PowerShell

Die PowerShell-Exploration eines NICHT-Admins

- in die Konsolen-Niederung,
- zu den weiten Script-Feldern,
- auf die Remote-Fernsicht und
- zur .NET-Verwandtschaft.

Themen

- Mein Einstieg

- Konsole
- Script
- Remote
- .NET-Verwandtschaft

- Ende

Motivation

Mein Berührungspunkt zu PowerShell

Anweisungen zum ***Update von BEFOmobil***, der Befo „Windows App“

1. Starten:	Windows PowerShell.exe
2. Wechseln auf:	V:
3. Wechseln ins Install.-Verz.:	cd \Referenz\App
4. Wechseln ins Betriebssys.-Verz:	cd ARM (für Win RT 8.1), cd x86 (sonst)
5. Zwei Befehle ausführen:	import-module appx
	add-appxpackage BEFOmobil.appx

Ankündigung: „Das wird zukünftig mal von einem Script erledigt.“

PowerShell Quellen

- Der eigene Rechner (powershell_ise.exe, powershell.exe)
- Suchmaschinen: Das Internet ist voll von Tutorials, Blogs, Tipps , …!
- www.microsoft.com/PowerShell
- Wikipedia PowerShell (de)
- Wikipedia PowerShell (en)
- Frank Koch Windows PowerShell 3.0 (kostenloses eBook)
- PowerShell Scripting Guy, Ed Wilson
- Windows PowerShell Owner's Manual
- …

Was ist PowerShell? (Wikipedia)

- PowerShell ist eine Microsoft Alternative zur Windows Konsole CMD.EXE.
- Den Kern der PowerShell bilden kleine Funktionseinheiten, genannt **Cmdlets** (gesprochen command-lets), die dem Benennungsschema Verb-Substantiv folgen, also beispielsweise **Get-Help** oder **Set-Location**.

  Für die Bezeichnungen einiger PowerShell-Befehle (Cmdlets) sind vordefinierte **Alias-Namen** vergeben, die den Befehlen klassischer Shells entsprechen, unter anderem als Hilfe für **Umsteiger** von der CMD-Kommandozeile.

Was ist PowerShell? (Schwichtenberg)

- Dr. Holger Schwichtenberg
  (Windows PowerShell 4.0 Das Praxisbuch, 2014, 926 Seiten):

> Das DOS-ähnliche Kommandozeilenfenster hat viele Windows-Versionen in beinahe unveränderter Form überlebt. Mit der Windows PowerShell (WPS) besitzt Microsoft nun endlich einen Nachfolger, der es mit den Unix-Shells aufnehmen kann und diese in Hinblick auf Eleganz und Robustheit in einigen Punkten auch überbieten kann. Die PowerShell ist eine Adaption des Konzepts von Unix-Shells auf Windows unter Verwendung des .NET Frameworks und mit Anbindung an die Windows Management Instrumentation (WMI).

Schwichtenberg: „… Umgebung für interaktive Systemadministration …“

In einem Satz: Die Windows PowerShell (WPS) ist eine neue, .NET-basierte Umgebung für interaktive Systemadministration und Scripting auf der Windows-Plattform.

Die Kernfunktionen der PowerShell sind:

- Zahlreiche eingebaute Befehle, die „Commandlets“ genannt werden
- Zugang zu allen Systemobjekten, die durch COM-Bibliotheken, das .NET Framework und die Windows Management Instrumentation (WMI) bereitgestellt werden
- Robuster Datenaustausch zwischen Commandlets durch Pipelines basierend auf typisierten Objekten
- Ein einheitliches Navigationsparadigma für verschiedene Speicher (z.B. Dateisystem, Registrierungsdatenbank, Zertifikatsspeicher, Active Directory und Umgebungsvariablen)
- Eine einfach zu erlernende, aber mächtige Skriptsprache mit wahlweise schwacher oder starker Typisierung
- Ein Sicherheitsmodell, das die Ausführung unerwünschter Skripte unterbindet
- Integrierte Funktionen für Ablaufverfolgung und Debugging
- Die PowerShell kann um eigene Befehle erweitert werden.
- Die PowerShell kann in eigene Anwendungen integriert werden (Hosting).

PowerShell Schlagworte für meinen Einstieg

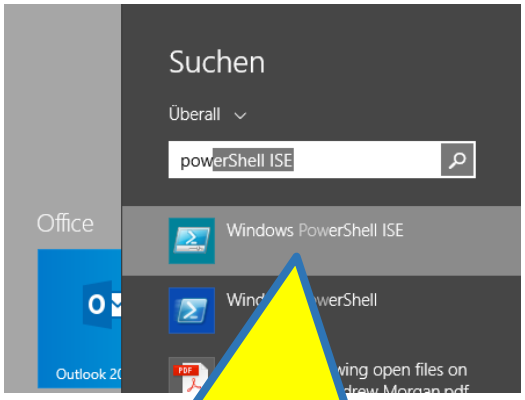
- Commandlets bzw. Cmdlets
- Alias-Namen
- Get-Help
- Objekte
- Pipeline
- einheitliches Navigationsparadigma
- mächtige Skriptsprache
- .Net Framework

… und „System-Administration“?

