# PowerShell

Die PowerShell (Programmdatei: powershell.exe) ist eine objektorientierte Befehlszeile auf Grundlage der .NET Architektur. Sie soll die klassische Eingabeaufforderung (Programmdatei: cmd.exe) in Zukunft immer mehr ablösen. Microsoft hat über viele Jahre die Weiterentwicklung der Kommandozeilen-Schnittstelle sträflich vernachlässigt und voll auf das GUI gesetzt und versucht nun den Rückstand zu den Unix basierten Systemen wieder wett zu machen. Erstmals ab Windows Server 2008 R2 ist diese Schnittstelle nun bereits standardmäßig installiert.

# Grundlagen

Die meisten bisherigen Kommandos und Batchdateien der Eingabeaufforderung funktionieren weiterhin auch in der PowerShell:

Help liefert eine Liste der Befehle

ASSOC Zeigt Dateierweiterungszuordnungen an bzw. ändert sie.  
ATTRIB Zeigt Dateiattribute an bzw. ändert sie.  
BREAK Schaltet die erweiterte Überprüfung für STRG+C ein bzw. aus.  
BOOTCFG Legt Eigenschaften zur Steuerung des Startladevorgangs in der Startdatenbank fest.  
CACLS Zeigt Datei-ACLs (Access Control List) an bzw. ändert sie.  
CALL Ruft eine Batchdatei von einer anderen Batchdatei aus auf.  
CD Zeigt den Namen des aktuellen Verzeichnisses an bzw. ändert diesen.  
CHCP Zeigt die aktive Codepagenummer an bzw. legt sie fest.  
CHDIR Zeigt den Namen des aktuellen Verzeichnisses an bzw. ändert es.  
CHKDSK Überprüft einen Datenträger und zeigt einen Statusbericht an.  
…

SYSTEMINFO Zeigt computerspezifische Eigenschaften und Konfigurationen an.  
TASKLIST Zeigt alle zurzeit laufenden Aufgaben inklusive der Dienste an.  
TASKKILL Bricht einen laufenden Prozess oder eine Anwendung ab oder beendet ihn bzw. sie.  
TIME Zeigt die Systemzeit an bzw. legt sie fest.  
TITLE Bestimmt den Fenstertitel des Eingabeaufforderungsfensters.  
TREE Zeigt die Ordnerstruktur eines Laufwerks oder Pfads grafisch an.  
TYPE Zeigt den Inhalt einer Textdatei an.  
VER Zeigt die Windows-Version an.  
VERIFY Legt fest, ob das ordnungsgemäße Schreiben von Dateien auf den Datenträger  
 überprüft werden soll.  
VOL Zeigt die Volumebezeichnung und die Seriennummer des Datenträgers an.  
XCOPY Kopiert Dateien und Verzeichnisstrukturen.  
WMIC Zeigt WMI-Informationen in der interaktiven Befehlsshell an.

Auch die Skripte des WSH (=Windows Scripting Host) sind unter der PowerShell weiterhin ausführbar.

# Cmdlets, Verb-Subtantiv

Die neuen Kommandos der PowerShell heißen nun **Cmdlets** und haben den grundsätzlichen Aufbau **Verb-Substantiv**

