five cmdlets: Clear-Variable, Get-Variable, New-Variable, Remove-Variable, and Set-Variable. Each cmdlet is listed under the "Name" column, which is aligned with the "Name" header. The "CommandType" column shows all entries as "Cmdlet". The "ModuleName" column shows all entries as "Microsoft.PowerShell.Utility".

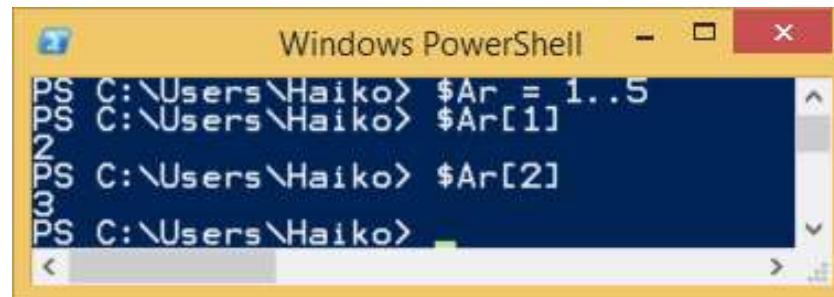
| CommandType | Name | ModuleName |
|-------------|-----------------|------------------------------|
| Cmdlet | Clear-Variable | Microsoft.PowerShell.Utility |
| Cmdlet | Get-Variable | Microsoft.PowerShell.Utility |
| Cmdlet | New-Variable | Microsoft.PowerShell.Utility |
| Cmdlet | Remove-Variable | Microsoft.PowerShell.Utility |
| Cmdlet | Set-Variable | Microsoft.PowerShell.Utility |

Variablen II

- Variablen existieren nur innerhalb bestimmter Bereiche (sog. „**lokale Variablen**“)
- Optional kann man **globale Variablen** definieren
- Variablennamen sollten nur aus **Buchstaben, Zahlen und Unterstrichen** bestehen
- Weitere Zeichen wären möglich, aber nicht empfohlen
- \$ ist kein Teil des Variablennamens sondern nur für den Zugriff

Variablen II

- Der Zugriff auf einzelne Objekte in einem Array geschieht etwa in der Form:

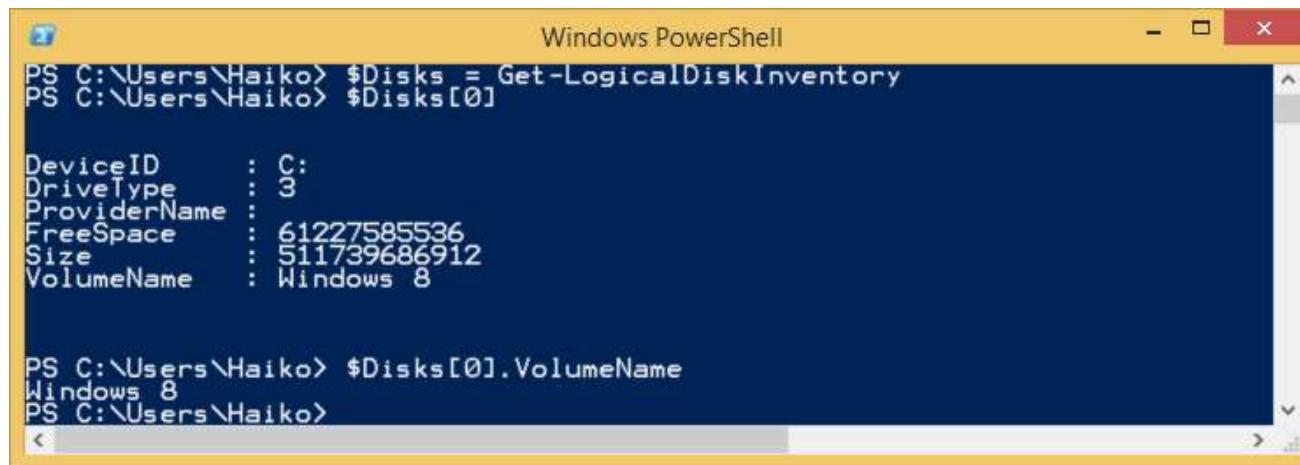


A screenshot of a Windows PowerShell window titled "Windows PowerShell". The command PS C:\Users\Haiko> \$Ar = 1..5 creates an array \$Ar with elements from 1 to 5. The command PS C:\Users\Haiko> \$Ar[1] outputs the value 2. The command PS C:\Users\Haiko> \$Ar[2] outputs the value 3. The command PS C:\Users\Haiko> is shown at the bottom.

- Zählung beginnt bei 0

Variablen II

- Mit Hilfe des Punktes (.) kann man Bestandteile eines Objektes direkt adressieren:



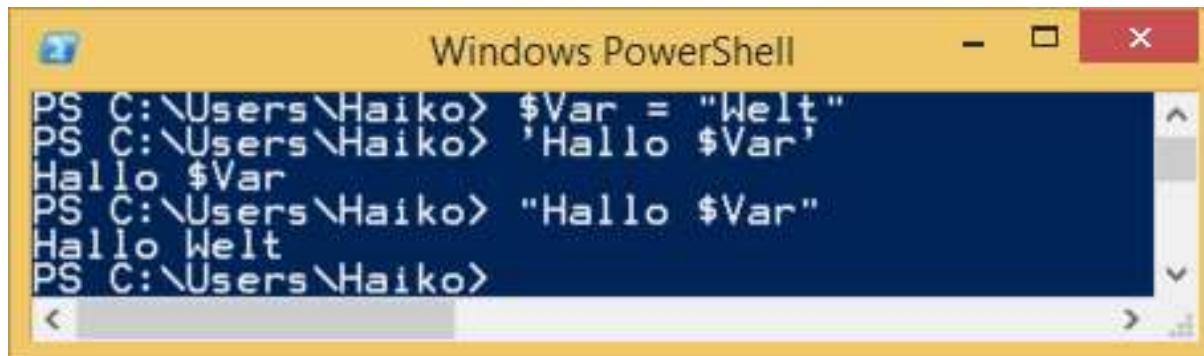
```
Windows PowerShell
PS C:\Users\Haiko> $Disks = Get-LogicalDiskInventory
PS C:\Users\Haiko> $Disks[0]

DeviceID      : C:
DriveType     : 3
ProviderName  :
FreeSpace     : 61227585536
Size          : 511739686912
VolumeName    : Windows 8

PS C:\Users\Haiko> $Disks[0].VolumeName
Windows 8
PS C:\Users\Haiko>
```

Variablen II

- Variablen in einfach Anführungszeichen:
 - Variablename als Wort
- Variablen in doppelten Anführungszeichen:
 - Wert der Variable

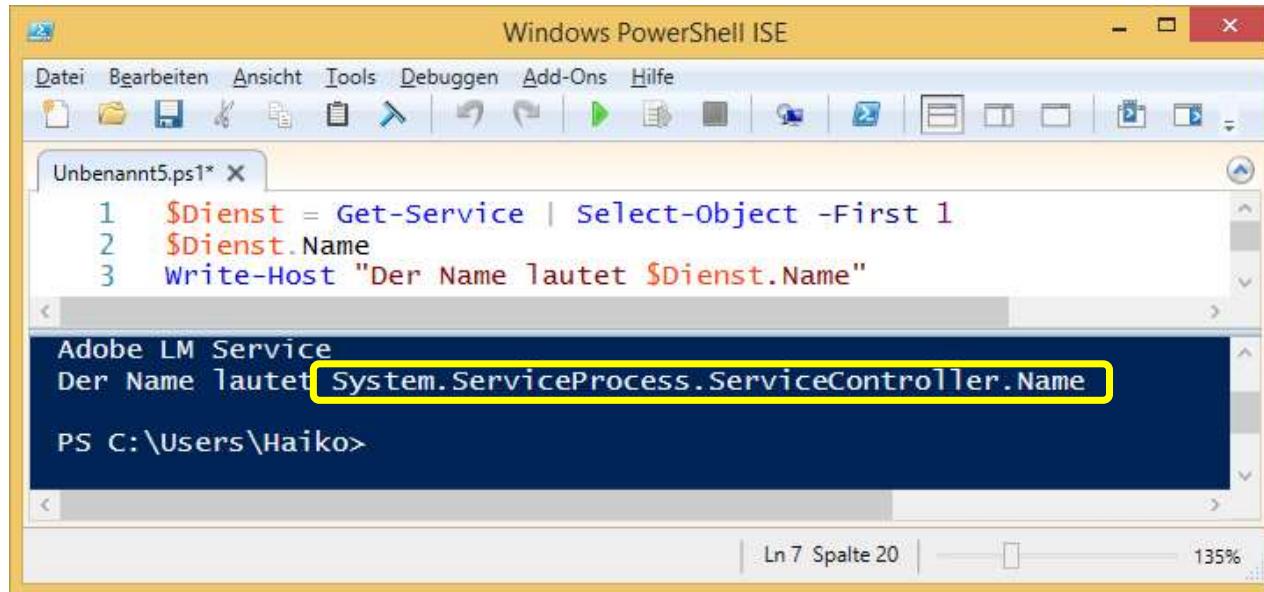


The screenshot shows a Windows PowerShell window titled "Windows PowerShell". The command PS C:\Users\Haiko> \$Var = "Welt" is entered and executed. Then, the command PS C:\Users\Haiko> 'Hallo \$Var' is entered, which outputs Hallo \$Var. Finally, the command PS C:\Users\Haiko> "Hallo \$Var" is entered, which outputs Hallo Welt.

```
PS C:\Users\Haiko> $Var = "Welt"
PS C:\Users\Haiko> 'Hallo $Var'
Hallo $Var
PS C:\Users\Haiko> "Hallo $Var"
Hallo Welt
PS C:\Users\Haiko>
```

Variablen II

- Will man innerhalb einer Ausgabe auf Member einer Variable zugreifen:



