PowerShell for Beginners

Basics and Complex Exercises

Table of Contents

PowerShell Server Tasks

Modules, overview Active Directory functions, updates

PowerShell Modules

- Modules are collections (bundles) of Cmdlets
- Modul development and integration is continuous:
  - Windows 7: ca. 10 Module
  - Windows 10: ca. 70 Module
  - Windows Server 2016 (without AD): ca. 80 Module
- Exchange Server, SharePoint Server have their own PowerShell-based administration shell

PowerShell Modules

- Various storage locations on the computer (OS-dependent)
- Displaying locations for a computer:

```
PS C:\Users\anr> $Env:PSModulePath -split ";"
C:\Users\anr\Documents\PowerShell\Modules
C:\Program Files\PowerShell\Modules
c:\program files\powershell\7\Modules
C:\Program Files\WindowsPowerShell\Modules
C:\windows\system32\WindowsPowerShell\v1.0\Modules
```

- Module management with Cmdlets from the `Module` family
- `Get-Module -ListAvailable` prints loaded and available modules
- `Get-Command -Module <MOD>` lists commands contained in module `<MOD>`

Module List Windows Server 2016
(excerpt; as DC with DNS Server)

- ActiveDirectory*
- ADDSDeployment*
- AppLocker
- BestPractices*
- BranchCache
- DnsClient
- DnsServer*
- GroupPolicy*
- NetTCPIP
- PKI
- PrintManagement
- RemoteDesktop*
- ScheduledTasks
- ServerCore*
- ServerManager*
- VpnClient

Loading Modules dynamically

- By using a Cmdlets the module that contains it is dynamically and automatically loaded. This is called implicit loading.

- Hands-on:
  Implicitly load the module *DnsClient* using Cmdlet *Resolve-DnsName*
  - 1. List active modules
  - 2. Resolve domain name www.cisco.com
  - 3. List active module again

Loading Modules manually

- Explicit manual loading with `Import-Module`
- Normally not necessary, but…
  - often integrated into logon scripts etc.
  - **saves time during script execution**
  - initial loading of PowerShell will take slightly longer

The Module ServerManager

- Is the CLI counterpart of the Server Manager GUI
- Allows for installation and management of roles and features

Cmdlet	Task
`Get-WindowsFeature`	List roles and features
`Add-WindowsFeature` `Install-WindowsFeature`	Install new role or new feature
`Remove-WindowsFeature` `Uninstall-WindowsFeature`	Uninstall a previously installed role of feature

Installation of Roles and Features

- Parameter for the Cmdlets in charge
  - correspond to input in Server Manager GUI wizard

Parameter	Meaning
`-Name`	Name of the role or feature
`-ComputerName`	Target computer for the installation
`-IncludeAllSubFeatures`	Install all dependent services/features as well
`-IncludeManagementTools`	Also install administration shell. Not included as default like with a GUI lead installation!
`-Restart`	Restart computer during the installation, if necessary

Installation von Rollen und Features

- Hands-on:
  List all currently installed roles and features

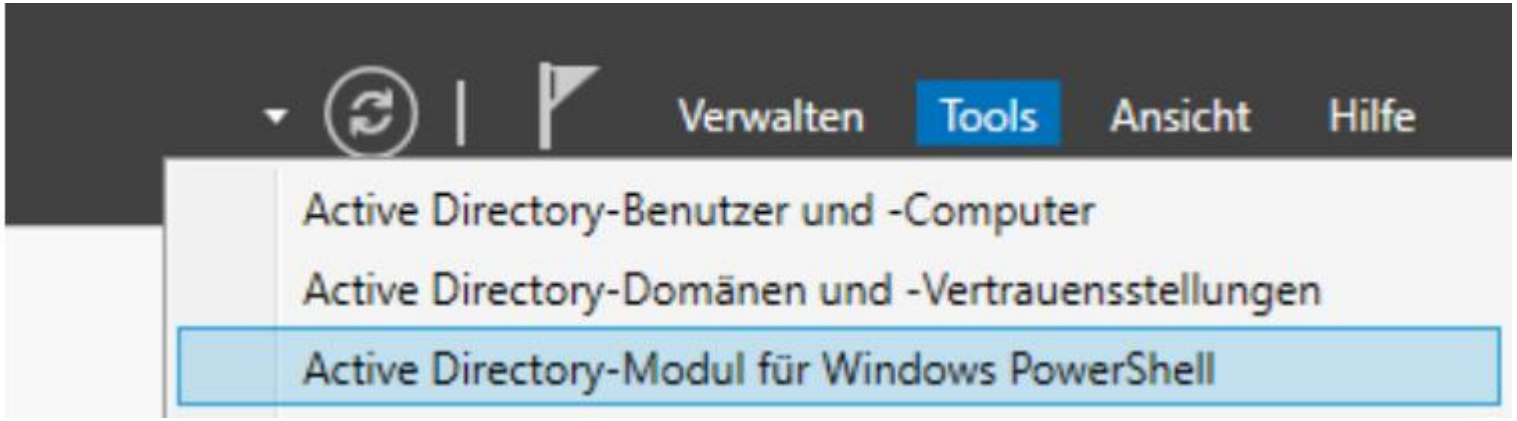
- Hands-on:
  Install Microsoft Webserver IIS with all dependent services on computer *WS01*.