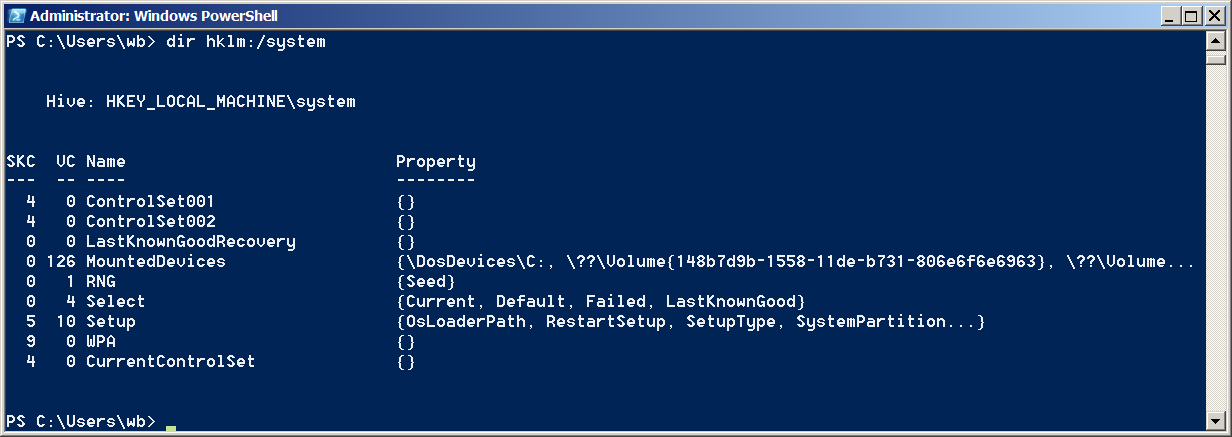
wie z.B.:   
  
Get-Help  
Update-Help (unbedingt mit administrativen Rechten!)

Evt: Update-Help -Verbose -Force -ErrorAction SilentlyContinue

Get-Process  
Start-Service

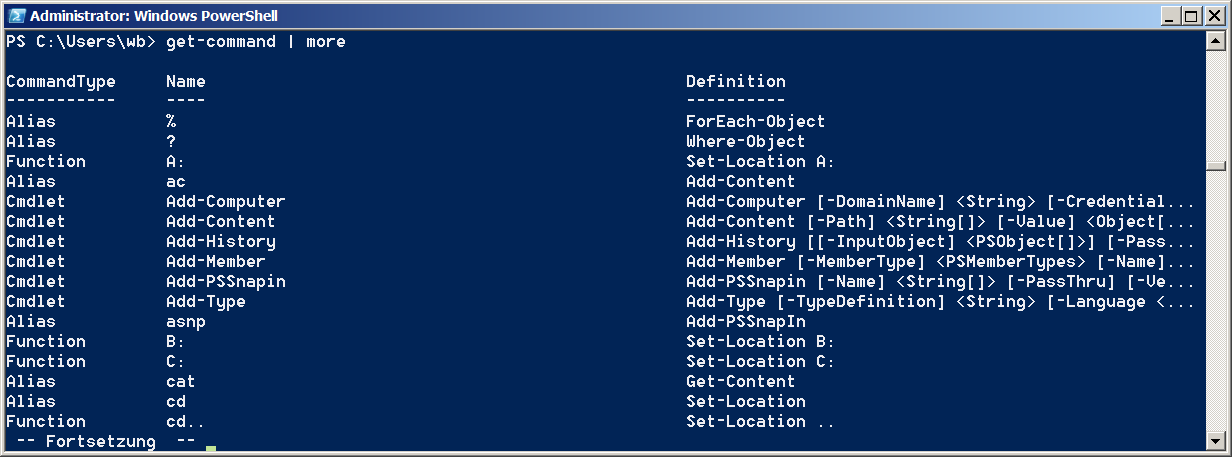
Powershell Laufwerke

Neben den bisher bekannten Laufwerksbuchstaben wie z.B. C:, D: usw. unterstützt die Powershell auch Registrystrukturen wie HKLM:, oder auch HKCU:



PowerShell Cmdlets, Aliases, Functions

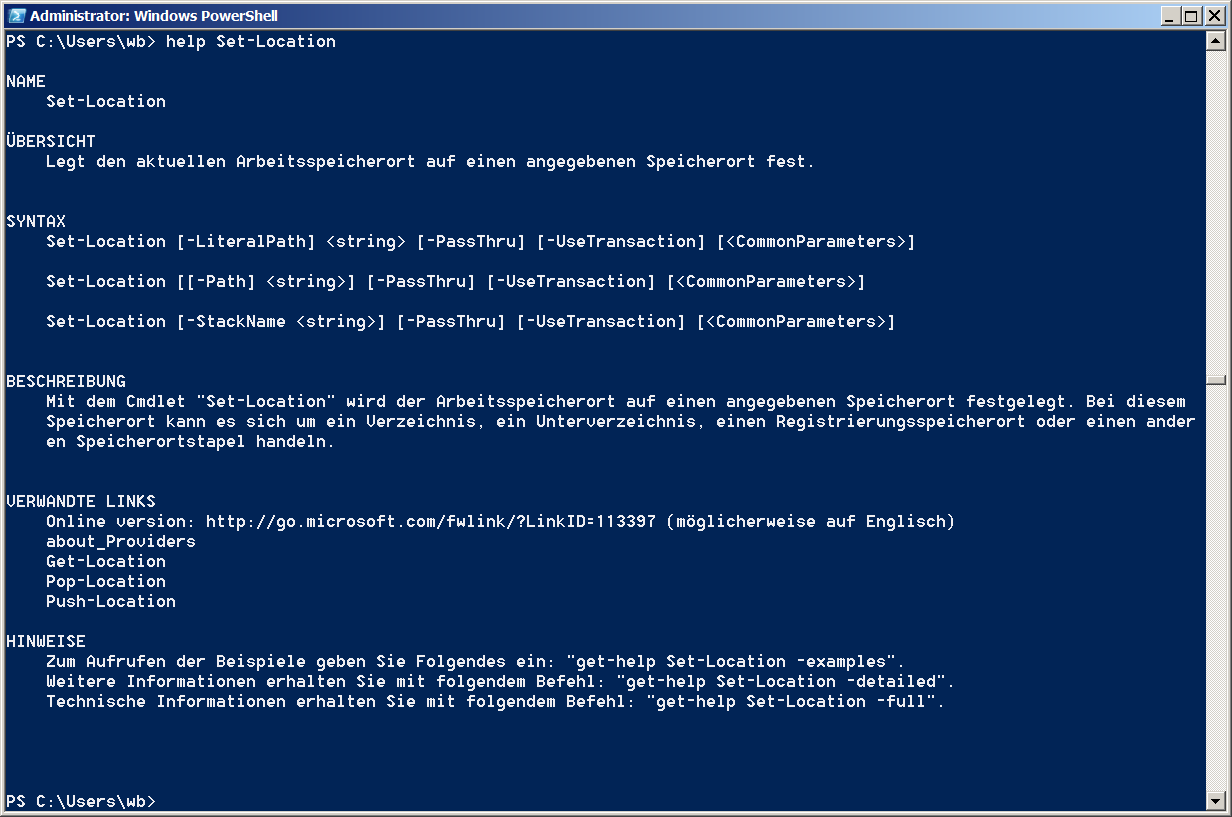
Mit get-command kann man sich eine Liste aller PowerShell Befehle, also eine komplette Referenz ausgeben lassen.

  
Wie das obige Cmdlet get-command demonstriert gibt es Cmdlets, Aliases und Functions.