The screenshot shows the Windows PowerShell ISE interface. In the top-left corner, there's an orange ribbon tab with the number '5' and a star icon. The main window has a title bar 'Windows PowerShell ISE'. The menu bar includes 'Datei', 'Bearbeiten', 'Ansicht', 'Tools', 'Debuggen', 'Add-Ons', and 'Hilfe'. Below the menu is a toolbar with various icons. A code editor window titled 'Unbenannt5.ps1*' contains the following PowerShell script:

```
1 $Dienst = Get-Service | Select-Object -First 1
2 $Dienst.Name
3 Write-Host "Der Name lautet $Dienst.Name"
```

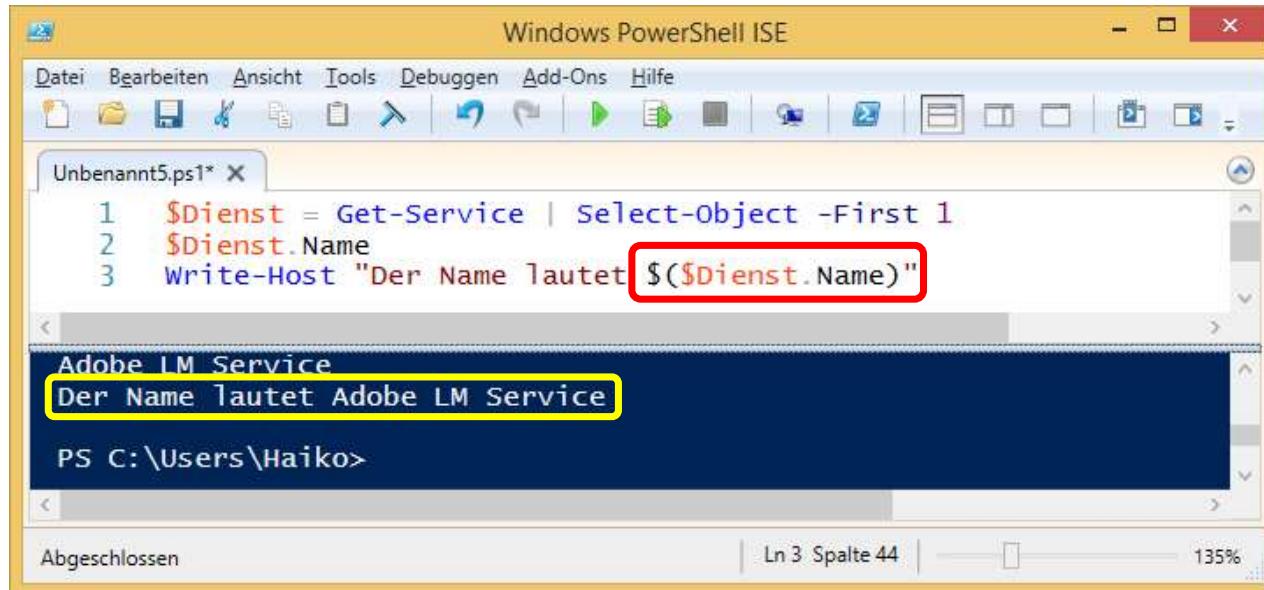
When the script is run, the output window shows the result of the command:

```
Adobe LM Service
Der Name lautet System.ServiceProcess.ServiceController.Name
```

The output line 'Der Name lautet System.ServiceProcess.ServiceController.Name' is highlighted with a yellow box.

Variablen II

- Hier muss man einen kleinen „Trick“ anwenden:



The screenshot shows the Windows PowerShell ISE interface. A script named "Unbenannt5.ps1" is open in the editor. The code contains three lines of PowerShell:

```
1 $Dienst = Get-Service | Select-Object -First 1
2 $Dienst.Name
3 Write-Host "Der Name lautet $($Dienst.Name)"
```

The line "Der Name lautet \$(\$Dienst.Name)" is highlighted with a red rectangle. The output window below shows the result of the script execution:

```
Adobe LM Service
Der Name lautet Adobe LM Service
```

The output "Der Name lautet Adobe LM Service" is highlighted with a yellow rectangle.

Übung: Variablen II

1. Legen Sie ein Array mit den Zahlen von 1 bis 5 an
2. Lassen Sie sich die Zahl an der 5. Stelle ausgeben
3. Verändern Sie diese Zahl



Mögliche Lösungen

1. Mehrere Varianten:
 1. \$e = @(); \$e += 1; \$e += 2; \$e += 3; ...
 2. [int[]] \$e = 1,2,3,4,5
 3. \$e = 1,2,3,4,5
 4. \$e = 1..5
2. \$e[4] (Array beginnt bei 0 zu zählen!)
3. \$e[4] = 99

Demonstration: Using Variables

In this demonstration, you will see several ways to use variables

- Naming variables
- Using variables
- Using double quotation marks
- Using subexpressions

Skriptsicherheit

Skripting

- Skriptsicherheit soll:
 - einen uninformierten Benutzer
 - welcher unbeabsichtigt
 - versucht, ein nicht-vertrauenswürdiges Skript auszuführen
 - verlangsamten.
- Sie kann nicht verhindern, dass ein informierter Benutzer gewollt ein Skript ausführt
- Sie ersetzt auch keinen Anti-Malware-Schutz

Skripting

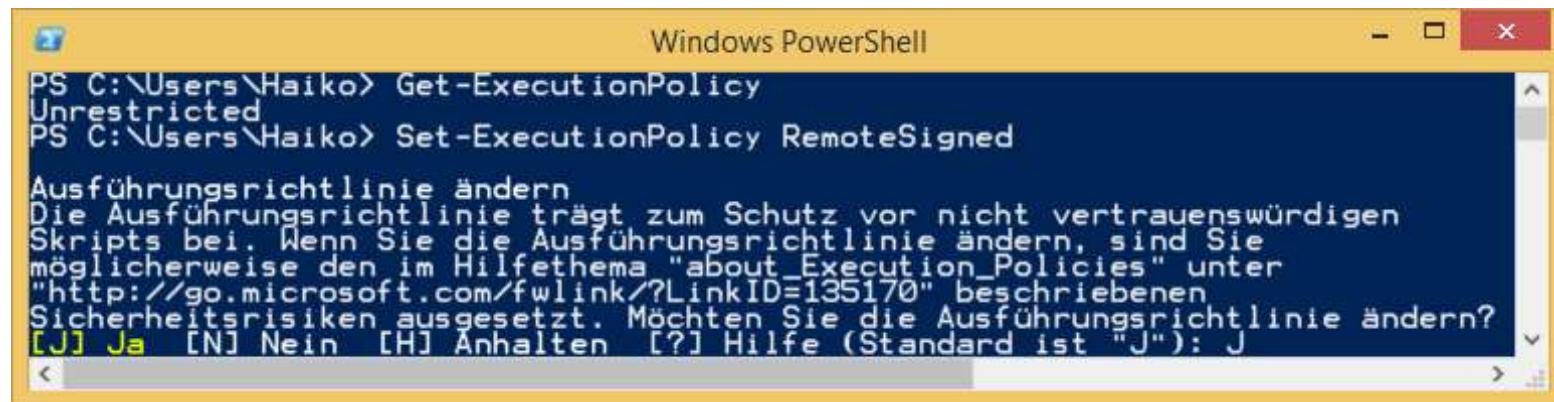
- Fünf „Execution Policy“ Einstellungen:
 - Restricted (Standard)
 - AllSigned
 - RemoteSigned
 - Unrestricted
 - Bypass
- Drei Wege, die Einstellung zu ändern:
 - **Set-ExecutionPolicy**
 - GPO
 - **-ExecutionPolicy** Paramater an **PowerShell.exe**

Skripting

| WERT | AUSWIRKUNG |
|---------------------|--|
| Restricted | Es werden keine Konfigurationsdateien geladen und keine Scripts ausgeführt (Standard) |
| AllSigned | Signierte Scripts und Konfigurationsdateien von einem vertrauenswürdigen Herausgeber werden ausgeführt. Auch lokal erstellte Scripts müssen signiert sein. |
| RemoteSigned | Aus dem Internet heruntergeladenen Scripts und Konfigurationsdateien müssen von einem vertrauenswürdigen Herausgeber signiert sein. |
| Unrestricted | Alle Konfigurationsdateien und alle Scripts werden ausgeführt. Bei nicht signierten Scripts aus dem Internet muss man jede Ausführung am Prompt bestätigen |
| Bypass | Keinerlei Blockade, keine Warnungen oder Prompts. |
| Undefined | Entfernt die gerade zugewiesene Richtlinie (nur für lokal zugewiesene Richtlinien, nicht für GPO-applizierte) |

Skripting

- Demo: Abfragen und Setzen der ExecutionPolicy



A screenshot of a Windows PowerShell window titled "Windows PowerShell". The command "Get-ExecutionPolicy" is run, returning the value "Unrestricted". Then, the command "Set-ExecutionPolicy RemoteSigned" is run. A confirmation dialog box appears, asking if the user wants to change the execution policy. The message in the box states: "Ausführungsrichtlinie ändern. Die Ausführungsrichtlinie trägt zum Schutz vor nicht vertrauenswürdigen Skripts bei. Wenn Sie die Ausführungsrichtlinie ändern, sind Sie möglicherweise den im Hilfethema "about_Execution_Policies" unter "http://go.microsoft.com/fwlink/?LinkId=135170" beschriebenen Sicherheitsrisiken ausgesetzt. Möchten Sie die Ausführungsrichtlinie ändern? [J] Ja [N] Nein [H] Anhalten [?] Hilfe (Standard ist "J"): J". The user has selected "Ja" (Yes).