The Module ActiveDirectory

- *ActiveDirectory* allows for management of users and groups (organizational units [OU]) in a domain.
- One of the most powerful modules for Windows Server.
- Contains approx. 150 own Cmdlets.

ActiveDirectory Shell

- Active Directory comes with its own administration shell



- Background is always black (instead of blue like in PowerShell versionen <=5)
- Alias for
```
%windir%\system32\WindowsPowerShell\v1.0\powershell.exe -NoExit
-Command Import-Module ActiveDirectory
```

ActiveDirectory Management

- ActiveDirectory establishes PSDrive AD:
- Navigating the drive with distinguished names (DN) as paths for *Set-Location*
- User management in AD
  - Family `LocalUser`
  - Family `LocalGroup`
  - Family `LocalGroupMember`

ActiveDirectory advanced Stuff

- Complete administration of domains possible with *ActiveDirectory* module
- Administration of OUs
- Protection against accidential deletion
- Creation, configuration and inheritence of group policies
- Integration of domains into forests
- Configuration of trust relationships
- Display and move FSMO roles
  - `Get-ADForest | Format-List -Property <FSMO-ROLE>`

Windows Updates

- Server(-Cluster) must be updated on a regular basis.
- PowerShell offers (since Windows Server 2016) the module *WindowsUpdateProvider*
- Provider is Windows Update (default) or a server with the Windows Server Update Services (WSUS) role.
  - Windows Update manages updates automatically
  - WSUS allows update control in a (company) network
- Functions in module work with both providers

Searching for Updates

- Cmdlet `Start-WUScan`
- Limiting updates via `-SearchCriteria <CONDITION_STRING>`
- Common combination: `"IsInstalled=0 AND IsHidden=0 AND IsAssigned=1“`
  - `IsInstalled=0` → update not yet installed
  - `IsHidden=0` → updates is publicly accessible
  - `IsAssigned=1` → update is assigned for the platform

Installation of Updates

- Installation through predefined provider.
- Cmdlet `Install-WUUpdates`

Exercise PS70

- Listing modules and their contents
- Loading modules
- Exploring the *ActiveDirectory* module

Log File Analysis

- Windows Logs can be saved/exported as *evtx* file.
- Loading logs in PowerShell with Cmdlet `Get-WinEvent`
- Use a pipeline to filter the events according to the specified criteria.

Exercise PS71

- Get used to Windows Event Viewer
- Perform log file analysis
- Find traces of user activities à la IT forensics
- Document findings and results

PowerShell goes Internet

HTTP intercktion, E-Mail delivery

Interaction with Web Sites

- PowerShell allows for interactions via CLI
- Roughly equivalent to UNIX commands *wget* and *curl*
  - *wget* and *curl* allow downloads of websites and files
  - *curl* also allows e.g. sending login data

- For all these actions PowerShell offers the Cmdlet `Invoke-WebRequest`
  - sends HTTP(S) requests to websites
  - is very flexible (>20 parameters)

Sending E-mails through PowerShell

- When scheduling tasks user defined actions are performed, if certain trigger conditions are met.
- One possible action could be an info mail to the administrator.
- PowerShell allows automation of this action with the help of Cmdlet `Send-MailMessage`