- PowerShell ist eine „Umgebung für interaktive Systemadministration“
- Ich bin kein Administrator
- Daher: PowerShell-Exploration eines NICHT-Admins
- Viele administrative Fragen, auf die PowerShell sicher eine Antwort hätte, stelle ich gar nicht und kann deswegen auch nichts dazu sagen.
- Meine ersten konkreten PowerShell Ziele sind:
  - Grundfunktionalitäten verstehen (s. o. „Schlagworte“)
  - Kleine alltägliche Registry-Manipulationen „automatisieren“
  - PS-Scripte für meine wenigen Standard-Batches (z.B. xBuild.bat)
  - PS-Script für das Update einer Win-App (s. „Motivation“)

… und … „Umsteiger“? Wo kommen wir her?

- Windows Konsole CMD.EXE
- Befehle, Funktionen:
  cd, cls, copy, del, dir, echo,
  md, move, popd, pushd,
  rd, type, <lw>:?
- Funktionieren diese Befehle in PowerShell?




```
E:
cd \engine\temp
cd \PsTmp
dir
dir *.ps1
```

Hinweis:
Zum Kennenlernen und Entwickeln „PowerShell ISE.exe“ statt „PowerShell.exe“ nutzen. Das bringt deutlich mehr User-Support. „PowerShell.exe“ ist für den operativen Einsatz wichtig (z.B. wg. der Aufrufparameter).
Nebenbei:
„ISE“ steht für ***Interactive Scripting Engine***

Grenzen der Cmd.exe Kompatibilität

```
dir *.
dir *.* /o-d
```

- CMD-Befehle in PowerShell sind offensichtlich nicht die CMD-Original-Befehle.
- Sie sind nicht 1:1 Parameter-Kompatibel.
- CMD-Befehle sind Alias-Benennungen von PowerShell Befehlen, den ***Cmdlets***.

- Welche Cmdlets stecken hinter den Alias-Benennungen?
- Wir kennen aus unseren Schlagworten das Cmdlet „Get-Help“

Cmdlets Vol1

- `Get-Help`

  „Windows PowerShell enthält keine Hilfedateien, Sie können die Hilfethemen jedoch online lesen. Oder laden Sie mit dem Cmdlet "**Update-Help**" Hilfedateien auf den Computer herunter, und zeigen Sie die Hilfethemen anschließend mit dem Cmdlet "Get-Help" in der Befehlszeile an.“
- `Update-Help`
- `Get-Help –Name alias`

  `VERWANDTE LINKS`

  `about_Aliases`

  (hinter about_... steht eine ***PS-Konzept***)
- `Get-Help Get-Alias`

  Gets the aliases for the current session.
- `Get-Alias`

  Liefert die Liste der Cmdlets, die hinter den uns bekannten Aliases stecken.

Hinweis:
Für den Aufruf von „Update-Help“ PowerShell im Admin-Modus starten.

Nebenbei:
Auf diesen Befehl bin ich sehr früh gestoßen. Evtl. hatte ich deswegen nie Probleme mit der PowerShell Hilfe.

Cmdlets Vol2

- `Get-Alias -Name cd`

```
CommandType     Name
-----------     ----
Alias           cd -> Set-Location
```

Mehrere Befehle – z.B. zur besseren Übersicht – in eine Zeile schreiben mit „;“

- `Get-Alias Cd; Get-Alias Dir`
- `cd -> Set-Location`
- `dir -> Get-ChildItem`

Infos zu Set-Location und Get-ChildItem selbstverständlich mit:

- `Get-Help Set-Location`
- `Get-Help Get-ChildItem`
- `Get-Help Set-Location -full`
- `Get-Help Set-Location –examples`

Cmdlets – Common Parameters

Parameter, die jedes Cmdlet unterstützt (in Klammern die Kurzbezeichnungen).

- -Debug (db)
- -ErrorAction (ea)
- -ErrorVariable (ev)
- -OutVariable (ov)
- -OutBuffer (ob)
- -PipelineVariable (pv)
- -Verbose (vb)
- -WarningAction (wa)
- -WarningVariable (wv)

Risiko minimierende “Common Parameters”:

- -WhatIf (wi)
- -Confirm (cf)

Infos dazu mit: `Get-Help about_CommonParameters`

Cmdlets Vol3

Wir wenden unser Wissen an (Cmdlets statt Aliases):

```
"";"Liste";Set-Location e:\engine\ctrl;Get-ChildItem *.exe

Liste

    Verzeichnis: E:\engine\ctrl

Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---         06.06.2001     02:20      39936 FileDate.exe
-a---         05.05.2011     09:58     121856 HoboCopy.exe
-a---         13.11.2013     22:22    1058199 RawCopy64.exe
-a---         20.11.2010     05:25     128000 RobocopyXP027.exe
```

Cmdlets Vol4

Cmdlets, die uns bekannte Aliase realisieren:

Cd -> Set-Location
Cls -> Clear-Host
Copy -> Copy-Item
Del -> Remove-Item
Dir -> Get-ChildItem
Echo -> Write-Output
Md -> New-Item
Move -> Move-Item
Popd -> Pop-Location
Pushd -> Push-Location
Ren -> Rename-Item
Set -> Set-Variable
Type -> Get-Content

Was sind Cmdlets?

- Cmdlets (gesprochen command-lets) sind die Funktionseinheiten von PowerShell.
- Cmdlets folgen dem Benennungsschema Verb-Substantiv, bspw.
  - **`Get-Help`** oder
  - **`Set-Location`**
  - Liste der „offiziellen Verben“ mit **`Get-Verb`**
- PowerShell 4.0 enthält 538 Cmdlets, 712 Functions und 150 Aliase.
- Cmdlets liefern Objekte zurück.
- Cmdlets können mit der Pipe „|“ verknüpft werden.
- Cmdlets sind als .Net-Klassen implementiert.

Objekte und Rückgabe Objekte

Was liefert Get-ChildItem tatsächlich zurück?