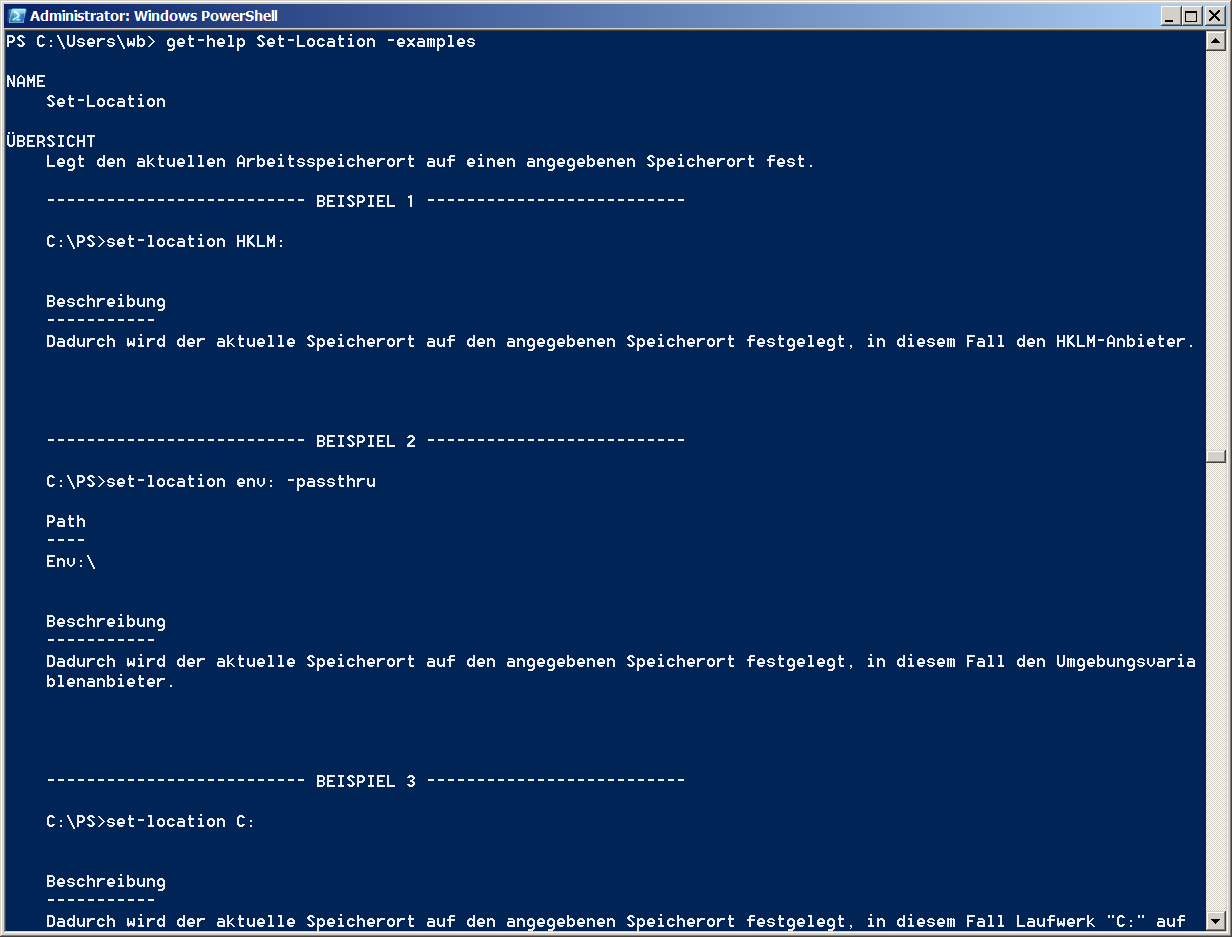
Die meisten der klassischen Befehle werden so auf die neuen Cmdlets, Aliases oder Funktionen der Powershell abgebildet, wie etwa im Falle von:

A: -> Set-Location A:

Die Referenz (=Informationen) zu jedem Befehl erhält man z.B. mit:



Sehr häufig ist es auch nützlich sich Musterbeispiele für Befehle anzeigen zu lassen:



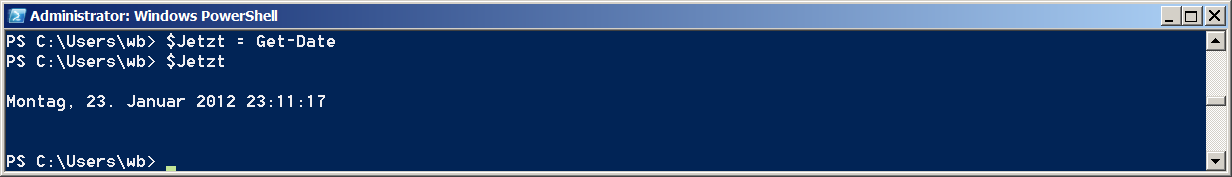
# Weitere Beispiele:

Test-NetConnection

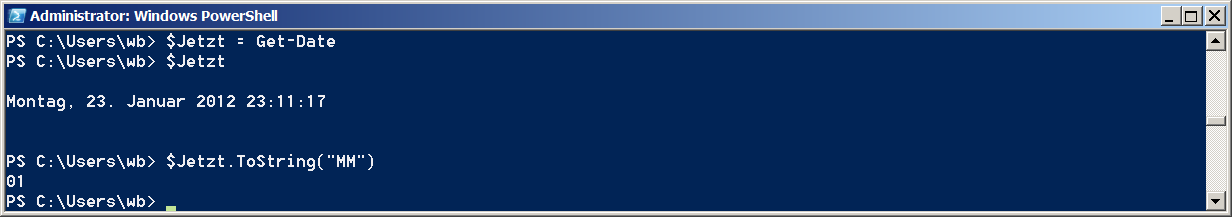
PowerShell Variable

Powershell Variable werden nun wie in Unix-Systemen üblich mit dem Präfix Character $ bezeichnet, und nicht mehr in der %Name% Syntax:

Beispiel:



Will man nun auf bestimmte Daten innerhalb der Variablen jetzt zugreifen, so kann man sich z.B. mit der Methode ToString("MM") nur den Monat aus der Variablen $Jetzt anzeigen lassen:



$Jetzt.ToString"yyyy"

zeigt das Jahr.

Oder auch direkt ohne Variable:

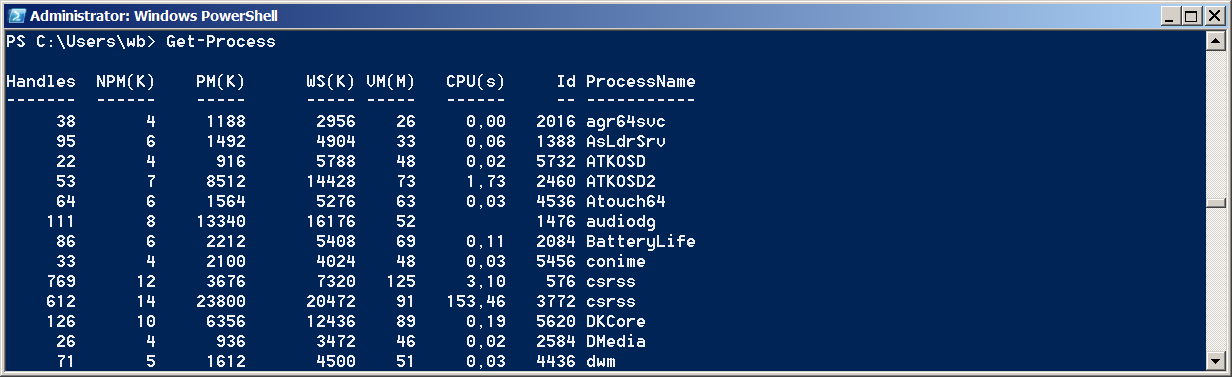
(Get-Date).tostring("yyyy")

$Jetzt | Get-Member

zeigt die verfügbaren Methoden an.