Exercise PS72

- Formulate and send HTTP requests
- Send E-mails via PowerShell
- Use PowerShell (somewhat) like *wget* and *curl* on *Nix systems

PowerShell CIM and WMI

Information gathering for administration and remote servicing

Common Information Model (CIM)

- Is a DMTF standard for management of IT systems
- CIM provides a data model
- Different implementations are possible
- Protocol: WS-Man (Web Services-Management)

Windows Management Instrumentation (WMI)

- The WMI is a CIM extension made by Microsoft
- Access to almost all settings
  - Read and write access
  - Local or over the network
- Well-knwon CLI for *cmd.exe* is WMIC
- Protocol: RPC (Remote Procedure Call)

Windows Management Instrumentation

- PowerShell implemented WMI Cmdlets prior to version 3.0
  - These are now deprecated
- Since PowerShell 3.0 there are CIM Cmdlets

WMI-Cmdlet (deprecated)	CIM-Cmdlet
Get-WmiObject	Get-CimInstance
Invoke-WmiObject	Invoke-CimMethod
Register-WmiEvent	Register-CimIndicationEvent
Remove-WmiObject	Remove-CimInstance
Set-WmiInstance	Set-CimInstance

Jobs and Sessions

Access to remote computers, workflows, concurrency

Controlling Remote Computers

- Backgrounds:
  - Administration often entails remote servicing
  - Number of serviced machines is potentially (very) large
- Idea:
  - Administration from a central workstation
  - Tasks („Jobs“) are executed on remote computers from there
  - Automation through PowerShell scripts

Using Jobs and Sessions

- *Job* in PowerShell refers to a background process
  - Already known as `Invoke-Command` (chapter on networks)
  - Some Cmdlets implement the parameter `-AsJob`
  - Roughly equivalent to *NIX option `--daemon` resp. `&` after a command

- A job does not block a PowerShell session
- Connection to remote computer via a session
  - local : Job results only available in the session
  - remote : Job results can be fetched or saved

Types of Sessions

- Local Session
  - Start with `Enter-PSSession`
  - End with `Exit-PSSession`
- Remote Session
  - Start with `New-PSSession`
  - Session is managed through a variable

Local Session
Pattern

Step	Description	Cmdlets
1	Start interactive session	`Enter-PSSession -ComputerName <Ziel>`
2	Started desired jobs	`Start-Job -ScriptBlock { … }`
3	Fetch job results	`Receive-Job -Name <JobName>`
4	Delete jobs on the target system	`Remove-Job -Name <JobName>`
5	Terminate interactive session	`Exit-PSSession`

Local Session
Hints

- Remember (or save) names of started jobs
  - Generated name is displayed when the job is started (see example)
  - Jobs can be queried with `Get-Job`
  - Job can be manually named with parameter `-Name`
- Created jobs should be deleted after fetching the results
  - jobs might otherwise become „zombies“
  - Fetch results multiple times with parameter `-Keep`
- Process multiple jobs e.g. with `ForEach-Object`

Local Session
Troubleshooting

- The service *Windows-Remoteverwaltung (WS-Verwaltung)* must be running on the target system.
  - If needed, start with `Start-Service -Name WinRM` manually
  - If needed, set startup type to automatic

- The network connection type to the target system must not be public.
  - Outside a domain: choose Home Network or Workplace Network
  - Inside Domäne: choose Domain Network

- If access is still denied, execute `winrm quickconfig` in a PowerShell with elevated privileges and confirm the changes with `y` key.

Local Session
Example

```
PS C:\Windows\System32> Enter-PSSession -ComputerName localhost
[localhost]: PS C:\Users\anr\Documents> Start-Job -ScriptBlock { Get-Process | Me
asure-Object | Format-Table -Property Count }

Id     Name            PSJobTypeName   State         HasMoreData     Location
--     ----            -------------   -----         -----------     --------
1      Job1            BackgroundJob   Running       True            localhost


[localhost]: PS C:\Users\anr\Documents> Receive-Job -Name Job1

Count
-----
  189


[localhost]: PS C:\Users\anr\Documents> Remove-Job -Name Job1
[localhost]: PS C:\Users\anr\Documents> Exit-PSSession
```

