PowerShell for Beginners

Basics and Complex Exercises

Table of Contents

Directories and Files

Basic manipulation of NTFS objects

Navigating the File System

- Cmdlets from the *Location* family, but not *WinHomeLocation*
- Paths can be managed through a stack like in cmd (*pushd*, *popd*)
- The terms *directory* and *folder* are used interchangeably

Operation	Cmdlet	Aliase (PS, cmd, bash)
Enter directory	`Set-Location`	`sl, cd, chdir`
Print current working directory	`Get-Location`	`gl, pwd`
Push path location to stack	`Push-Location`	`pushd`
Get path location from stack	`Pop-Location`	`popd`

Operations in the File System

- CRUD pattern (Create, Read, Update, Delete) for files
- Some Cmdlets from the *Item* family

- Caution:
  - `Invoke-Item` calls a file‘s default application
  - File IO with *Content* (*Get*, *Set*, *Add*, *Clear*) family
  - Files and directories are created by the same Cmdlet, but directories (folders) require parameter *-ItemType Directory*

Operations in the File System

Operation	Cmdlet	Aliases (PS, cmd, bash)
Create a file	`New-Item`	`ni`
Create a folder (directory)	`New-Item … -ItemType Directory`	`ni`
Handle to a file system object	`Get-Item`	`gi`
Print file content	`Get-Content`	`gc,type,cat`
Move file/folder	`Move-Item`	`mi,move,mv`
Rename file / folder	`Rename-Item`	`rni,ren`
Copy file / folder	`Copy-Item`	`cpi,copy,cp`
Write to file	`Set-Content`	
Delete file / folder	`Remove-Item`	`ri,del,rd,rm, rmdir,erase`

Searching the File System

- Listing and searching items with `Get-ChildItem`
  - Implements actions of *dir* resp. *ls*
  - these two are available as built-in aliases
  - PowerShell alias is *gci*
  - important parameters: *-Recurse*, *-Include*, *-Exclude*
  - *System.IO.FileInfo* has 100(!) properties and methods

Exercises PS51, PS52
File System

- CRUD operations in the file system with PowerShell (PS51)
- Searching the file system, using wildcards (PS52)

File System Permissions

Working with NTFS access permissions

NTFS Permissions
Basics

- Describe access rights for file system objects through ACLs (Access Control Lists)
- Entities called *Principals* are granted permissions
- There are *allow* and *deny*) permissions
- Windows implements basic and enhanced permissions
- Configuring file system permissions is usually important for server administrators.

NTFS Permissions
Permission Levels (Pyramide)

Level	Basic Permissions
FullAccess (FA)	Full access; special permissions
ModifyAccess (MA)	Modify, write
ReadAccess (RA)	Read, execute; display content
NoAccess (NA)	--

- Each level contains the permissions of the levels below
  - subset relation can be regarded as ordering relation
  - the permission levels are ordered by subset relations

☞ What are the properties of an ordering relation ?

NTFS Permissions
Permission Levels

- Defined in .NET namespace `System.Security.AccessControl`
  - available in `enum FileSystemRights`
- important fields in the *enum* (matching permission levels)

Level	Enum field	Number value
RA	Read	131209
	ReadAndExecute	131241
MA	Modify	197055
FA	FullControl	2032127

- https://learn.microsoft.com/en-us/dotnet/api/system.security.accesscontrol.filesystemrights?source=recommendations&view=net-7.0

NTFS Permissions
Changing Access Permissions

- In cmd changes are made via *icacls* (this also works in PowerShell)
- Cmdlets for changing NTFS permissions
  - `Get-Acl` resp. `Set-Acl`
  - important parameters `-Path`, `-AclObject`

- Basic concept:
  - each permission is a *FileSystemAccessRule* object
  - permissions can be added or deleted
  - Inheritence behavior is controlled through an ACL method

NTFS Permissions

The PowerShell Way

Step	Description	Cmdlets
1	Get ACL (NTFS permissions) of the file system object and store it in a variable	`$acl=Get-Acl -Path <Folder>`
2	Break inheritence and delete inherited permissions	`$acl` `, $F`

3	Create object with new access permissions	`$rule=New-Object`
		`System.Security.AccessControl.File`
		`SystemAccessRule('<computer>\<acco`
		`unt>',<level>,<inheritence>,<propa`
		`gation>,<ruleType>)`