Das Cmdlet, um PS-Objekte zu untersuchen, heißt:

```
Get-Member
```

(Aber Get-Member will ein Objekt haben.)

„Get-Member : Sie müssen ein Objekt für das Cmdlet "Get-Member" angegeben. “

Es lohnt sich die Fehlermeldungen zu lesen

Alles ist ein Objekt

- `Get-Member -InputObject "Michael"`
  `TypeName: System.String`
- `"Michael".Length`
  `7`

Objekt per „Pipe“ an Cmdlet weiterreichen

- `"Michael" | Get-Member`

- `Get-ChildItem | Get-Member`
  `TypeName: System.IO.DirectoryInfo`
  `TypeName: System.IO.FileInfo`

Objects

Was kann man mit Objekten alles machen? Außer Nutzung der eigenen Methoden.

```
get-help object
```

liefert Liste spezieller Object-Cmdlets (und es gibt noch weitere Cmdlets).

Name	Category	Synopsis
ForEach-Object	Cmdlet	Performs an operation against each item in a c...
Where-Object	Cmdlet	Selects objects from a collection based on the...
Compare-Object	Cmdlet	Compares two sets of objects.
Group-Object	Cmdlet	Groups objects that contain the same value for...
Measure-Object	Cmdlet	Calculates the numeric properties of objects, ...
New-Object	Cmdlet	Creates an instance of a Microsoft .NET Framew...
Register-ObjectEvent	Cmdlet	Subscribes to the events that are generated by...
Select-Object	Cmdlet	Selects objects or object properties.
Sort-Object	Cmdlet	Sorts objects by property values.
Tee-Object	Cmdlet	Saves command output in a file or variable and...
Get-WmiObject	Cmdlet	Ruft Instanzen von Klassen der Windows-Verwalt...
Remove-WmiObject	Cmdlet	Löscht eine Instanz einer vorhandenen Klasse d...
about_Objects	HelpFile	Provides essential information about objects i...
about_Object_Creation	HelpFile	Explains how to create objects in Windows Powe...

Objects und Pipeline

Welche Werte liefert das Property „Attributes“ des Get-ChildItem-Objekts?

```
Get-ChildItem | ForEach-Object {$_.attributes}
Directory
Archive
```

`|`	Objekt „weiterleiten“ zum nächsten Befehl
`{}`	ist die Syntax für einen Codeblock.
`$_`	ist die Syntax für den Zugriff auf ein Item eines Objects.
`attributes`	das Property findet man mit Get-Item|Get-Member

Nebenbei, es gibt viele Abkürzungen. Beispiel:

```
gci | ForEach {$_.attributes}
```

`$_` ist die Abkürzung für **`$PSItem`**

Objects und Pipeline

An **„`Dir *.`“** waren wir eben gescheitert. Nachbildung mit **`Get-ChildItem`**:

Objekt „weiterleiten“ zum nächsten Befehl mit Pipe „|“

```
Get-ChildItem | Where-Object {$_.attributes -match "Directory"}
```

`|`	Objekt „weiterleiten“ zum nächsten Befehl
`{}`	ist die Syntax für einen Codeblock.
`$_`	ist die Syntax für den Zugriff auf ein Item eines Objects.
`attributes`	das Property findet man mit Get-ChildItem|Get-Member
`-match`	den Parameter findet man mit Get-Help|Where-Object
`"Directory"`	mit „Get-ChildItem|ForEach-Object {$_.attributes}“ ermittelt

Cmdlets Vol5 „alles ist in Laufwerken“

```
Get-PSDrive
```

- `Get-ChildItem alias:c*`
- `Get-ChildItem Env:`
- `Get-ChildItem Env:path`
- `(Get-ChildItem Env:path).value`
- …

Zugriffe auf alles:
Files, Environment, Registry (HKLM u. HKCU), Cert, Functions, Variablen, …

Alles ist ein „Laufwerk“!
Das ist wohl das

„einheitliche Navigationsparadigma“

von Schwichtenberg s.o.

Dateien anzeigen, öffnen

```
Cd Script
Get-Content -Path Greet-World.ps1
Cd ..
Get-Content x.csv
Get-Content $env:userprofile\ineta.xls*
Invoke-Item $env:userprofile\ineta.xls*
```

Der komplette Pfad (oder „.\“, falls die Datei im akt. Verz. steht) wird für den ausführenden Aufruf benötigt, das erfordert das PS-Sicherheitskonzept. Es sein denn, die ausführbare Datei steht im Suchpfad.

```
Start-Process -FilePath notepad -ArgumentList greet-world.ps1
Get-Process -Name notepad
Stop-Process -Name notepad -WhatIf
Stop-Process -Name notepad
Stop-Process -Name notepad
Stop-Process -Name notepad -ErrorAction SilentlyContinue
```

Vor der Skriptverarbeitung nebenbei noch folgende Frage: Welche PowerShell Version läuft eigentlich gerade?

- `Get-Host`
- `$PSVersionTable` `# scheint richtiger`

PowerShell – Exploration

P A U S E

https://github.com/MichaelPae/PS-Pieces

PowerShell- Exploration: Was bisher geschah.

IN EIGENER SACHE: Ihr seht meine schmale Sicht.
PowerShell kann viel mehr!