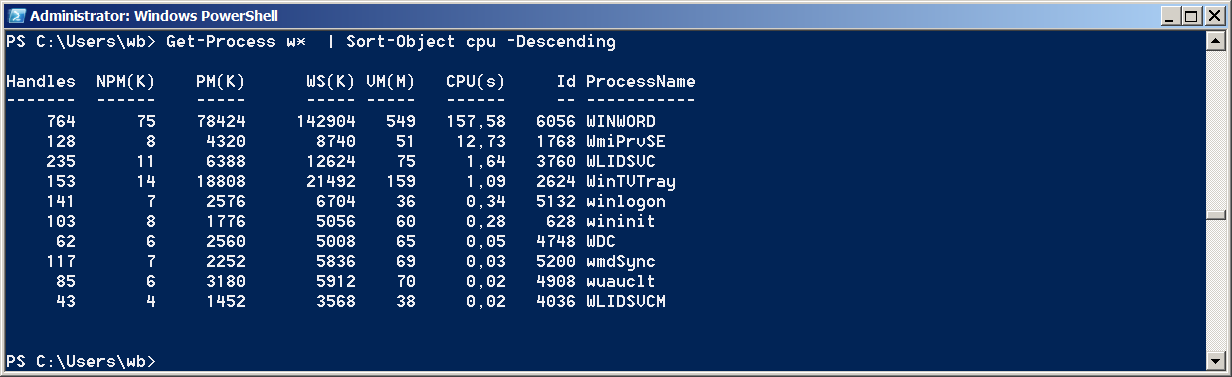
PowerShell und Prozesse

Mit get-process



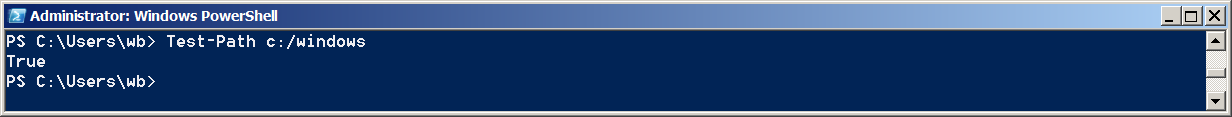
Kann man sich die Informationen zu den im System laufenden Prozessen anzeigen lassen.

Auch die Verwendung von Wildcards und speziell sortierte Ausgabe ist möglich:

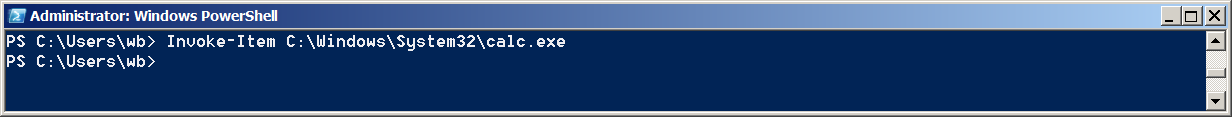


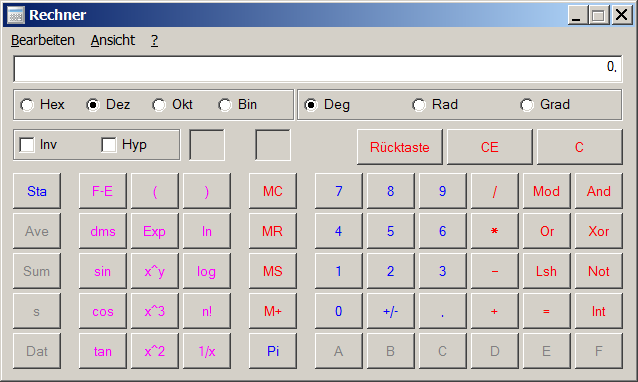
PowerShell weitere Beispiele

Mit Test-Path kann man leicht die Existenz eines Pfades überprüfen:

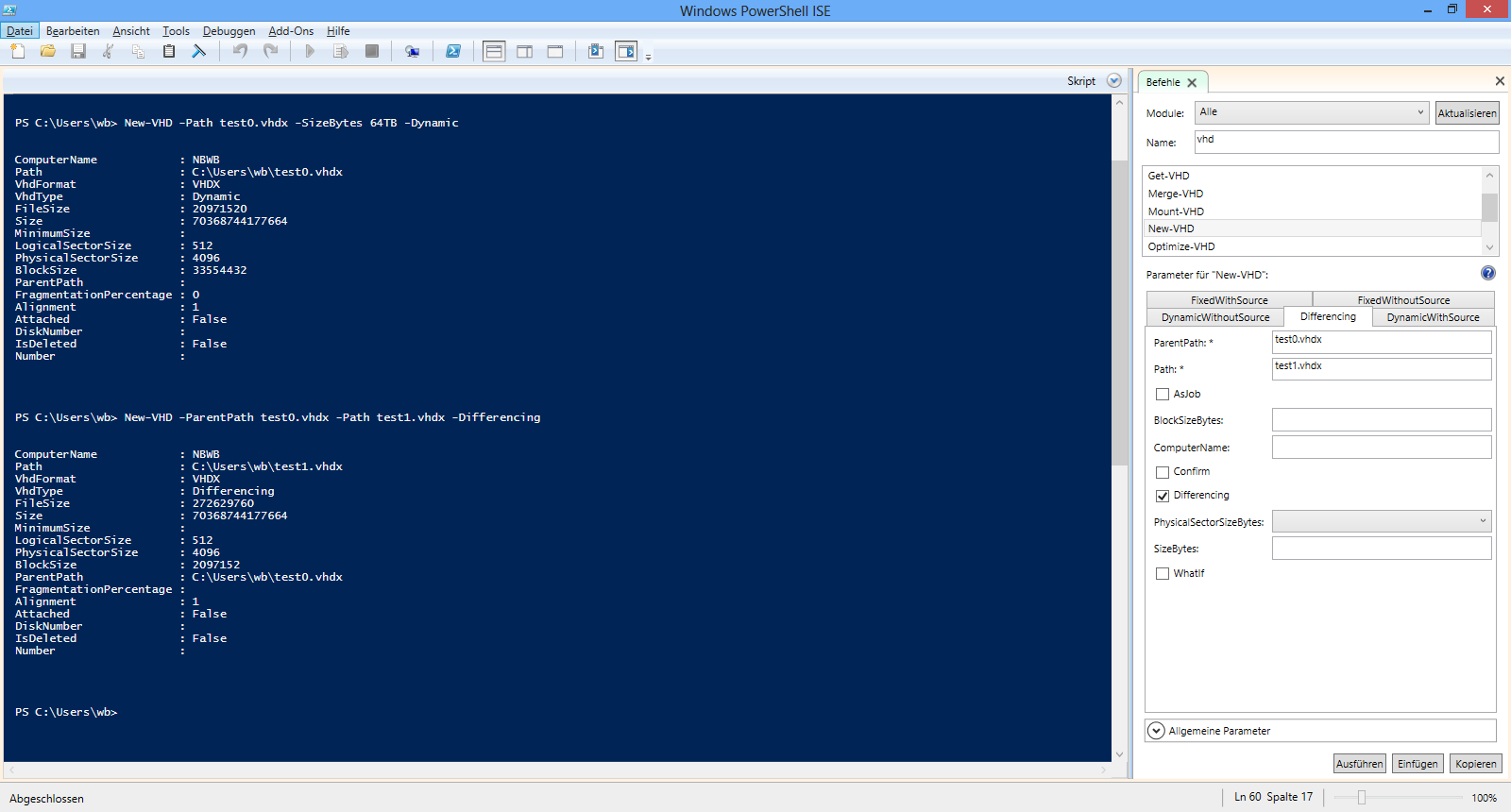


Mit Invoke-Item lassen sich jederzeit Programme starten..





Bei der Entwicklung komplexerer Powershell Befehle und Scripte steht auch die Powershell ISE



zur Verfügung. Sie kann einfach mit dem Befehl ise im Powershell Prompt gestartet werden.

update-help

Ausgabe aller Dateien (recursiv also einschliesslich aller unter Verzeichnisse) nach der Extension sortiert, incl. voller Pfadangabe:

dir -recurse | sort extension | select length,extension,fullname

Ausführliche (alle Attribute) Informationen zu Prozessen:

get-process | select \*

Nur Name und Pfad der Programmdatei der Prozesse:

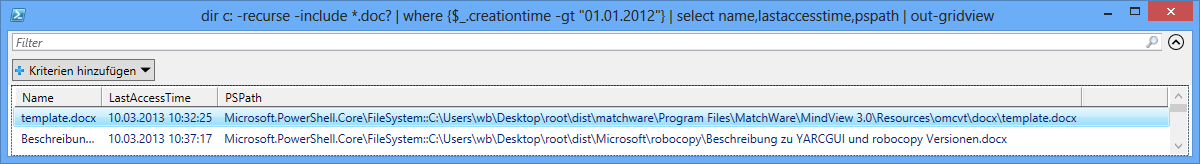
get-process | select name,path

Alle verfügbaren Attribute für Prozesseobjekte anzeigen:

get-process | get-member

Alle Word Dokumente finden, die nach dem 1.1.2012 erstellt wurden, Anzeige im out-gridview:

dir c: -recurse -include \*.doc? | where {$\_.creationtime -gt "01.01.2012"} | select pspath,name,lastaccesstime | out-gridview



Mit dem Befehl „Get-WmiObject“ gelangt man an alle Informationen aus der WMI (Windows Management Instrumentation) z.B.:

$b=GWMI Win32\_OperatingSystem  
$b.ConvertToDateTime($b.LastBootUpTime)

Starten von COM Applicationen z.B. Internet Explorer:

$i = new-object -ComObject „internetexplorer.application“  
$i.navigate(„www.heise.de/newsticker/“)  
$i.visible=1  
  
Oder auch Excel:

$e = new-object -ComObject „excel.application“  
$e.visible=1  
  
$e | get-member

Zeigt die dabei verfügbaren Methoden.