Remote Session
Pattern

Step	Description	Cmdlets
1	Create new session and store it in a variable	`$s=New-PSSession -ComputerName <Ziel>`
2	Start remote jobs	`Invoke-Command -Session $s -ScriptBlock { Start-Job { … } }`
3	Fetch job results and store them in a variable	`$result=Invoke-Command -Session $s -ScriptBlock { Receive-Job … }`

Remote Session
Hints

- All hints for local sessions are still valid
- A session can be interrupted with `Invoke-Command -Disconnected` for some time.
- Use job results locally (in a variable `$result`):
  - `$result=Invoke-Command -Session $s -ScriptBlock { Receive-Job … }`
- Save job results on remote system (in file *C:\xyz*):
  - `Invoke-Command -Session $s -Command { Receive-Job … | Out-File C:\xyz }`

Remote Session
Example

```
PS C:\Windows\System32> $s=New-PSSession -ComputerName localhost
PS C:\Windows\System32> Invoke-Command -Session $s -ScriptBlock { Start-Job -ScriptBlock { Get-Process | Measure-Object
| Format-Table -Property Count } }

Id     Name            PSJobTypeName   State         HasMoreData     Location             Command                  PSC
                                                                                                                  omp
                                                                                                                  ute
                                                                                                                  rNa
                                                                                                                  me
--     ----            -------------   -----         -----------     --------             -------                  ---
1      Job1            BackgroundJob   Running       True            localhost             Get-Process | Measure-O… lo…

PS C:\Windows\System32> $result=Invoke-Command -Session $s -ScriptBlock { Receive-Job -Name Job1 -Keep }
PS C:\Windows\System32> Invoke-Command -Session $s -Command { Receive-Job -Name Job1 | Out-File -FilePath "C:\Users\anr\
Downloads\ps.txt" }
```

Scheduled Jobs

- Automation of classic Windows Task Scheduler
- Jobs can be managed with both Task Scheduler and PowerShell
- Scheduled Jobs are called *Aufgaben* on a German Windows system

- Differences to *normal* jobs:
  - Usually run periodically (like GNU/Linux cronjobs)
  - Execution depends on conditions (triggers)
  - Job can have options
  - Registered jobs remain on the system, e.g. for later modifications

Scheduled Jobs
Pattern

Step	Description	Cmdlets
1	Define triggers	`New-JobTrigger`
2	Define options	`New-ScheduledJobOption`
3	Create and register job	`Register-ScheduledJob`

- Triggers and options are best managed with a variable each
- This separation increases clarity
- A *Scheduled Job* should be provided with a *telling* name

Scheduled Jobs
Hints

- Scheduled Jobs are not available as default.
- The module *PSScheduledJob* has to be imported.
- Problem: Import is blocked due to compatibility restrictions
- Details: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_windows_powershell_compatibility?view=powershell-7.3
  - Modify Deny List in *$PSHOME\powershell.config.json*
  - Administrator provileges required (e.g. invoke file from administrator PowerShell)
  - Delete entry for *PSScheduledJob*, save the file
  - Exit PowerShell

- Run a new PowerShell as administrator
- `Import-Module –Name PSScheduledJob -SkipEditionCheck`

Scheduled Jobs
Hands-on

- A Scheduled Job with name *Prozessuebersicht* is to be created. The job should list all system processes and save them in a log file. Further requirements include:
  - The job is to run only on tuesdays and wednesdays.
  - The job shall be executed at 9 a.m. precisely.
  - The Job shall run with elevated privileges (although actually not necessary here).
  - If the computer is on hibernate or asleep, it shall be „woken up“, for the job to run in any case.
  - The job shall even run if the computer is running on battery.