- PowerShell ist eine Konsolen-Umgebung für interaktive Systemadministration und Scripting auf WIN-Plattformen.
- Die Funktionseinheiten von PowerShell sind Cmdlets (gesprochen command-lets). Sie folgen dem Benennungsschema Verb-Substantiv. Z. Bsp.:
  - **`Get-Help`** oder
  - **`Set-Location`**
- **`Get-Help`** ist der zentrale Hilfebefehl. Neben der Hilfe zu den Cmdlets gibt es auch Hilfen zu den PowerShell-Konzepten (z. B. `about_CommonParameters`)
- Uns bekannte Befehle aus CMD.EXE sind nichts anderes als Aliase von Cmdlets.
- Cmdlets liefern in der Regel Objekte zurück, deren Properties und Methoden man nutzen kann (s. **`Get-Member`**)
- Zusätzlich gibt es eine Reihen von Cmdlets, die speziell die Objektverarbeitung unterstützen (z. B. **`ForEach-Object`**, **`Where-Object`**) und das Pipe-Symbol „|“ zur Verknüpfung von Objekten.
- Das „einheitliche Navigationsparadigma“ von PowerShell heißt: „alles ist in Laufwerken“ (s. **`Get-PSDrive`**)

Scripte – about Execution Policies

PowerShell_ISE.exe

- starten und erläutern
- Ansicht -> Symbolleiste, Skriptbereich, Befehls Add-On
- "HalloWelt" # Ausführen liefert:
  Hallo Welt
- Als Script Unbenannt1.ps1 speichern und ausführen, liefert:

```
PS E:\PsTmp> E:\PsTmp\Unbenannt1.ps1
Die Datei "E:\PsTmp\Unbenannt1.ps1" kann nicht geladen werden, da die Ausführung von
Skripts auf diesem System deaktiviert ist. Weitere Informationen finden Sie unter
"about_Execution_Policies" (http://go.microsoft.com/fwlink/?LinkID=135170).
    + CategoryInfo          : Sicherheitsfehler: (:) [], ParentContainsErrorRecordExcept
   ion
    + FullyQualifiedErrorId : UnauthorizedAccess
```

- On the fly in zweitem PS-Fenster im Admin-Modus:
  `Set-ExecutionPolicy -ExecutionPolicy RemoteSigned`

Damit ist die Skriptverarbeitung freigeschaltet.

Scripte – Hallo Welt!

Hinweis:
Störende Variablen mit „Remove-Variable“ löschen.
Demo: 1. Start von Konsole
2. Start Script mit <F5>

Infos (Write-HalloWelt.ps1)

```
# PowerShell Beispiel Script
# "#"             Inline Kommentar
# <# ... #>       Blockkommentar
# `               Code-Zeilen-Umbruch mit dem „Gravis“-Zeichen
# ;               Befehle in einer Zeile trennen
# (...)           Runde Klammern haben algebraische Bedeutung:
#                 Sie teilen der Shell mit, was zuerst ausgeführt wird.
# {...}           Geschweifte Klammern schließen Codeblöcke ein
# $               Variablen-Kennzeichen
# $true, $false   Boolsche Konstanten als Pseudo-Variablen
[String]$HalloWelt = "Hallo Welt"     # Typisierung ist nicht zwingend
Write-Host ""; Write-Host $HalloWelt  # in der ISE sind die Var. nach Ausf. <F5> noch da
Write-Host "mit         ", ("Powershell-Version"+" " + $PSVersionTable.psversion).ToString()
Write-Host "auf PC      ", (Get-Content Env:Computername)
Write-Host "mit CPU-Typ ", (`
    Get-ItemProperty -path `
    "HKLM:\SYSTEM\CurrentControlSet\Control\Session Manager\Environment"`
    ).PROCESSOR_ARCHITECTURE
<#
    # alternativer Code-Vorschlag
    Write-Host "mit CPU-Typ ", (Get-Content Env:Processor_Architecture)
#>

if (-not$false) { # Bool-Operatoren  -not ! -eq, -nq, ...
    Write-Host "von User    ", (Get-Content Env:Username)
} else {
    "Diese Stelle wird nie erreicht"
}
```

Scripte – Parameter / Eingaben

dir *.* /o-d -> DirO-D (DirO-D.ps1)

```
param([Parameter(Mandatory=$true)][String]$Pattern, `
 [Int]$Anzahl=10)
Get-ChildItem $Pattern | `
 Sort-Object LastWriteTime -Desc | `
 Select-Object -first $Anzahl
```

Parameter-Übergabe-Optionen:

- zwingend, [Parameter(Mandatory=$true)]
- typisiert, [String]$Pattern
- mit Default-Wert, $Anzahl=10

Weitere „Eingaben“ neben Parametern

- Viele Powershell Cmdlets holen Daten ran (s.o. Get-ChildItem)
- `Get-Content -Path <file>`

Scripte – Ausgaben

Viele Möglichkeiten … nur einige Beispiele

```
Write-Host
Write-Output
Out-File
Export-Csv
Export-Clixml
```

Fortlaufendes Log (Write-MeinLog.ps1)

```
$Log = "Mein.log"
# Mal mit und Mal ohne -Force
new-item -path . -name $Log -type "file" –value "1. Zeile`r`n" -Force
"nächste Zeile" | add-content -Path $Log
get-content -Path $Log
```

Scripte – erste Scripte

Registry manipulieren (Set-BefoDdDir_LOKAL.ps1)

- `Get-Process -name $ProzessName -ErrorAction SilentlyContinue`
- `Stop-Process -name $ProzessName`
- `Start-Process -FilePath C:\befo2\run\bConsole\BCONSOLE.EXE -ArgumentList "-Unn -Pnn"`
- `Set-ItemProperty -path "HKLM:\SOFTWARE\BEFO II\Dictionary" -name Verzeichnis -value $Verzeichnis`
- `Start-Process PowerShell -Verb runAs -ArgumentList $arguments`