PowerShell Skripte

Powershell Skripte haben die Extension: \*.ps1

Es muss, wie auch unter Unix immer ein voll qualifizierter Pfad angegeben werden: ./xyz.ps1

Damit Skripte ausgeführt werden können müssen sie zuvor mit dem Cmdlet

Set-ExecutionPolicy RemoteSigned

zugelassen werden. Dies funktioniert nur in einer Powershell oder ISE mit administrativen Rechten oder mit deaktivierter UAC.

Es gibt folgende Werte für die Ausführungsrichtlinie der PowerShell Skripte:

**Restricted**: Eine Ausführung von Skripten und das Laden von Konfigurationsdateien ist nicht möglich (Standardeinstellung).

**AllSigned**: Sämtliche Skripte und Konfigurationsdateien müssen von einem vertrauenswürdigen Herausgeber signiert worden sein.

**RemoteSigned**: Aus dem Internet geladene Skripte und Konfigurationsdateien müssen signiert sein, lokal erstellte werden ohne Signatur ausgeführt.

**Unrestricted**: Alle Skripte und Konfigurationsdateien werden akzeptiert, allerdings wird vor der Ausführung von unsignierten aus dem Internet geladenen Dateien eine Warnmeldung angezeigt.

**Bypass**: Nichts wird blockiert, es erfolgen keine Warnungen.

Unser erstes Powershell Script (Datum.ps1):

$datum = get-date  
$datum.tostring("MM") # Monat  
$datum.tostring("dd") # Tag  
$datum.tostring("yyyy-MM-dd") # Jahr-Monat-Tag

Ausgeführt wird es mit ./Datum.ps1:

Ein Powershell Script zum umbenennen des PC Namen mit automatischem Reboot (hostname.ps1):

$Hostname="pc-franz" #

Rename-Computer -Computername localhost -NewName $Hostname -force

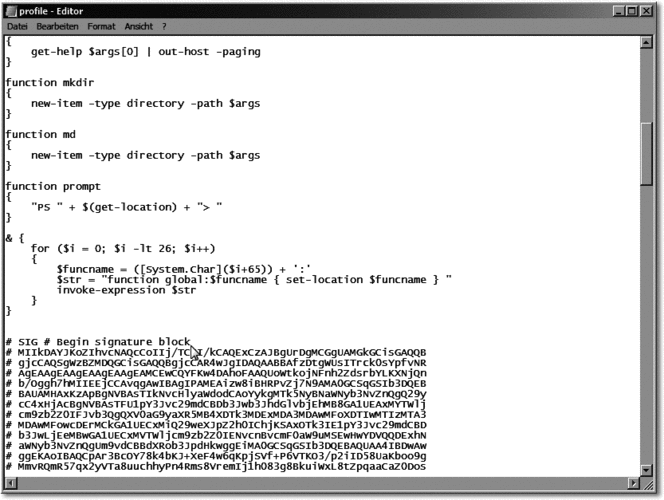
Restart-Computer

Ein einfaches Powershell Script zum versenden von Mail (mail.ps1):

$PSEmailServer="mail.bs2-landshut.de"

send-mailmessage -to weberhj@yahoo.com -from test@firma.de  
-subject "MailFromScript" -body "Der Messagetext."

Ein Beispiel für ein signiertes Powershell Skript:



Links zum Themengebiet Powershell:

http://blogs.msdn.com/PowerShell

http://www.powershell-ag.de/ps/Downloads/Skriptsammlung/tabid/76/Default.aspx