Scheduled Jobs
Example

- Under Windows Server 2019
- `Import-Module PSScheduledJob` has to be executed in advance

```
PS C:\Users\Administrator> $trigger=New-JobTrigger -Weekly -DaysOfWeek "Tuesday","Wednesday" -At "09:00"
PS C:\Users\Administrator> $jOptions=New-ScheduledJobOption -StartIfOnBattery -RunElevated -WakeToRun
PS C:\Users\Administrator> Register-ScheduledJob -Name "Prozessuebersicht" -ScriptBlock { Get-Process | Out-File "C:\use
rs\Administrator\Downloads\ps.txt" } -Trigger $trigger -ScheduledJobOption $jOptions

Id         Name            JobTriggers     Command                                 Enabled
--         ----            -----------     -------                                 -------
1          Prozessueber... 1               Get-Process | Out-File "C:\users\Adm... True
```

Exercise PS73

- Evaluate installed *HotFixes*
- Start local and remote sessions with jobs
- Configure and test a Scheduled Job
  - only under Windows Server

Workflows and Concurrency

Workflows

- Workflows are only available until PowerShell 5 !
- Frequently used actions are aggregated to workflow
- Execution by Windows Workflow Foundation (WWF)
- PowerShell (<=5) keyword is `Workflow`
- Especially useful for running tasks in parallel
  - e.g. scripts, that run on multiple computer simultaneously

Workflows
Example

- Print *Hello World!* on *stdout*

- Important parameters for all workflows
  - `-AsJob` to execute the workflow as job (i.e. in the background)
  - `-PSComputerName` for the workflow‘s target system
  - `-PSCredentials` for the user account running the workflow
  - `-JobName` for a suitable job name

Workflows
Concurrency

- The WWF allows further control mechanisms
  - e.g. explicitly requested parallel processing (concurrency)
  - only for workflows (in the WWF), not for other (PS-)scripts

- Knowledge of WWF and Workflows are only applicable in legacy PowerShell
  - But what can be do in PowerShell 7 without workflows ?

Concurrency since PowerShell 7

- The Cmdlet `ForEach-Object` now provides parameter *-Parallel*
  - allows concurrent execution of script blocks
  - available since version 7
  - Version 6 has neither workflows nor *ForEach -Parallel* !
- Important parameter *-ThrottleLimit*
  - defines the maximum number of parallel blocks
  - default value is 5
- The execution time of commands can be measured with `Measure-Command`

Concurrency since PowerShell 7
Example

- Print 5 lines with an interval of 1 second (sequential execution)

```
PS C:\Users\anr> 1..5 | ForEach-Object { Write-Host "Nummer $_"; Start-Sleep -Se
conds 1 }
```

- Demo

- Execution time

```
PS C:\Users\anr> (Measure-Command -Expression { 1..5 | ForEach-Object { "Nummer
$_"; Start-Sleep -Seconds 1 } }).Seconds
5
```

Concurrency since PowerShell 7
Example

- Print 5 lines with an interval of 1 second (parallel execution)

```
PS C:\Users\anr> 1..5 | ForEach-Object -Parallel { Write-Host "Nummer $_"; Start
-Sleep -Seconds 1 }
```

- Demo

- Execution time

```
PS C:\Users\anr> (Measure-Command -Expression { 1..5 | ForEach-Object -Parallel
{ Write-Host "Nummer $_"; Start-Sleep -Seconds 1 } }).Seconds
1
```

Concurrency since PowerShell 7
Example

- The example makes perfect use of concurrency.

```
PS C:\Users\anr> 1..5 | ForEach-Object -Parallel { Write-Host "Nummer $_"; Start
-Sleep -Seconds 1 } -ThrottleLimit 3
```

☞ What will be the execution time now ?

Concurrency
Using concurrency with jobs

- Original example of a job

```
PS C:\Users\anr> $job = 1..5 | ForEach-Object -Parallel { Write-Host "Nummer $_"
; Start-Sleep -Seconds 1 } -AsJob
```

- Fetch results with `Receive-Job` as usual
  - Problem: how do I know all concurrent threads have finished execution ?
  - Solution: use `Wait-Job`

- `Wait-Job` waits for a job to finish execution
  - Job results can be passed in a pipeline

```
PS C:\Users\anr> $job | Wait-Job | Receive-Job
```

Concurrency
Hints and Best Practices

- Concurrency only works with `ForEach-Object` (Cmdlet),
  not with *ForEach* (as keyword for a loop in PS programming)

- When to use concurrency…
  - with CPU intense scripts
  - with scripts that are waiting for an event

- When not to use concurrency…
  - with commands with very short execution time (mind the overhead!)

Exercise PS74

- Create concurrent script blocks
- Measure command execution time
- Compare sequential and parallel execution