Xbase Build Aufruf (Build-Xpp.ps1)

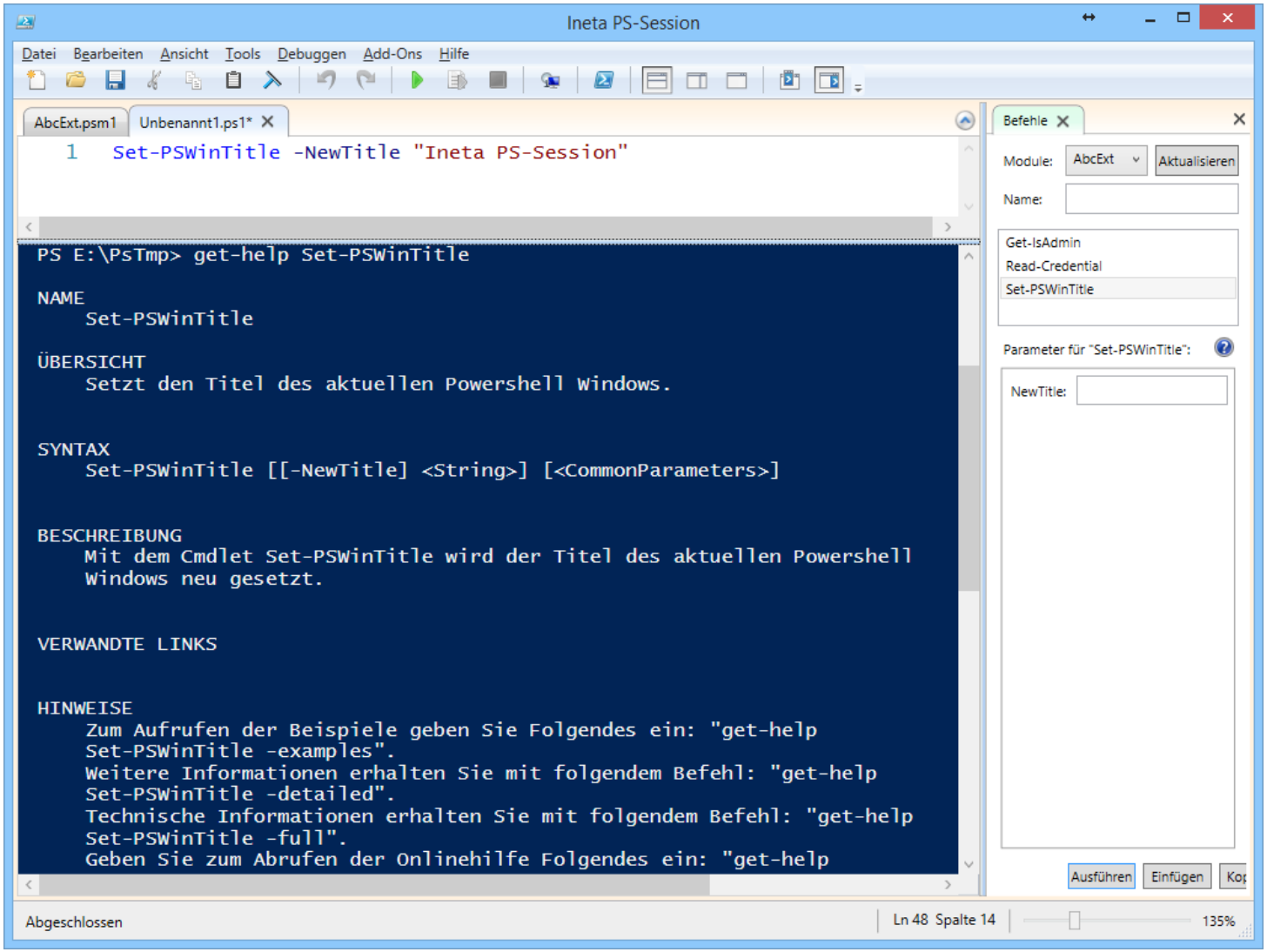
- `Start-Process -FilePath $Path -ArgumentList $Args -NoNewWindow -Wait`
- `new-item . -name $Log -type "file" -value ("Start "+$Script +"`r`n") -force | Out-Null`
- `$LogLine | Add-Content $Log`

Windows-App updaten (Update-Befomobil.ps1)

- `Export-Clixml # Serialisieren, Deserialisieren`
- `Import-Clixml # Properties bleiben erhalten, Methoden nicht.`
- `New-Item -Path $RootDir -Name $SubDir -ItemType directory | Out-Null`
- `Start-Sleep -Seconds 5`
-
```
$Diff = Compare-Object -ReferenceObject $Update `
-DifferenceObject $LastUpdate -property LastWriteTime, Length, Name -PassThru | Where-Object {
$_.SideIndicator -eq '=>' }
```
- `Import-Certificate -Filepath .\BEFOmobil.cer -CertStoreLocation cert:\LocalMachine\AuthRoot`
- `Import-Module`
- `if ((get-content -Path $AppxLog | Measure-Object -Line).Lines -eq 1)`

Dieses Script zu erstellen, war die anfängliche Motivation!