4	Add rule with new access permissions to existing ACL	`$acl.AddAccessRule($acl)`
5	Apply updated ACL to the file system object	`Set-Acl -Path <Folder>` `-AclObject $acl`

Yes, the .NET class really is *System.Security.AccessControl.FileSystemAccessRule*

NTFS Permissions
Example

- Directory *Test* shall obtain FA permissions for user *anr* and RA permissions for group User (*Benutzer*). (The folder already exists.)

```
PS C:\Users\anr\Downloads> $acl = Get-Acl -Path .\Test\
PS C:\Users\anr\Downloads> $acl.SetAccessRuleProtection($True, $False)
PS C:\Users\anr\Downloads> $rule=New-Object System.Security.AccessControl.FileSystemAccess
Rule("anr","FullControl","ContainerInherit, ObjectInherit","None","Allow")
PS C:\Users\anr\Downloads> $acl.AddAccessRule($rule)
PS C:\Users\anr\Downloads> $rule=New-Object System.Security.AccessControl.FileSystemAccess
Rule("Benutzer","ReadAndExecute","ContainerInherit, ObjectInherit", "None", "Allow")
PS C:\Users\anr\Downloads> $acl.AddAccessRule($rule)
PS C:\Users\anr\Downloads> Set-Acl -Path Test -AclObject $acl
```

NTFS Permissions
Example

- In directory *Test* the RA permission for group User (*Benutzer)* is to be removed.

```
PS C:\Users\anr\Downloads> $acl = Get-Acl -Path .\Test\
PS C:\Users\anr\Downloads> $rule=New-Object System.Security.AccessControl.FileSystemAccess
Rule("Benutzer","ReadAndExecute","ContainerInherit, ObjectInherit", "None", "Allow")
PS C:\Users\anr\Downloads> $acl.RemoveAccessRule($rule)
True
PS C:\Users\anr\Downloads> Set-Acl -Path .\Test\ -AclObject $acl
```

NTFS Permissions
Hints

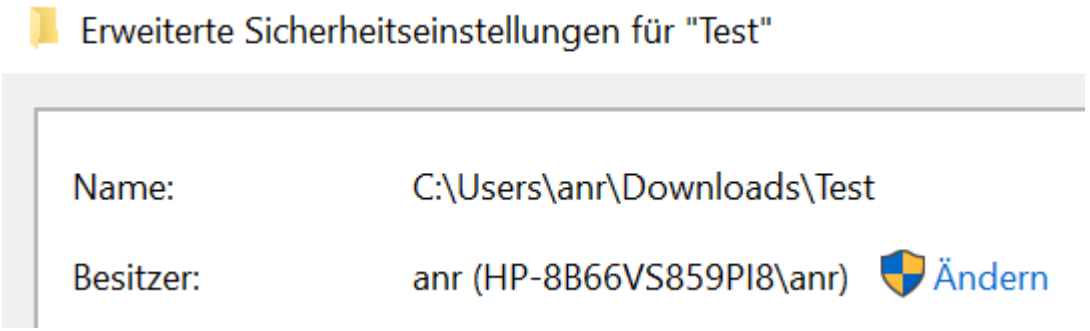
- Basic OOP knowledge is really helpful for the PowerShell workflow
  - Instantiate an object `New-Object …`
  - accessing a property `(Get-Acl <Folder>).Owner`
  - working with methods `$acl.AddAccessRule(…)`

- If in doubt, let students also use *icacls*

- After breaking the inheritence and deleting all existing permissions, a PowerShell with elevated privileges is necessary for `Set-Acl`, since there are no explicit permissions left for the file system object.

Taking Ownership of an Object

- Each file system object has exactly one owner
- In Windows Explorer
  → call context menu of the object (right click or button in ribbon)
  → Tab *Security*
  → Button *Extended*



Taking Ownership of an Object

- Change owner via button *Change* in GUI (elevated privileges required)
- In cmd with `takeown /f <Folder>`
  - this also works in PowerShell…
  - …but of course there are Cmdlets as well

Taking Ownership of an Object
The PowerShell Way

Step	Description	Cmdlets
1	Get file system object ACL (NTFS permissions) and store it in a variable	`$acl=Get-Acl -Path <Folder>`
2	Create user object for the new user	`$user=New-Object System.Security.Principal.Ntaccount('<computer>\<account>')`
3	Set new owner in ACL	`$acl.SetOwner($user)`
4	Apply updated ACL to file system object	`Set-Acl -Path <Folder> -AclObject $acl`