Übung: Skriptsicherheit

1. Fragen Sie die Execution Policy von DC1 und Member1 ab
2. Setzen Sie alle Execution Policies auf „Unrestricted“

| | |
|---------------------|--------------------|
| DC.kurs.intern | KURS\Administrator |
| Member.1kurs.intern | KURS\Administrator |
| Host | Administrator |



Mögliche Lösungen

1. Get-ExecutionPolicy
2. Set-ExecutionPolicy Unrestricted

5
★

TRAINING



Skripting

Skripting

- Zu Beginn: Einfacher Aufruf mit spezifischen Parametern
- Testen, ob Syntax und Ausgabe passen

```
Get-EventLog -LogName Security -ComputerName  
localhost |  
Where EventID -eq 4624 |  
Select -First 50
```

Skripting

- Dann: Werte identifizieren, die sich beim nächsten Einsatz evtl. ändern könnten:

```
Get-EventLog -LogName Security -ComputerName  
localhost |  
Where EventID -eq 4624 |  
Select -First 50
```

Skripting

- Parameter einführen:

```
[CmdletBinding() ]  
Param(  
    [Parameter(Mandatory=$True) ]  
    [string] $ComputerName,  
    [int] $EventID = 4624  
)  
Get-EventLog -LogName Security -ComputerName $ComputerName |  
Where EventID -eq $EventID |  
Select -First 50
```



Standard-Wert; kann geändert werden

Skripting

- Bei Aufruf über ISE: Nur Mandatory-Parameter werden abgefragt
- Um restliche Parameter anzugeben / angeben zu können: Aufruf des Skriptes mit Pfad und Parametern

Skripting

- Demo:
 - Direktes Auslösen des Codes
 - Aufruf als Skript mit Parametern

```
[CmdletBinding() ]  
Param(  
    [Parameter(Mandatory=$True) ]  
    [string]$ComputerName,  
  
    [int]$EventID = 4624  
)  
Get-EventLog -LogName Security -ComputerName $ComputerName |  
Where EventID -eq $EventID | Select -First 50
```

Skripting

- Skripte sollten vor dem Test immer gespeichert werden
- ISE speichert danach selbstständig vor dem Ausführen
- Zum Testen: **F5** in ISE
- Mit **Write-Verbose** können im Skript Details zur Abarbeitung ausgegeben werden; normalerweise unterdrückt
- Mit **-verbose** beim Skriptaufruf werden diese Infos angezeigt

Skripting

- Demo: Verbose-Ausgabe in einem Skript



© ppedv AG

242

Skripting

- Im Skript kann eine Dokumentation hinterlegt werden
- Schlagworte:
 - Synopsis
 - Description
 - Parameter
 - Example
 - Weitere
- **help about_comment_based_help** (sehr umfangreich!)

Skripting

Beispiel:

```
<#
 .SYNOPSIS
Retrieves network adapter information from a computer.

 .DESCRIPTION
Uses CIM to retrieve information about physical adapters only.

 .PARAMETER ComputerName
The name of the computer to query.

 .EXAMPLE
.\Get-NetAdapterInfo.ps1 -ComputerName LON-DC1 -Verbose
#>
```

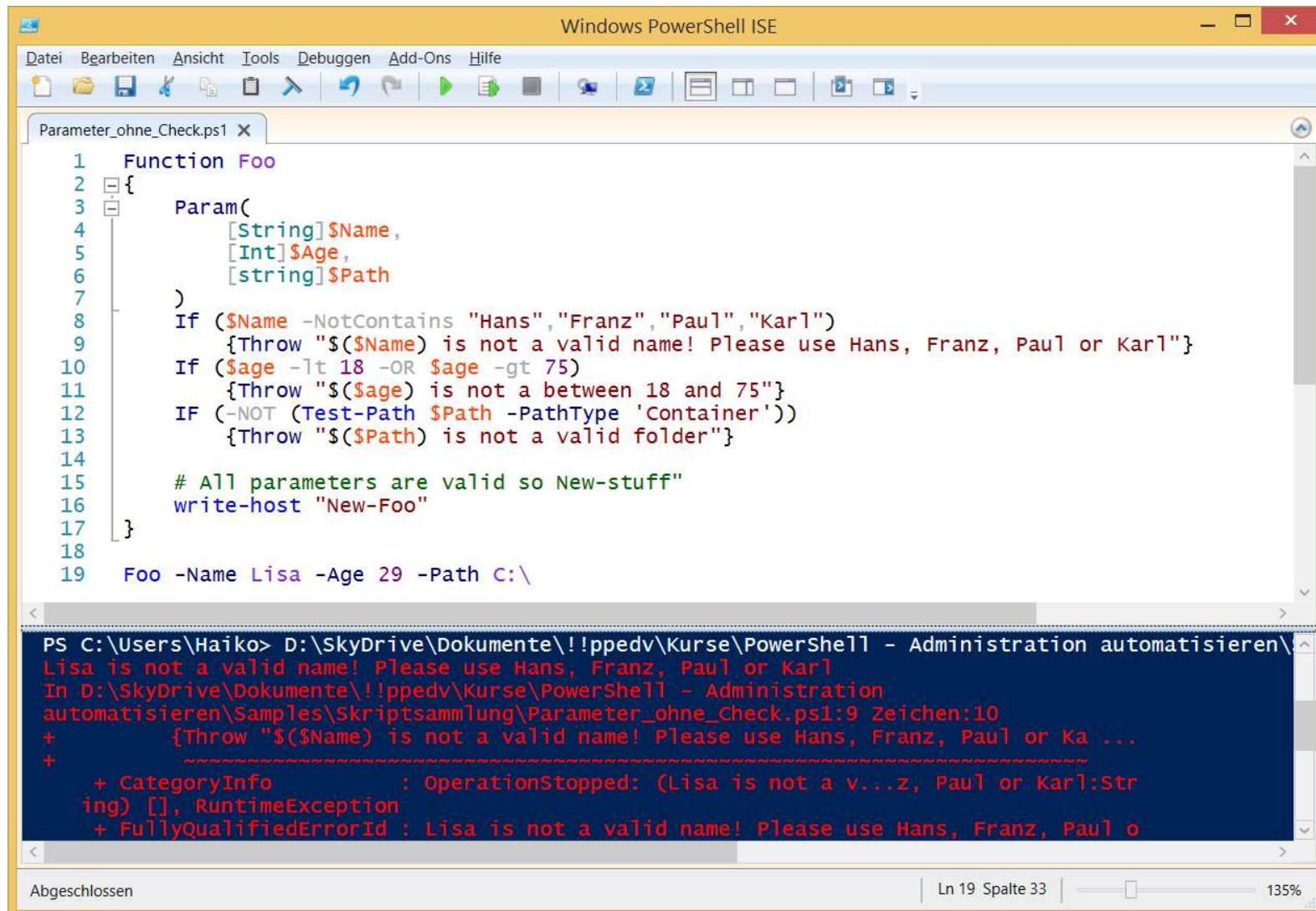
Demonstration: Adding Comment-Based Help

- In this demonstration, you will see how to add a basic comment-based help block to a script

Skripting

- Parameter können/sollten auf Gültigkeit getestet werden
- Eine Variante:

Skripting



The screenshot shows the Windows PowerShell ISE interface. The top window displays a PowerShell script named `Parameter_ohne_Check.ps1`. The script defines a function `Foo` that takes three parameters: `$Name`, `$Age`, and `$Path`. It contains validation logic: if `$Name` does not contain "Hans", "Franz", "Paul", or "Karl", it throws an error. If `$Age` is less than 18 or greater than 75, it throws an error. If `$Path` is not a valid folder, it throws an error. Otherwise, it writes "New-Foo" to the console.

```
1 Function Foo
2 {
3     Param(
4         [String]$Name,
5         [Int]$Age,
6         [string]$Path
7     )
8     If ($Name -NotContains "Hans", "Franz", "Paul", "Karl")
9         {Throw "$($Name) is not a valid name! Please use Hans, Franz, Paul or Karl"}
10    If ($age -lt 18 -OR $age -gt 75)
11        {Throw "$($age) is not a between 18 and 75"}
12    IF (-NOT (Test-Path $Path -PathType 'Container'))
13        {Throw "$($Path) is not a valid folder"}
14
15    # All parameters are valid so New-stuff"
16    write-host "New-Foo"
17 }
18
19 Foo -Name Lisa -Age 29 -Path C:\
```

The bottom window shows the execution of the script in the PowerShell ISE. The command `Foo -Name Lisa -Age 29 -Path C:\` is run, and the output shows that `Lisa` is not a valid name, resulting in an error message.

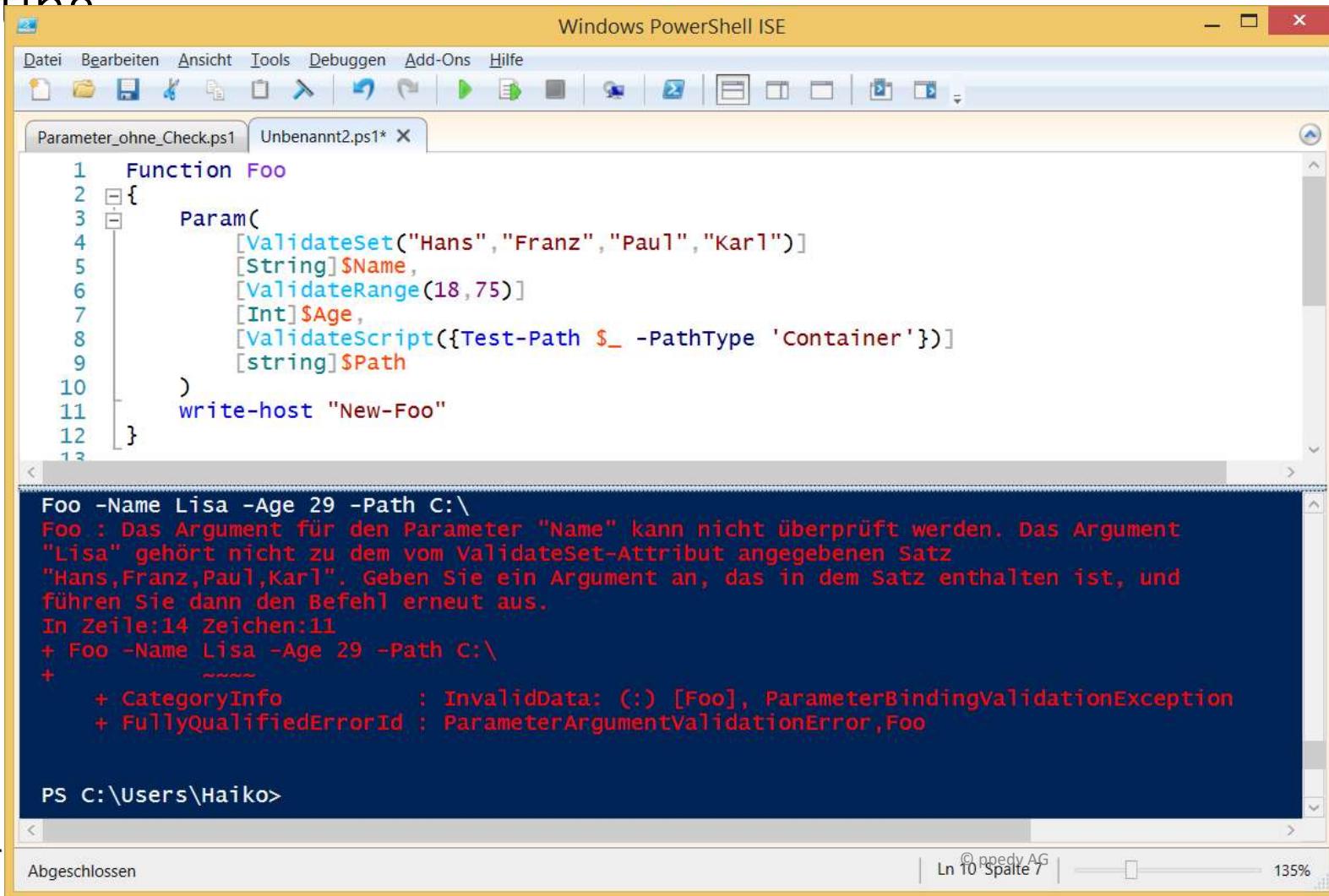
```
PS C:\Users\Haiko> D:\SkyDrive\Dokumente\!ppedv\Kurse\PowerShell - Administration automatisieren\Samples\Skriptsammlung\Parameter_ohne_Check.ps1:9 Zeichen:10
Lisa is not a valid name! Please use Hans, Franz, Paul or Karl
In D:\SkyDrive\Dokumente\!ppedv\Kurse\PowerShell - Administration
automatisieren\Samples\Skriptsammlung\Parameter_ohne_Check.ps1:9 Zeichen:10
+             {Throw "$($Name) is not a valid name! Please use Hans, Franz, Paul or Ka ...
+
+ CategoryInfo          : OperationStopped: (Lisa is not a v...z, Paul or Karl:Str
ing) [], RuntimeException
+ FullyQualifiedErrorId : Lisa is not a valid name! Please use Hans, Franz, Paul o
```