Scripte – Eigene Module (Script-Library)



Scripte – Eigene Module

Implementierung eines Script-Moduls (AbcExt.psm1)

```
AbcExt.psm1                                          (Cmdlet-Code in .psm1-Files strukturieren)
New-ModuleManifest -path .\AbcExt.psd1 `             (.psd1 ist notwendig)
 -CompanyName „Ineta DevGroup GöKs" `
 -Copyright '(c) 2015 Michael Pätzold. Alle Rechte vorbehalten.'
Import-Module -Name .\AbcExt.psm1                    (lädt das Modul)
```

(danach in ISE: Befehle : Aktualisieren : Modul-Auswahl : Set-PSWinTitle : <?>)

Anwendung

```
Set-PSWinTitle –NewTitle "Ineta PS-Session"     (Cmdlet verwenden)
Remove-Module –Name AbcExt                      (entlädt das Modul)
Set-PSWinTitle –NewTitle "Ineta PS-Session "    (liefert Fehler)
```

Script COMMENT BASED HELP

```
get-help about_comment_based_help
```

Was sind Cmdlets? – Die Zweite

- Cmdlets folgen dem Benennungsschema Verb-Substantiv, bspw.
  - **`Get-Help`** oder
  - **`Set-Location`**
  - Liste der „offiziellen Verben“ mit **`Get-Verb`**
- Cmdlets liefern Objekte zurück.
- Cmdlets müssen die Pipe „|“ unterstützen
- Cmdlets unterstützen die „Comment-Based-Help-Syntax“
- Cmdlets verwenden die Standard-Codeblöcke:

```
Function Test-Demo {
    Param (...)
    Begin{ ... }
    Process{ ... }
    End{...}
}
```

- Cmdlets müssen … (… sicherlich noch weitere Eigenschaften haben)

Scripte – Modules & Snap-Ins

Die Funktionalität der PowerShell lässt sich durch sogenannte *Snap-Ins* erweitern, welche auf einen Schlag ganze Sätze von zusätzlichen Cmdlets importieren und dem Benutzer zur Verfügung stellen.

Bsp. Microsoft.PowerShell.Core

> Dieses Windows PowerShell-Snap-In enthält Cmdlets zum Verwalten von Windows PowerShell-Komponenten.

Modules vs. **Snap-Ins** (aus der **PowerShell Hilfe**)

“You can add commands to your session from modules and snap-ins. Modules can add all types of commands, including cmdlets, providers, and functions, and items, such as variables, aliases, and Windows PowerShell drives. Snap-ins can add only cmdlets and providers.”

Remote – Voraussetzungen

Remote-Verbindung in einer Domain, Workgroup

Voraussetzung PowerShell auf dem Host im Administrator-Modus:

```
Enable-PSRemoting -Force
Restart-Service WinRM
```

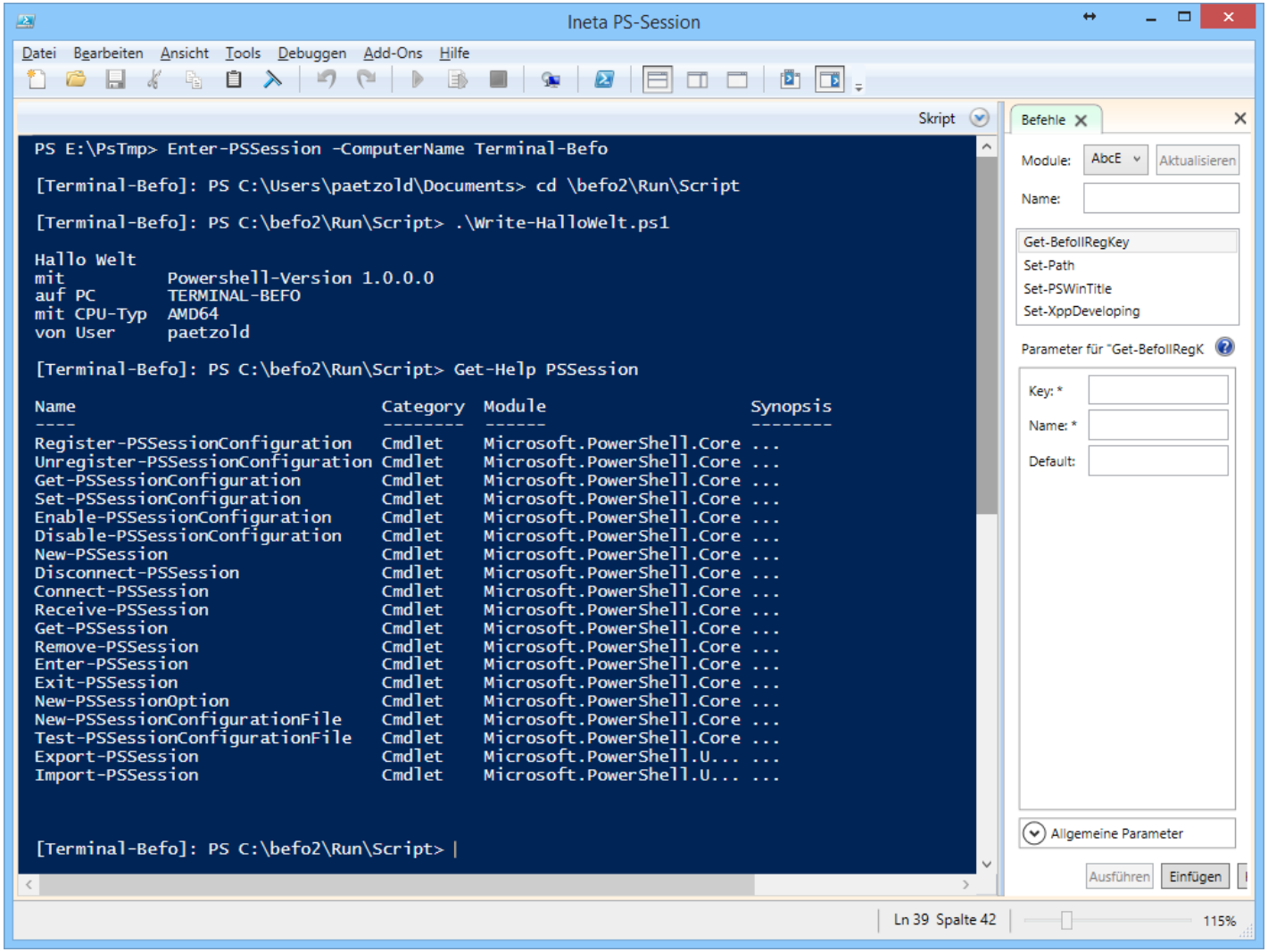
Voraussetzung PowerShell auf dem Domain-Client im Administrator-Modus:

```
Test-WSMan <RemotePcName> # muss ohne Fehler laufen
# nur falls Domain-Clients auf nicht Domain-PC zu verbinden sind:
Set-Item WSMan:\localhost\Client\TrustedHosts -value <RemotePcName>
```