NTFS Permissions

- PowerShell seems more complicated than *icacls*, *takeown* at first glance, but
  - more structured, clearer, more readable
  - more possibilities due to object oriented approach (many properties)
  - is easily converted to scripts

- Demo: Self-made function `Set-NtfsPermissions`

☞ Scripts, programming and functions follow in chapter 8 ;-)

Exercises PS53
NTFS Permissions

- Retrieve and set NTFS permissions
- Handle ACLs for file system objects

Shares and Network Drives

Creating shares, mapping network drives

Shares and Network Drives

- Folders are shared to allow accessing resources from other computers.
  - there are some default shares such as *C$*, *Admin$*, …
  - user defined shares are a more interesting topic

- Network drives are resources that can be mapped and behave like a local drive.
  - often reside in the LAN (mainly serve as data storage)
  - remote access of e.g. WebDAV directories is possible
  - access to cloud storage such as OneDrive, iCloud, Nextcloud,…

Shares and Network Drives
Retrieving the necessary information

- Existing shares on a system can be displayed with `Get-SmbShare`
  - *Smb* means *Server Message Block*
  - protocol for file, print and server services in networks
  - Cmdlet is roughly equivalent to functionality of *net share*

```
PS C:\Users\anr> Get-SmbShare

Name    ScopeName Path       Description
----    --------- ----       -----------
ADMIN$  *         C:\windows Remoteverwaltung
C$      *         C:\        Standardfreigabe
IPC$    *                    Remote-IPC
```

Shares and Network Drives
Retrieving the necessary information

- Existing drives on a system can be displayed with `Get-PSDrive`
  - drives on the local host are a *PSDrive*,
  - …so are environment variables and also…
  - …two hives of the registry are available as drives

- The type of drive is stored in the *Provider* property
  - *FileSystem* are regular file systems (local or remote)
  - *Environment* contains environment variables
  - *Registry* is for hives or keys from the Windows registry

Shares and Network Drives

Retrieving the necessary information

```
PS C:\Users\anr> Get-PSDrive

Name           Used (GB)     Free (GB) Provider      Root
----           ---------     --------- --------      ----
Alias                                  Alias
C                 175,94         61,53 FileSystem    C:\
Cert                                   Certificate   \
Env                                    Environment
Function                               Function
HKCU                                   Registry      HKEY_CURRENT_USER
HKLM                                   Registry      HKEY_LOCAL_MACHI…
Temp              175,94         61,53 FileSystem    C:\Users\anr\App…
Variable                               Variable
WSMan                                  WSMan
```

Shares and Network Drives
Retrieving the necessary information

- Mapped network drives can be displayed with `Get-SmbMapping`
  - *Smb* means *Server Message Block*
  - Cmdlet is roughly equivalent to the functionality of *net use*

Shares and Network Drives
Workflow

- Mapping a network drive typically involves three steps:

(1) Create folder in file system, configure NTFS permissions

(2) Share folder, configure share permissions

(3) Map shared folder as network drive

Shares and Network Drives
Example

- Creating a shared folder with write access for all users
- ☞ Which Cmdlet creates a new folder ?

```
PS C:\Users\anr\Downloads> ni Tausch -ItemType Directory

    Directory: C:\Users\anr\Downloads

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----          21.02.2023    18:42                Tausch
```

- Creating the folder in the file system, set NTFS permissions
  - `Get-Acl`, break inheritence (if required), define rules, `Set-Acl`

Shares and Network Drives
Example

- Creating share, configuring share permissions

```
PS C:\Users\anr\Downloads> New-SmbShare -Name "Tauschlaufwerk" -Path "C:\Users\anr\
Downloads\Tausch\" -ConcurrentUserLimit 10 -ChangeAccess "Jeder"

Name           ScopeName Path                          Description
----           --------- ----                          -----------
Tauschlaufwerk *         C:\Users\anr\Downloads\Tausch
```

- Parameter *-Path* contains the absolute path to folder
- Maximum number of concurrent accesses set by *-ConcurrentUserLimit*
- Write permission is called *ChangeAccess* for share permissions
  - With NTFS permissions it is called *Modify*
  - All principals are contained in share group *Anyone* (*Jeder*), not in *User* (*Benutzer*)

Shares and Network Drives
Example

- Map shared folder as network drive

```
PS C:\Users\anr> New-SmbMapping -LocalPath "T:" -RemotePath "\\localhost\Tauschlaufwerk"

Status Local Path Remote Path
------ ---------- -----------
OK     T:         \\localhost\Tauschlaufwerk
```