Skripting

- Besser: Gültige Parameter-Werte bei Definition festlegen!

```
Param(  
    [ValidateSet("Hans", "Franz", "Paul", "Karl")]  
    [String]$Name,  
    [ValidateRange(18, 75)]  
    [Int]$Age,  
    [ValidateScript({Test-Path $_ -PathType 'Container'})]  
    [string]$Path  
)
```

Skripting



The screenshot shows the Windows PowerShell ISE interface. The top menu bar includes Datei, Bearbeiten, Ansicht, Tools, Debuggen, Add-Ons, and Hilfe. The toolbar below has various icons for file operations. Two tabs are open: "Parameter_ohne_Check.ps1" and "Unbenannt2.ps1*". The code in "Parameter_ohne_Check.ps1" defines a function Foo with parameters \$Name, \$Age, and \$Path, and outputs "New-Foo". The execution output window shows the command "Foo -Name Lisa -Age 29 -Path C:\", followed by an error message indicating that "Lisa" does not belong to the ValidateSet "Hans, Franz, Paul, Karl". It also shows the error object with properties CategoryInfo, FullyQualifiedErrorId, and ErrorDetails. The bottom status bar indicates the session is "Abgeschlossen" (closed), with zoom set to 135%.

```
1 Function Foo
2 {
3     Param(
4         [ValidateSet("Hans", "Franz", "Paul", "Karl")]
5         [String]$Name,
6         [ValidateRange(18, 75)]
7         [Int]$Age,
8         [ValidateScript({Test-Path $_ -PathType 'Container'})]
9         [string]$Path
10    )
11    write-host "New-Foo"
12 }
```

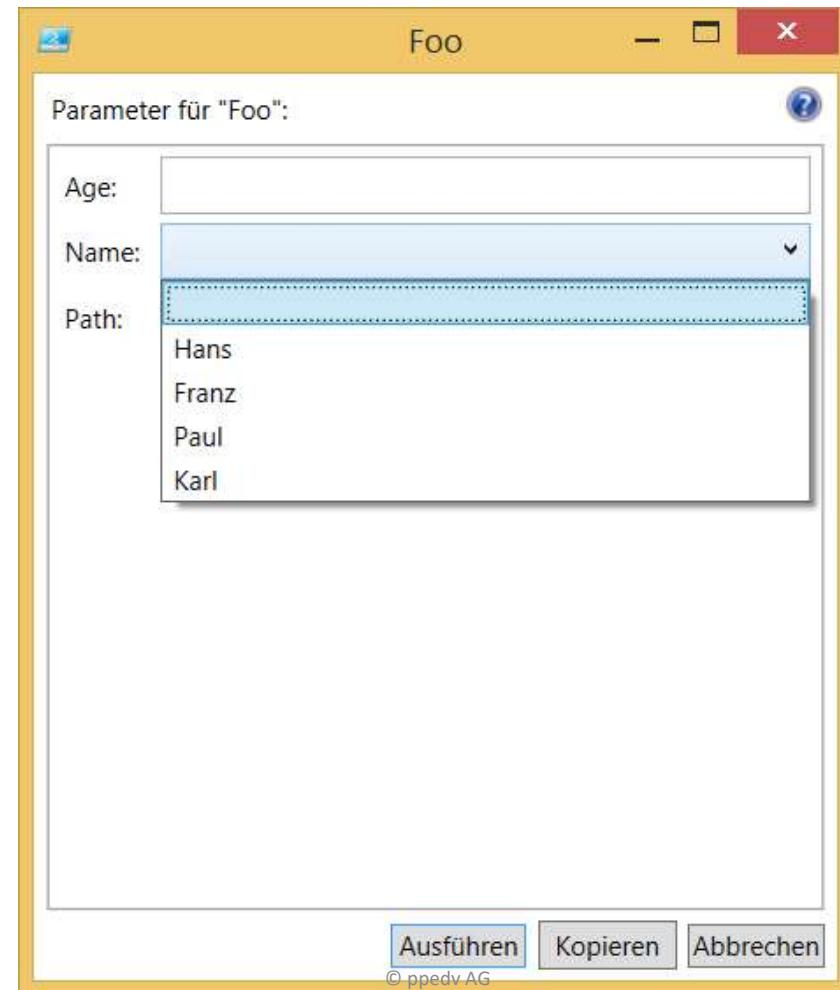
```
Foo -Name Lisa -Age 29 -Path C:\
Foo : Das Argument für den Parameter "Name" kann nicht überprüft werden. Das Argument
"Lisa" gehört nicht zu dem vom ValidateSet-Attribut angegebenen Satz
"Hans,Franz,Paul,Karl". Geben Sie ein Argument an, das in dem Satz enthalten ist, und
führen Sie dann den Befehl erneut aus.
In Zeile:14 Zeichen:11
+ Foo -Name Lisa -Age 29 -Path C:\
+   ~~~
+     CategoryInfo          : InvalidData: (:) [Foo], ParameterBindingValidationException
+     FullyQualifiedErrorId : ParameterArgumentValidationError, Foo

PS C:\Users\Haiko>
```

Abgeschlossen | © ppedv AG | Ln 10 Spalte 7 | 135%

Skripting

- Bei Show-Command:



Skripting

- Achtung: Nur Daten, die der User angibt, werden überprüft!

```
Param (
    [Parameter(Mandatory=$false, ValueFromPipeline=$true)]
    [ValidateRange(18, 75)]
    [Int]$Age = 100
)
```

Übung: Scripting

- Schreiben Sie ein kleines Script, welchem man Parameter übergeben kann, deren mögliche Werte sie vorgeben! Testen Sie das Script anschließend.
- Mögliche Ideen:
 - Abfrage bestimmter Einträge aus dem Ereignisprotokoll
 - Abfrage von AD-Objekten
 - Abfrage von Diensten
 - Abfrage von Dateien



Funktionen und Module

Skripting

- Code in Funktion verpacken:

```
function Get-SecurityEvents {  
    [CmdletBinding()]  
    Param(  
        [string]$ComputerName,  
        [int]$EventID  
    )  
    Get-EventLog -LogName Security -ComputerName $ComputerName |  
    Where EventID -eq $EventID |  
    Select -First 50  
}
```

Skripting

Wichtig:

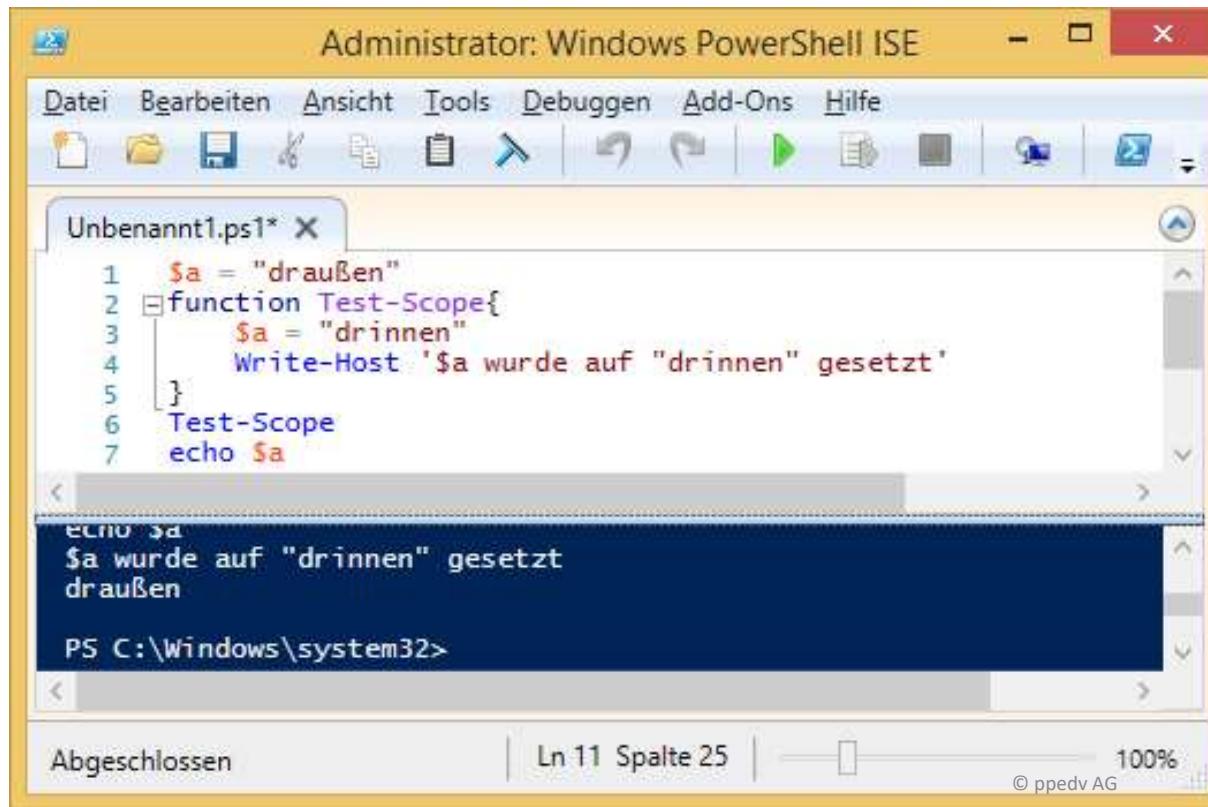
- Nur Code ausführen genügt nicht – dabei wird die Funktion nur definiert, nicht ausgeführt
- Wenn das Skript beendet ist, ist die Funktion u. U. nicht mehr „bekannt“
- Grund: **Scope** („Bereich“)

Skripting

- Scope existiert in der Shell um:
 - Die Shell selber
 - Jedes Skript, das ausgeführt wird
 - Jede Funktion, die erzeugt wird
- Objekte (u. A. Variablen, Funktionen, ...) in einem Scope existieren nur, solange der Scope existiert
- Mehr Info: **about_scope**

Skripting

- Ein kleines Beispiel für Scope:



The screenshot shows the Windows PowerShell ISE interface. The title bar reads "Administrator: Windows PowerShell ISE". The menu bar includes Datei, Bearbeiten, Ansicht, Tools, Debuggen, Add-Ons, and Hilfe. The toolbar contains various icons for file operations. A script window titled "Unbenannt1.ps1*" contains the following PowerShell code:

```
1 $a = "draußen"
2 function Test-Scope{
3     $a = "drinnen"
4     Write-Host '$a wurde auf "drinnen" gesetzt'
5 }
6 Test-Scope
7 echo $a
```

The output pane below shows the results of running the script:

```
ECHO: $a
$a wurde auf "drinnen" gesetzt
draußen

PS C:\Windows\system32>
```

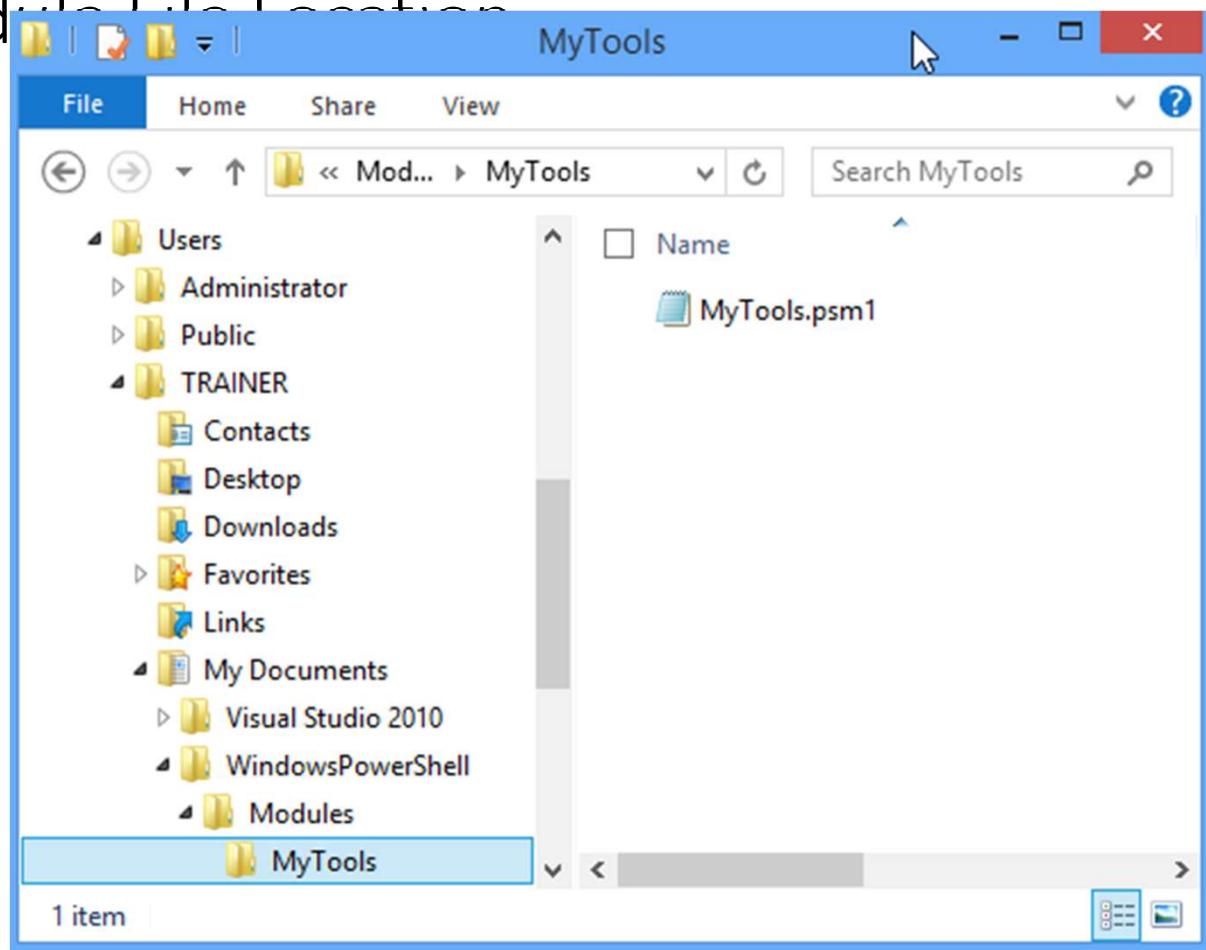
The status bar at the bottom indicates "Abgeschlossen" (Completed), "Ln 11 Spalte 25", and "100%". The copyright notice "© ppedv AG" is also visible.

Skripting

- PowerShell lädt Funktionen und ähnliches aus **Modulen**
- Diese werden seit PS 3.0 automatisch bei Bedarf geladen
- Funktionen können in eigenen Modulen abgelegt werden
- Wenn diese am richtigen Ort abgespeichert sind, werden sie ebenfalls nach Bedarf geladen
- Funktionen im Modul können dann wie reguläre Commandlets aufgerufen werden

C:\Windows\System32\WindowsPowerShell\v1.0\Modules

Script Module File Location



Skripting

- Debugging-Haltepunkte durch Write-Debug
- Ähnlich wie Write-Verbose
- Bei Ausführung mit -Debug wird die entsprechende Ausgabe erzeugt und die Ausführung des Skriptes angehalten; man kann jetzt:
 - Die Ausführung fortsetzen
 - Die Ausführung abbrechen
 - Die Ausführung pausieren und im aktuellen Scope Untersuchungen vornehmen



© ppedv AG

Begin-Process-End

Begin-Process-End

- Ziel: Mehrere Objekte sollen gegen eine eigene Funktion „ge-pipe-d“ werden

```
1 Function Test-Demo
2 {
3     Param ($Param1)
4
5     Write-Host "Start"
6     Write-Host "Verarbeite $input mit $Param1"
7     Write-Host "Ende"
8
9 }
10
11 Echo Objekt1, Objekt2 | % {$_. | Test-Demo Parameter}
```

```
< PowerShell, Objekte | % {$_ | Test-Demo Parameter} >
Start
Verarbeite Objekt1 mit Parameter
Ende
Start
Verarbeite Objekt2 mit Parameter
Ende
```

Begin-Process-End

- Ziel: Mehrere Objekte sollen gegen eine eigene Funktion „ge-pipe-d“ werden



```
1 Function Test-Demo
2 {
3     Param ($Param1)
4
5     Write-Host "Start"
6     Write-Host "Verarbeite $input mit $Param1"
7     Write-Host "Ende"
8 }
9
10 Echo Objekt1 mit Parameter
11 Echo Objekt2 mit Parameter
```

The screenshot shows a PowerShell script in a code editor. The script defines a function named 'Test-Demo' that takes a parameter '\$Param1'. Inside the function, it outputs 'Start', processes the input with the parameter, and then outputs 'Ende'. Two lines outside the function call it with 'Objekt1' and 'Objekt2' respectively, both followed by 'Parameter'. A large orange circle with a diagonal slash and the German text 'So nicht!' is overlaid on the right side of the script, indicating that this is a bad practice.

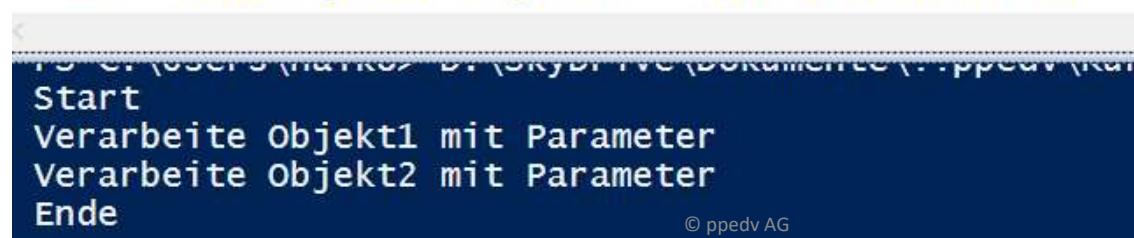
Begin-Process-End

- Grund: GESAMTE Funktion wird mehrfach komplett durchlaufen
- idR: Am Anfang der Funktion stehen Initialisierungen o.ä.
- Diese müssten normalerweise insgesamt nur 1x ausgeführt werden
- Lösung: **Begin-Process-End-Konstruktion**

Begin-Process-End

Beispiel:

```
1 Function Test-Demo
2 {
3     Param ($Param1)
4     Begin
5     {
6         Write-Host "Start"
7     }
8     Process
9     {
10        Write-Host "Verarbeite" $_ mit $Param1
11    }
12    End
13    {
14        Write-Host "Ende"
15    }
16}
17
18 Echo Objekt1, Objekt2 | Test-Demo Parameter
```



```
<snip>
Start
Verarbeite Objekt1 mit Parameter
Verarbeite Objekt2 mit Parameter
Ende
```

Begin-Process-End

- **Begin**-Block wird nur einmal ausgeführt, z. B. zum Initialisieren o.ä.
- **Process**-Block wird für jedes Objekt, das der Funktion übergeben wurde, ausgeführt (idR also mehrfach)
- **End**-Block wird am Ende nur einmal ausgeführt, z. B. zum „Aufräumen“

Begin-Process-End

- Funktionsdeklarationen sollten in den **Begin**-Block

5

TRAINING



Fehlerbehandlung

Fehlerbehandlung

- Fehler in Skripten haben verschiedene Ursachen
- Teilweise durch fehlerhaften Code
- Aber auch durch „unpassende“ Umgebung oder auch Nichterreichbarkeit von Systemen möglich
- Damit Skript nicht einfach abbricht: Fehlerbehandlung nötig!

Fehlerbehandlung

- Terminierende Fehler brechen ein Kommando ab
- Nicht-terminierende Fehler machen es möglich, dass ein Kommando weiterarbeitet (z. B. nur Ausgabe einer Warnung)
- Bei nicht-terminierenden Fehlern muss das Kommando wissen, wie es verfahren soll
- Dafür: **ErrorAction**

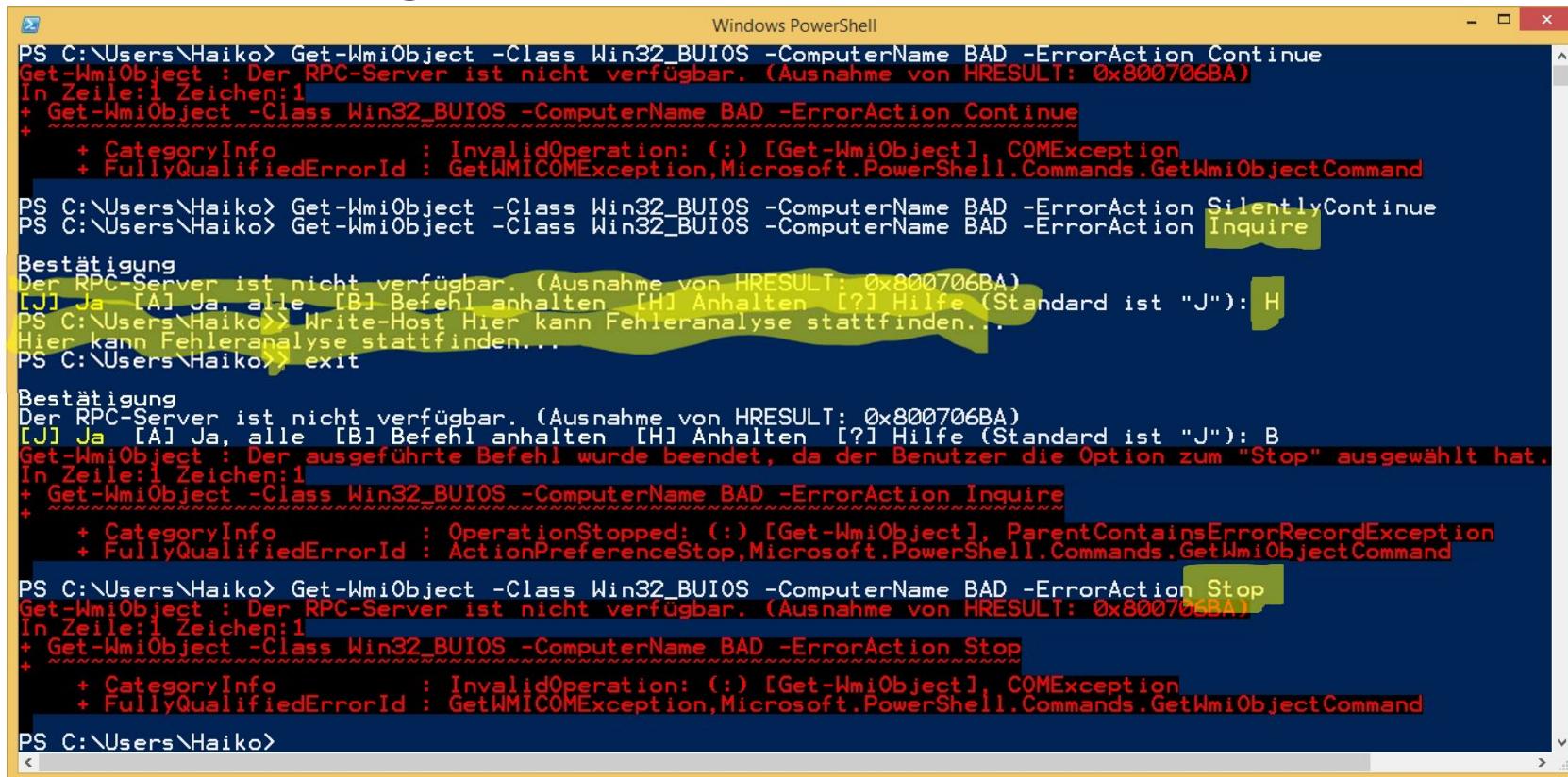
Fehlerbehandlung

- **\$ErrorActionPreference** ist eine globale Variable
- Setzt das Standard-Verhalten bei nicht-terminierenden Fehlern
 - **Continue** – Standard; lässt Kommando weitermachen
 - **SilentlyContinue** – unterdrückt Fehlermeldungen und lässt Kommando weitermachen
 - **Inquire** – fragt den User nach Vorgehen
 - **Stop** – Behandelt den Fehler wie einen terminierenden und bricht das Kommando ab

Fehlerbehandlung

- Parameter **-ErrorAction** steuert Fehler-Aktion individuell
- Alias: **-ea**
- Wirkt nur auf das aktuelle Kommando
- Verfügbar für alle:
 - Cmdlets
 - Funktionen
 - Workflows
- Nicht verfügbar für Methoden

Fehlerbehandlung



The screenshot shows a Windows PowerShell window with several command-line examples demonstrating error handling:

```
Windows PowerShell
PS C:\Users\Haiko> Get-WmiObject -Class Win32_BIOS -ComputerName BAD -ErrorAction Continue
Get-WmiObject : Der RPC-Server ist nicht verfügbar. (Ausnahme von HRESULT: 0x800706BA)
In Zeile:1 Zeichen:1
+ Get-WmiObject -Class Win32_BIOS -ComputerName BAD -ErrorAction Continue
+ CategoryInfo          : InvalidOperation: () [Get-WmiObject], COMException
+ FullyQualifiedErrorId : GetWMICOMException,Microsoft.PowerShell.Commands.GetWmiObjectCommand

PS C:\Users\Haiko> Get-WmiObject -Class Win32_BIOS -ComputerName BAD -ErrorAction SilentlyContinue
PS C:\Users\Haiko> Get-WmiObject -Class Win32_BIOS -ComputerName BAD -ErrorAction Inquire
Bestätigung
Der RPC-Server ist nicht verfügbar. (Ausnahme von HRESULT: 0x800706BA)
[J] Ja [A] Ja, alle [B] Befehl anhalten [H] Anhalten [?] Hilfe (Standard ist "J"): H
PS C:\Users\Haiko>> Write-Host Hier kann Fehleranalyse stattfinden...
Hier kann Fehleranalyse stattfinden...
PS C:\Users\Haiko>> exit

Bestätigung
Der RPC-Server ist nicht verfügbar. (Ausnahme von HRESULT: 0x800706BA)
[J] Ja [A] Ja, alle [B] Befehl anhalten [H] Anhalten [?] Hilfe (Standard ist "J"): B
Get-WmiObject : Der ausgeführte Befehl wurde beendet, da der Benutzer die Option zum "Stop" ausgewählt hat.
In Zeile:1 Zeichen:1
+ Get-WmiObject -Class Win32_BIOS -ComputerName BAD -ErrorAction Inquire
+ CategoryInfo          : OperationStopped: () [Get-WmiObject], ParentContainsErrorRecordException
+ FullyQualifiedErrorId : ActionPreferenceStop,Microsoft.PowerShell.Commands.GetWmiObjectCommand

PS C:\Users\Haiko> Get-WmiObject -Class Win32_BIOS -ComputerName BAD -ErrorAction Stop
Get-WmiObject : Der RPC-Server ist nicht verfügbar. (Ausnahme von HRESULT: 0x800706BA)
In Zeile:1 Zeichen:1
+ Get-WmiObject -Class Win32_BIOS -ComputerName BAD -ErrorAction Stop
+ CategoryInfo          : InvalidOperation: () [Get-WmiObject], COMException
+ FullyQualifiedErrorId : GetWMICOMException,Microsoft.PowerShell.Commands.GetWmiObjectCommand

PS C:\Users\Haiko>
```

Fehlerbehandlung

- Wenn Fehler wahrscheinlich sind bzw. bei deren Auftreten individuell verfahren werden soll: **Try/Catch** Konstruktion!
- **Try**-Block enthält den potentiellen Fehler
 - Nach Möglichkeit nur eine Aktion ausführen!
 - **-ErrorAction** auf **Stop** setzen
- **Catch**-Block enthält, was bei Fehlern ausgeführt werden soll
- **Finally**-Block wird in jedem Fall ausgeführt

Fehlerbehandlung

```
Try {  
    Get-WmiObject -Class Win32_Service -  
    ComputerName $name -ErrorAction Stop  
}  
  
Catch {  
    Write-Verbose "Error connecting to $name"  
}
```

Fehlerbehandlung

- Demo: Try...Catch



© ppedv AG

276

Fehlerbehandlung

- Weitere Ausbaustufe:

```
Try {  
    # Etwas, das schief gehen könnte  
}  
Catch {  
    # Was passiert, wenn es schief geht?  
}  
Finally {  
    # Was soll in jedem Fall passieren?  
}
```

Fehlerbehandlung

- Fehler können gespeichert werden
- Dadurch kann im Nachgang auch individuell auf Fehler eingegangen werden
- Das eingebaute **\$Error** Array speichert die aktuellste Fehlermeldung in **\$Error[0]**
- Mit dem Parameter **-ErrorVariable (-EV)** kann auch ein alternativer Variablenname für die Speicherung bestimmt werden
 - Variablennamen enthalten kein \$!
 - Speichert nur die Fehler des benannten Kommandos

Fehlerbehandlung

- Demo: -ErrorVariable bzw. \$Error



© ppedv AG

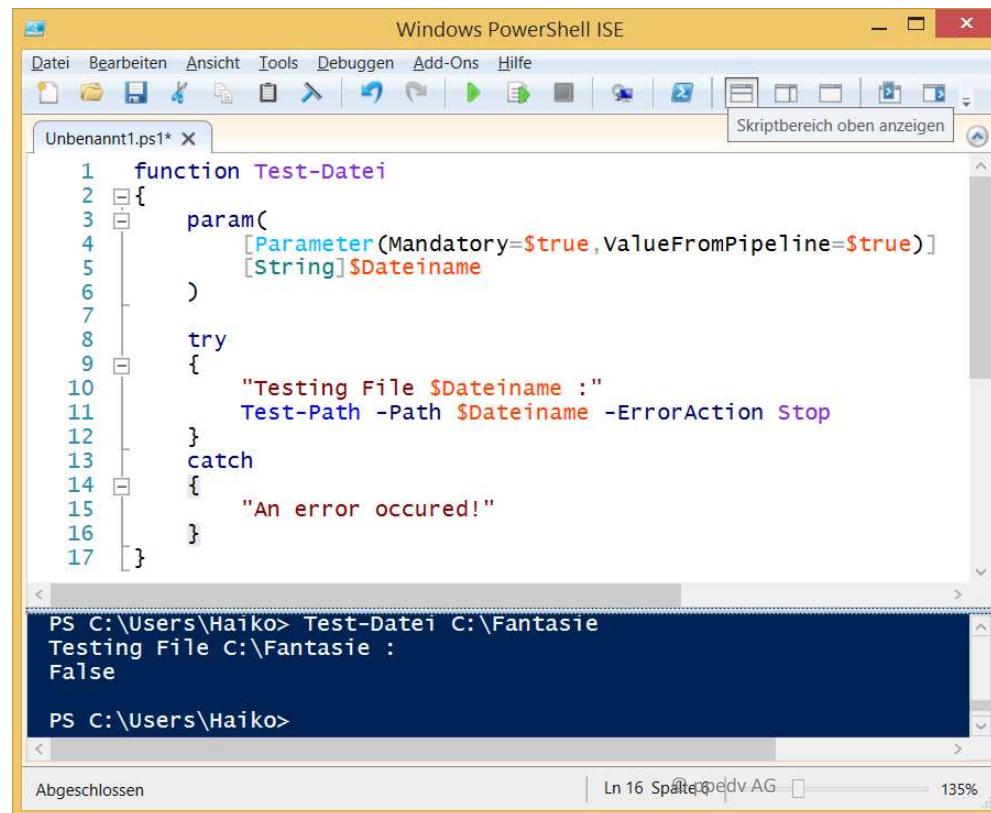
279

Fehlerbehandlung

- Wenn bei einem Fehler IMMER in den Catch-Block gesprungen werden soll:
-ErrorAction Stop !

Fehlerbehandlung

- Nicht jedes Commandlet wirft einen Fehler, wo man ihn vielleicht erwartet hätte:



The screenshot shows the Windows PowerShell ISE interface. The top menu bar includes Datei, Bearbeiten, Ansicht, Tools, Debuggen, Add-Ons, and Hilfe. Below the menu is a toolbar with various icons. The main window title is "Unbenannt1.ps1*". The script content is as follows:

```
1 function Test-Datei
2 {
3     param(
4         [Parameter(Mandatory=$true,ValueFromPipeline=$true)]
5         [String]$Dateiname
6     )
7
8     try
9     {
10        "Testing File $Dateiname :"
11        Test-Path -Path $Dateiname -ErrorAction Stop
12    }
13    catch
14    {
15        "An error occurred!"
16    }
17 }
```

In the bottom pane, the PowerShell command "Test-Datei C:\Fantasie" is run, resulting in the output:

```
PS C:\Users\Haiko> Test-Datei C:\Fantasie
Testing File C:\Fantasie :
False

PS C:\Users\Haiko>
```

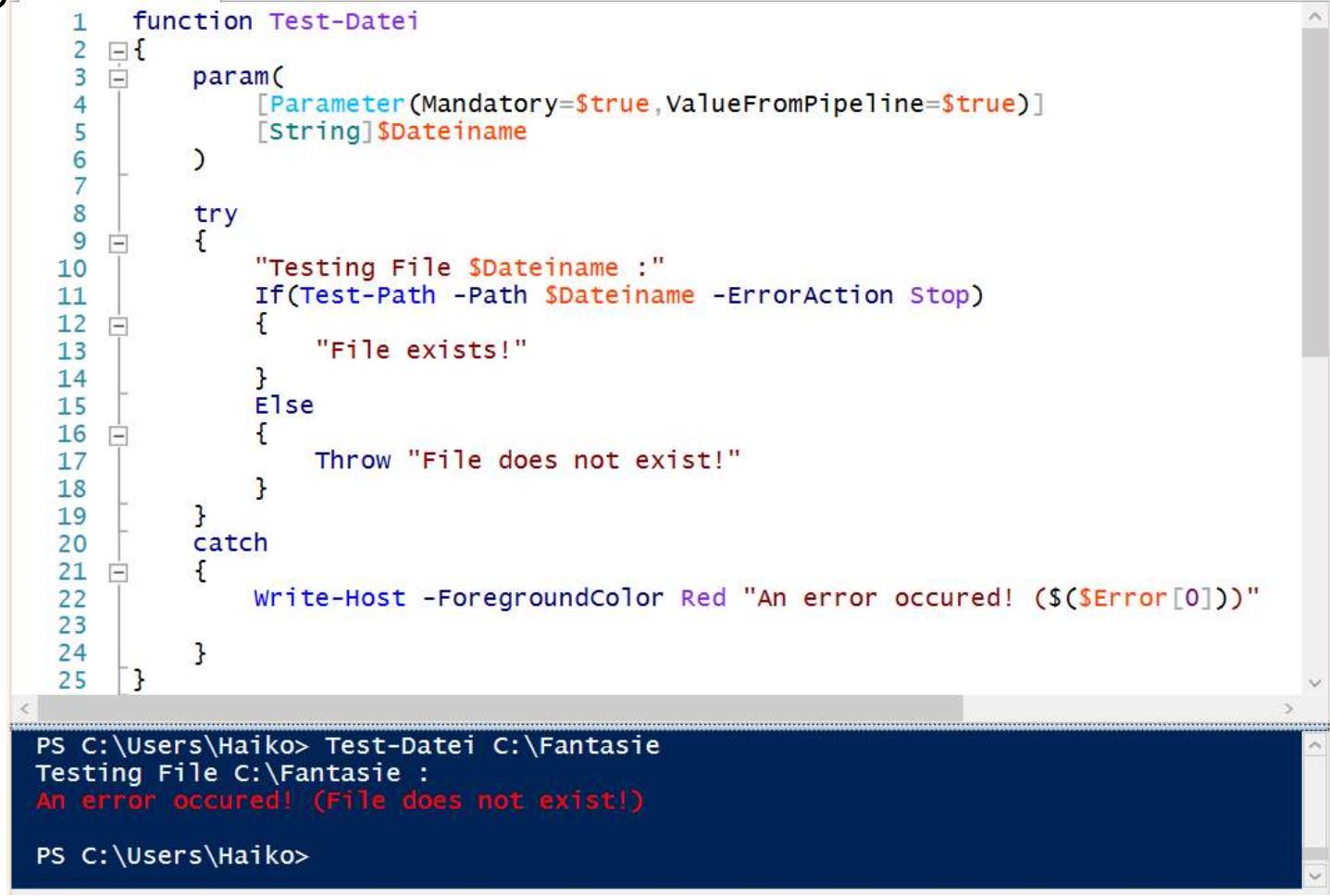
At the bottom of the ISE window, status information includes "Abgeschlossen", "Ln 16 Spalte 6", and a zoom level of "135%".

Fehlerbehandlung

- Test-Path liefert idR True oder False
- False allein ist aber kein Fehler!
- Bei Bedarf: Fehler individuell erzeugen
- **Throw bzw. Write-Error**

Fehlerbehandlung

```
1 function Test-Datei
2 {
3     param(
4         [Parameter(Mandatory=$true,ValueFromPipeline=$true)]
5         [String]$Dateiname
6     )
7
8     try
9     {
10        "Testing File $Dateiname :"
11        If(Test-Path -Path $Dateiname -ErrorAction Stop)
12        {
13            "File exists!"
14        }
15        Else
16        {
17            Throw "File does not exist!"
18        }
19    }
20    catch
21    {
22        Write-Host -ForegroundColor Red "An error occurred! ($($Error[0]))"
23    }
24 }
25 }
```



PS C:\Users\Haiko> Test-Datei C:\Fantasie
Testing File C:\Fantasie :
An error occurred! (File does not exist!)

PS C:\Users\Haiko>

Fehlerbehandlung

- Fehler individuell behandeln:

```
1  try
2  {
3      1/0
4  }
5  catch [DivideByZeroException]
6  {
7      Write-Host "Divide by zero exception"
8  }
9
10 catch [System.Net.WebException], [System.Exception]
11 {
12     Write-Host "Other exception"
13 }
14
15 finally
16 {
17     Write-Host "cleaning up ..."
18 }
```