Danach geht vom Client

```
Invoke-Command -Computername <Name> `
   -ScriptBlock { Get-ChildItem C:\ } -Credential <User>
Enter-PSSession -Computername <Name> -Credential <User>
```

Remote – PSSession



Remote – im Script

Remote-Verbindungen automatisieren

- Test-Connection -ComputerName <Name> -Quiet -Count 1 liefert Bool
- Write-SecureString.ps1 EVA
- Read-Credential.ps1
- Invoke-CmdIE11WIN7.ps1 Invoke-Command
- Check-MyProcess.ps1 Session-Variable, Variable in Scriptblock
- Start-PsSession.ps1 Enter-PSSession

WICHTIG:

Innerhalb einer Domain ist keine „Anmeldung“ notwendig.

PS .NET – Verwandtschaft (mit schwarzem Schaf)

PowerShell ist .NET basiert.

PowerShell .NET Beispiel mit Magie (Use-NetFrameWork.ps1)

```
# Bsp.-Aufgabe: Log-Filenamen erzeugen
#
#pures PowerShell
$Script1 = $myInvocation.MyCommand.Name
$Log1 = $Script1.BaseName + ".log"
Write-Host "Script-Name: "$Script1.ToString()
Write-Host "   Log-Name: "$Log1.ToString()

# .Net-Klasse in PowerShell:
$Script2 = $myInvocation.MyCommand.Name
$Log2 = ([system.io.fileinfo]$Script2).BaseName + ".log"
Write-Host "Script-Name: "$Script2.ToString()
Write-Host "   Log-Name: "$Log2.ToString()
```

PS .NET – Type Conversion Magic

Here are the steps that PowerShell takes on your behalf to convert input to a given type. As with many things, there is no magic - just a lot of hard work.

- **Direct assignment.** If your input is directly assignable, simply cast your input to that type.
- **Language-based conversion.** These language-based conversions are done when the target type is void, Boolean, String, Array, Hashtable, PSReference (i.e.: [ref]), XmlDocument (i.e.: [xml]). Delegate (to support ScriptBlock to Delegate conversions), and Enum.
- **Parse conversion.** If the target type defines a Parse() method that takes that input, use that.
- **Static Create conversion.** If the target type defines a static ::Create() method that takes that input, use that.
- **Constructor conversion.** If the target type defines a constructor that takes your input, use that.
- **Cast conversion.** If the target type defines a implicit or explicit cast operator from the source type, use that. If the source type defines an implicit or explicit cast operator to the target type, use that.
- **IConvertible conversion.** If the *source type* defines an IConvertible implementation that knows how to convert to the target type, use that.
- **IDictionary conversion.** If the source type is an IDictionary (i.e.: Hashtable), try to create an instance of the destination type using its default constructor, and then use the names and values in the IDictionary to set properties on the source object.
- **PSObject property conversion.** If the source type is a PSObject, try to create an instance of the destination type using its default constructor, and then use the property names and values in the PSObject to set properties on the source object. If a name maps to a method instead of a property, invoke that method with the value as its argument.
- **TypeConverter conversion.** If there is a registered TypeConverter or PSTypeConverter that can handle the conversion, do that. You can register a TypeConverter through a types.ps1xml file (see: $pshome\Types.ps1xml), or through Update-TypeData.

PS .NET – weitere Beispiele

LAN-Broadcast (Send-WakeOnLanToMacAddress.ps1)


```
$packet = ConvertTo-MagicPacket -Mac '74:D4:35:B2:D3:5F'

# .NET Framework User Datagram Protocol-Netzwerkdienst
$UDPclient = new-Object System.Net.Sockets.UdpClient
$UDPclient.Connect(([System.Net.IPAddress]::Broadcast),4000)
$UDPclient.Send($packet, $packet.Length) | out-null
```

FTP-Verbindungen (Get-FtpListDir.ps1)

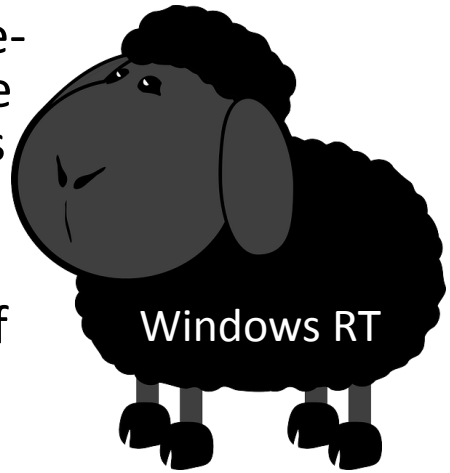
```
$request = [System.Net.FtpWebRequest]::Create($Path)
...
$request.Method = [System.Net.WebRequestMethods+FTP]::ListDirectoryDetails
$response = $request.GetResponse()
```