- The *LocalPath* doesn‘t need to be a driver letter; although that is quite handy
- The *RemotePath* is passed in UNC format

Shares and Network Drives
Example

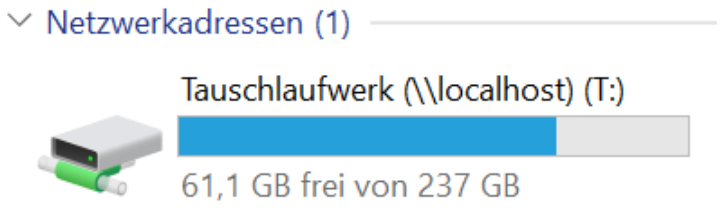
- Within PowerShell the drive T: is immediately accessible

```
PS C:\Users\anr> T:
PS T:\>
```

- Using the network drive in Windows Explorer:
  - Restart Explorer process (restarts automatically after stopping)

```
PS C:\Users\anr> Stop-Process -ProcessName "explorer"
```

  - Open Explorer window



Shares and Network Drives
Removal

- First remove network drive mapping with `Remove-SmbMapping`
- Second remove share (elevated privileges required) with `Remove-SmbShare`
- If necessary, remove file system object (folder) with `Remove-Item` (elevated privileges may be required)

Shares and Network Drives
Hints

- For creation and configuration of share permission a PowerShell wirh elevated privileges may be required.
- To map the drive, use a normal PowerShell.
  - Otherwise the drive may not be visible in Windows Explorer
- Since the used Cmdlets don‘t produce output, there is no object structure we can benefit from. Hence *net share*, *net use* can also be used without severe disadvantages.
- With WebDAV drives it is advisable to use the GUI or *net* commands
  - In PowerShell some use cases cannot be realized
  - Handling of URLs is problematic; UNC usually works

Exercise PS54, PS55
Shares and Network Drives

- Create and manage SMB shares (PS54)
- Create, map and manage network drives (PS55)

Working with the Windows Registry

Read and define Registry Keys

Windows Registry

- The Windows registry is the central storage of configuration information for Windows computers and installed software
- Consists of 5 so-called *hives* as roots of key-trees
  - HKEY_CLASSES_ROOT (HCR)
  - HKEY_CURRENT_USER (HKCU)
  - HKEY_LOCAL_MACHINE (HKLM)
  - HKEY_USERS (HKU)
  - HKEY_CURRENT_CONFIG (HCC)
- HKEY means *handle (to a) key*

Windows Registry

- Access via PowerShell:
  - the hives HKCU and HKLM are available as PSDrive as standard
  - recognizable by provider *Registry*
  - behaves (almost) like a file system

```
PS C:\Users\anr\Downloads> Get-PSDrive -PSProvider Registry

Name           Used (GB)     Free (GB) Provider      Root
----           ---------     --------- --------      ----
HKCU                                   Registry      HKEY_CURRENT_USER
HKLM                                   Registry      HKEY_LOCAL_MACHINE
```

- Cmdlets specially tailored for manipulation of registry keys
  - `Get-ItemProperty, Set-ItemProperty`

Windows Registry
Example

- Fetching some basic information about the current user session

```
PS C:\Users\anr> Set-Location HKCU:
PS HKCU:\> Get-ItemProperty "Volatile Environment"

LOGONSERVER                : \\HP-8B66VS859PI8
USERDOMAIN                 : HP-8B66VS859PI8
USERNAME                   : anr
USERPROFILE                : C:\Users\anr
HOMEPATH                   : \Users\anr
HOMEDRIVE                  : C:
APPDATA                    : C:\Users\anr\AppData\Roaming
LOCALAPPDATA               : C:\Users\anr\AppData\Local
USERDOMAIN_ROAMINGPROFILE  : HP-8B66VS859PI8
PSPath                     : Microsoft.PowerShell.Core\Registry::HKEY_CURR
                             ENT_USER\Volatile Environment
PSParentPath               : Microsoft.PowerShell.Core\Registry::HKEY_CURR
                             ENT_USER
PSChildName                : Volatile Environment
PSDrive                    : HKCU
PSProvider                 : Microsoft.PowerShell.Core\Registry
```

Exercise PS56
Windows Registry

- Use registry as PSDrive
- Display keys and their values
- Create new keys in the registry