```
' Divide by zero exception
' cleaning up ...
```

Übung: Fehlerbehandlung

- Schreiben Sie ein kleines Script, welches Try-Catch verwendet, z. B. für eine Abfrage von AD-Benutzern!



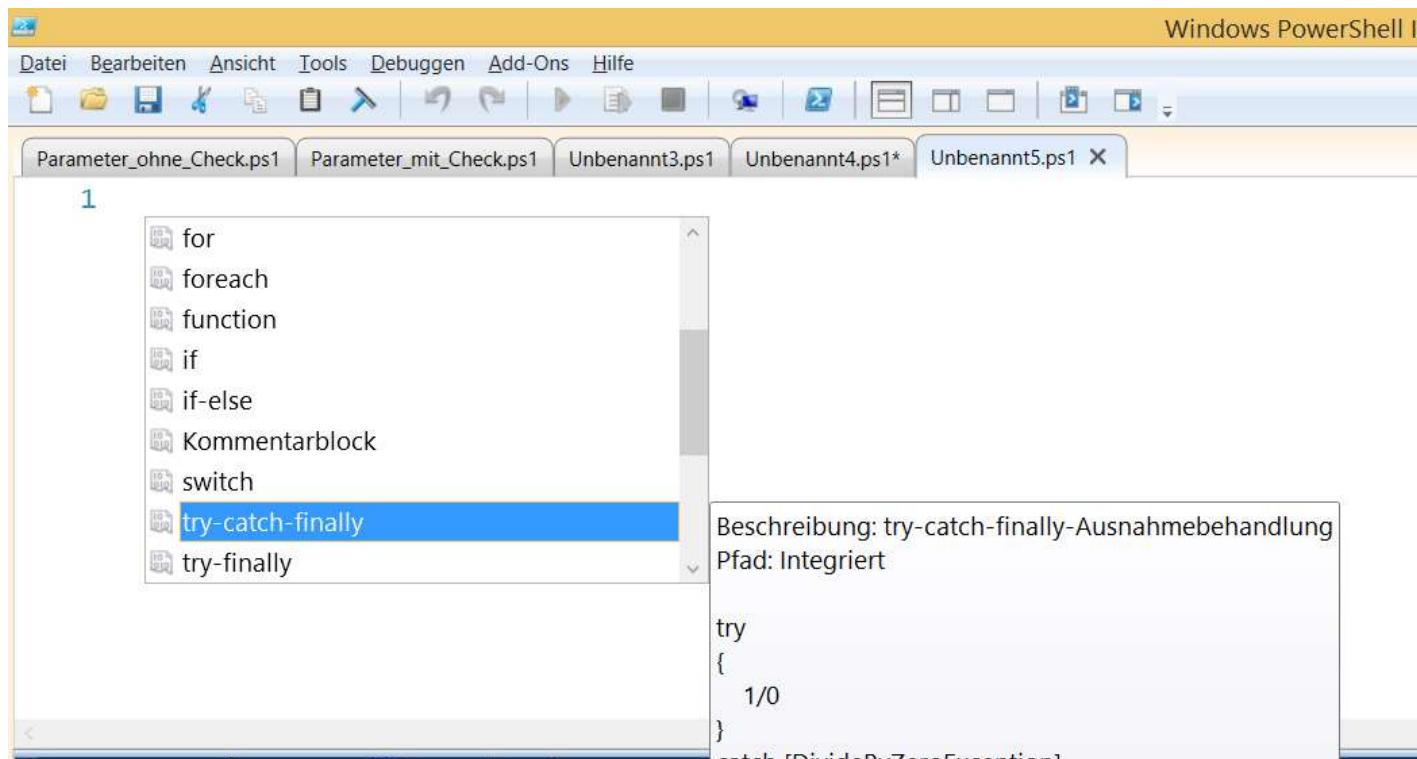
Skripting-Tips

Tipps

- Nicht vermuten – sicherstellen!
 - Bsp.: Aufruf/Skript benötigt einen Dienst, der „normalerweise“ läuft -> Skript-Status prüfen, notfalls starten
- Erwarte das unerwartete – und behandle es!
- Möglichst keine Kurzformen
- Abwärtskompatibilität so weit wie möglich
- Skript in möglichst viele „Teile“ zerlegen, z. B. durch Funktionen

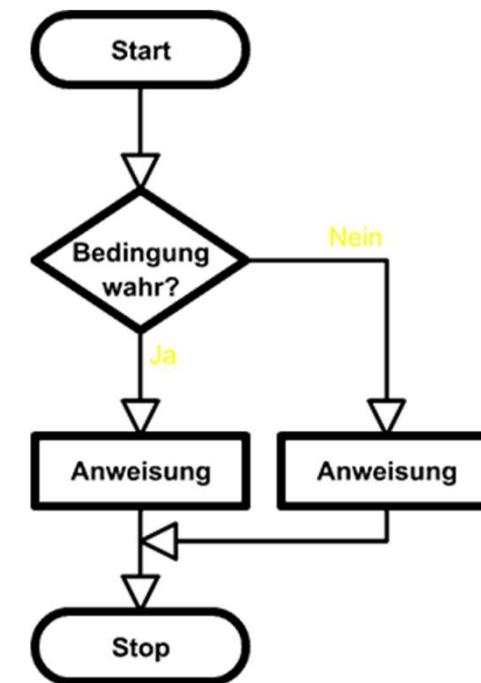
Tipps

- ISE: [STRG]+[J] -> Fertige Skript-Blöcke!



Tipps

- Programm-Ablauf-Diagramme helfen!
- Dokumentieren ist sehr wichtig...
- ... und Testen auch!



PowerShell Remoting

PS Remoting

- Ermöglicht interaktives Login auf remote PCs
- Viele Commandlets: **-Computer** Parameter, um Ziel anzugeben (z. B. **Restart-Computer** oder auch **Get-WmiObject**)
- Muss u.U. erst mit **Enable-PSRemoting** aktiviert werden
- Universelles Remoting: Auch Cmdlets ohne **-Computer** können remote ausgeführt werden

PS Remoting

1. Cmdlets mit „ComputerName“-Parameter
2. Interaktive Remote-Session im „Telnet-Stil“ Enter-PSSession
-Computername server1
3. Remote-Ausführung einzelner Cmdlets oder eines Scripts

PS Remoting

Enter-PSSession

```
-ScriptBlock {ipconfig.exe}  
-ComputerName SRV2
```

oder

Invoke-Command

```
-FilePath xyz.ps1  
-ComputerName
```

PS Remoting

Session erstellen/verwenden

```
$s = New-PSSession  
    -Computername server1
```

```
Invoke-Command -session $s  
    -ScriptBlock { ... }
```

Übung: PS Remoting

- Testen Sie den universellen Remote-Zugriff mittels **Enter-PSSession** und **Invoke-Command**
- Finden Sie ein oder zwei Commandlets, die mittels **-ComputerName** Parameter ein direktes Remoting zulassen



PowerShell Web Access

PS Web Access

- Stellt „Powershell-Gateway“ bereit
- Zugriff (z. B. von außen) über Webanwendung
- Tunnelung nicht nur zum PS Web Access Server sondern zu jedem beliebigen Zielsystem mit PSRemoting

PS Web-Access

- ⇒ PowerShell Web-Access als Feature installieren
- ⇒ Install-PswaWebApplication -WebSiteName „Default Web Site“ -UseTestCertificate
- ⇒ Add-PswaAuthorizationRule * * *
- ⇒ <https://SRV1/pswa>



Übung: PowerShell Web Access

- Installieren und konfigurieren Sie PSWA auf dem Server (Default Web Site)
- Nutzen Sie dazu ein Testzertifikat
- Testen Sie den Zugriff vom Hostsystem aus
(<https://server.ppedv.kurs/pswa>)



Desired State Configuration

Erst ab PS 4.0!

Desired State Configuration

- Ermöglicht „Vorkonfiguration“ eines oder mehrerer Server
- Server verarbeiten „Skript“ und stellen gewünschten Zustand her
 - Rollen/Features
 - Dienste
 - Berechtigungen
 - ...

Desired State Configuration

- Nutzt MOF-Files (Management Object Format)
- Erlaubte Konfigurationselemente:
 - WindowsFeature – konfiguriert Windows Features
 - Registry – setzt/löscht RegKeys
 - Script – Führt PowerShell Skriptblöcke aus
 - Archive – Entpackt Zip Files
 - File – Datei Operationen, kopiert Files und Ordner oder löscht sie
 - Package – Paketdateien (wie MSI oder EXE) können ausgeführt werden
 - Environment – Setzt/löscht Umgebungsvariablen
 - Group – Erstellt/löscht Lokale Gruppen
 - User – Erstellt/löscht Lokale User
 - Log – Schreibt Konfigurationsmeldungen ins Log
 - Service – Verwaltet Dienste
 - WindowsProcess – Konfiguriert Prozesse

Desired State Configuration

```
Configuration MyWebServers
{
    Node("SRV1.kurs.intern", "SRV2.kurs.intern")
    {
        WindowsFeature IIS
        {
            Ensure = "Present"
            Name = "Web-Server"
        }

        Service WWW-Dienst
        {
            Name = "w3svc"
            StartUpType = "Automatic"
        }

        Group Webadmins
        {
            Ensure = "Present"
            Groupname = "Webadministratoren"
        }

        File Webfolder
        {
            Ensure = "Present"
            Type = "Directory"
            Recurse = $true
            SourcePath = '\\$SRV1\Demofiles'
            DestinationPath = "C:\inetpub\wwwroot"
            DependsOn = "[WindowsFeature]IIS"
        }
    }
}
```

Desired State Configuration

- Configuration-Script erzeugen & ausführen
- Ausführen der erzeugten Config:

```
PS C:\Users\administrator.KURS> MyWebServers

Verzeichnis: C:\Users\administrator.KURS\MyWebServers

Mode          LastWriteTime    Length Name
----          -----        ---- 
-a---  27.02.2014 22:57       2954 SRV2.kurs.intern.mof
```

- Dabei entsteht ein *.mof File
- Start-DSCConfiguration [-Wait] –Path .\ConfigName



Desired State Configuration

- Über ein Ressource Kit lassen sich weitere Provider nutzen, z. B. ist dann die Steuerung der Firewall möglich
- Downloadquelle:
- <https://gallery.technet.microsoft.com/scriptcenter/DSC-Resource-Kit-All-c449312d>

Fragen und offene Punkte

?

StefanO@ppedv.de