Nebenbei: Web-Verbindungen (direkt in PowerShell implementiert)

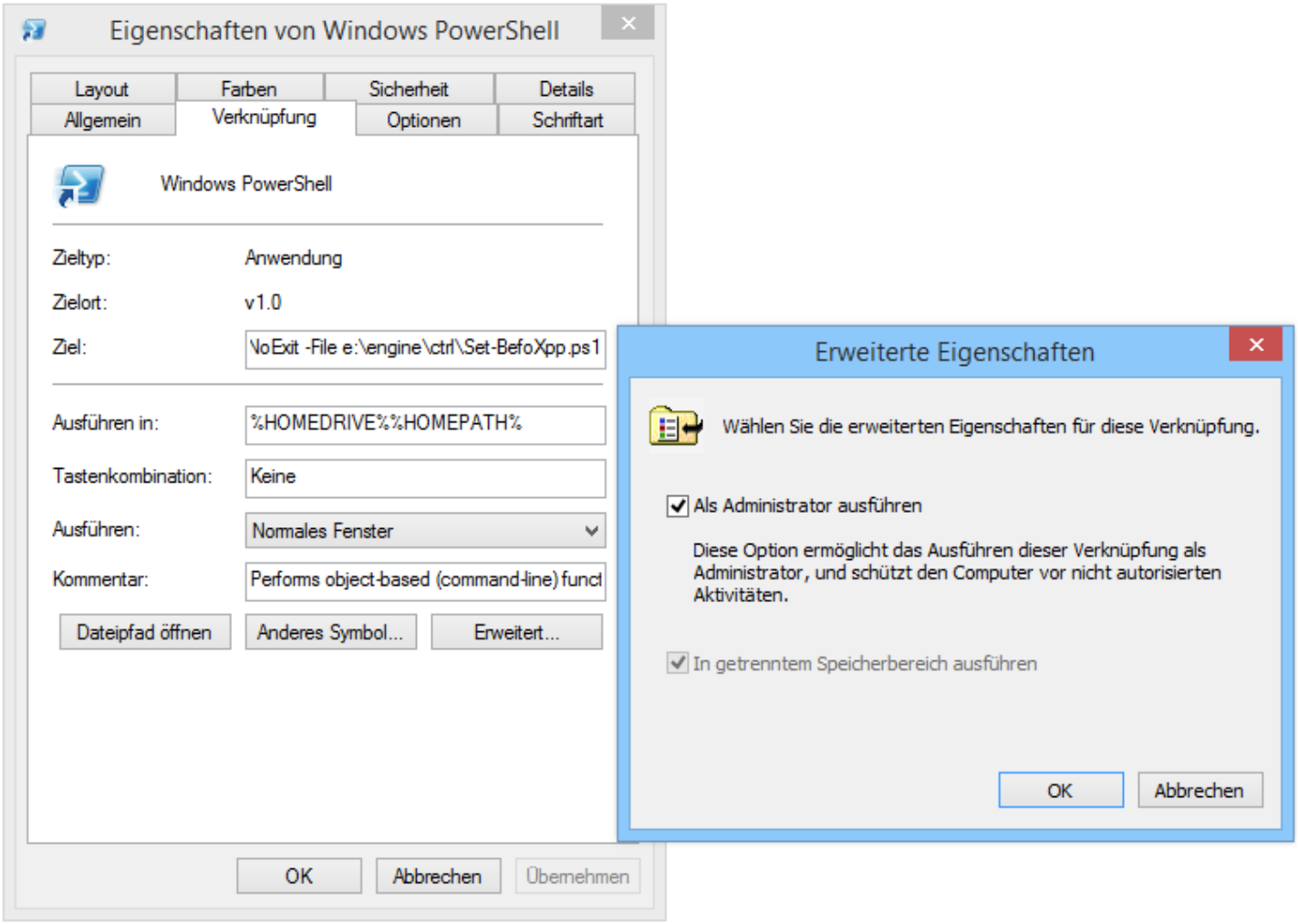
```
Invoke-WebRequest #  Gets content from a web page on the Internet.
```

PS .NET – … und das schwarze Schaf der Familie

- **Plattform-Unterschiede**:
  Konkrete Erfahrungen mit unserem Application-Update-Skript „Update-BEFOmobil.ps1“, das am Windows Store vorbei Windows Apps installiert, haben gezeigt, dass es bei PowerShell Plattform-Unterschiede gibt.
- **.NET-Statements**:
  Scripte, die .NET-Klassen direkt nutzen, laufen nicht auf Windows RT. Dort werfen sie Fehler.
- **Erkenntnisgewinn** nach Recherche:
  „Microsoft lässt aktuell für Windows RT-PowerShell (PowerShell 4.0) den direkten Zugriff auf .NET-Methoden nicht zu.“
- **Notwendigkeit**:
  PowerShell-Skripte, die auf allen Windows 8 Plattformen laufen sollen, müssen sich auf „pure PowerShell“ Elemente beschränken.



PowerShell – An die Eigenschaften denken …



PowerShell – Ende meiner Exploration

Material zu dieser Session unter

https://github.com/MichaelPae/PS-Pieces

Vielen Dank für Eure Aufmerksamkeit!