Windows PowerShell

βeta VZW

(versie 2018.05-beta)

Hoofdstuk 1. Inleiding

1. Wat is PowerShell

PowerShell is de moderne scripting taal voor Windows computers. Het is de opvolger van de "DOS-command prompt" (die nog steeds bestaat) en VBScript. PowerShell kan gebruikt worden om commando's in te geven via de command line en om scripts te schrijven. Het is tevens een object-georiënteerde programmeertaal die geïntegreerd is met de NET omgeving van Microsoft. Deze syllabus is gebaseerd op PowerShell 5.1. De allereerste versie werd uitgebracht in 2006 voor Windows XP SP2 en Windows Server 2003.

Tabel 1. Versies van PowerShell

VERSIE	BESTURINGSSYSTEEM
1.0	Windows XP SP2, Windows Server 2003, Windows Vista Feature in Windows Server 2008
2.0	Onderdeel van Windows 7 en Windows Server 2008 R2 Aparte installatie voor Windows XP SP3, Windows Server 2003 SP2 en Windows Vista SP1
3.0	Geïntegreerd in Windows 8 en Windows Server 2012. Aparte installatie voor Windows 7 SP1, Windows Server 2008 SP1 en Windows Server 2008 R2 SP1. Het kan ook geïnstalleerd worden als onderdeel van Windows Management Framework 3.0
4.0	Geïntegreerd in Windows 8.1 en Windows Server 2012 R2 Het kan ook geïnstalleerd worden als onderdeel van Windows Management Framework 4.0
5.0	Geïntegreerd in Windows 10 en Windows Server 2016. Aparte installatie voor Windows 7 SP1, Windows 8.1, Windows Server 2008 R2 SP1, Windows Server 2012 en Windows Server 2012 R2 Het kan ook geïnstalleerd worden als onderdeel van Windows Management Framework 5.0
5.1	Geïntegreerd in Windows 10 Build 14393("Anniversary update") en Windows Server 2016. Aparte installatie voor Windows 8.1, Windows Server 2008 SP1, Windows Server 2012 en Windows Server 2012 R2
6.0	Beschikbaar voor Windows en Linux als aparte download

In deze syllabus gebruiken we versie 5.1.

2. PowerShell editors

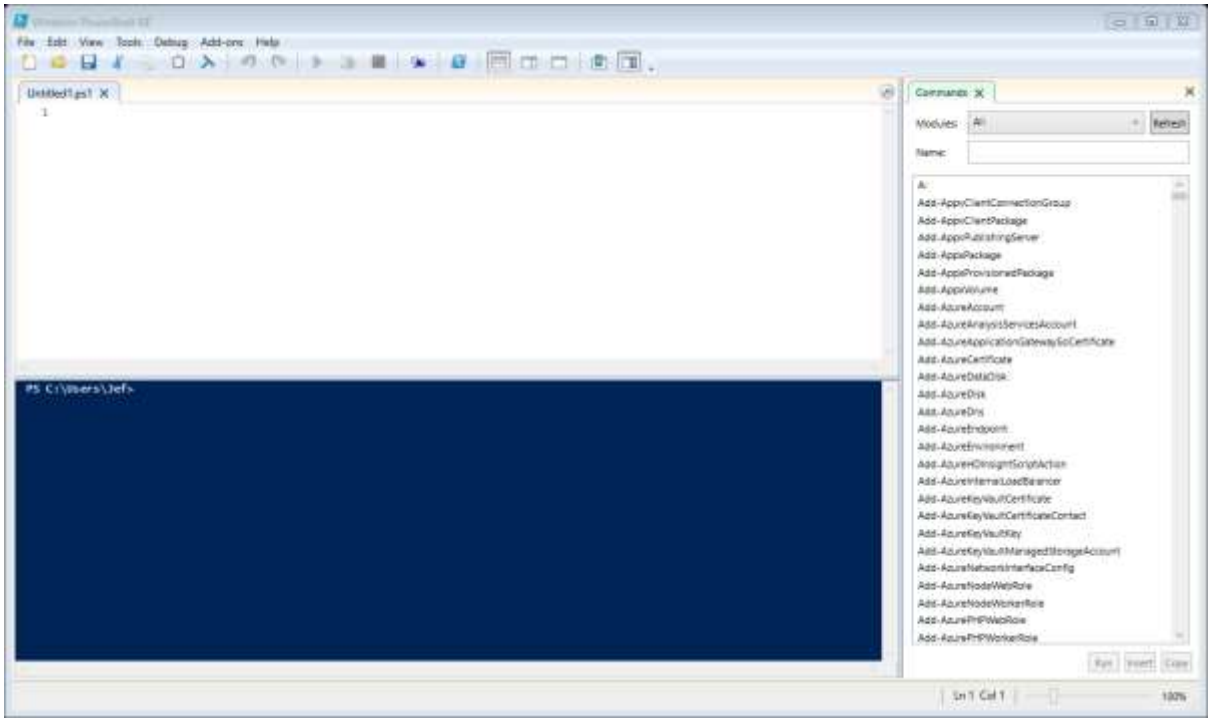
We kunnen de Windows PowerShell op twee verschillende manieren gebruiken:

- In een command prompt omgeving waar we een commando intypen en uitvoeren
- Als script waarbij we meerdere commando's samenvoegen en bewaren in een bestand. Dat bestand (script) kunnen we vervolgens uitvoeren

Een script is een gewoon tekstbestand en kan met eender welke gewone teksteditor (dus niet met MS-Word) geschreven worden. Het is echter handig wanneer men wat extra hulp krijgt bij het intypen van de verschillende commando's. Daarom is het efficiënter gebruik te maken van gespecialiseerde editors. Microsoft biedt er twee aan: PowerShell Integrated Scripting Environment (PowerShell ISE) en Visual Studio Code.

2. A. PowerShell ISE

PowerShell ISE werd voor het eerst geïntroduceerd met PowerShell 2.0. De editor wordt tezamen met PowerShell geïnstalleerd en is dus altijd voor handen op een Windows machine.



Figuur 1. PowerShell ISE

Het PowerShell ISE scherm bestaat uit een editor, een command prompt en een help scherm. Commando's kunnen automatisch worden aangevuld (IntelliSense), er is automatische controle van de syntax en mogelijkheden om te debuggen.

Op technet biedt Microsoft ook een artikel aan met een lijst met extra features die geïnstalleerd kunnen worden in Powershell ISE (Battalio, 2017).

De laatste nieuwe versie van PowerShell ISE (de “preview”) kan geïnstalleerd worden met het volgende PowerShell commando:

```
Install-Module -Name PowershellISE-Preview
```

Vervolgens kan men het Update-Module commando gebruiken om PowerShell ISE te updaten. Om de preview versie te starten kan men Start-ISEPreview gebruiken.

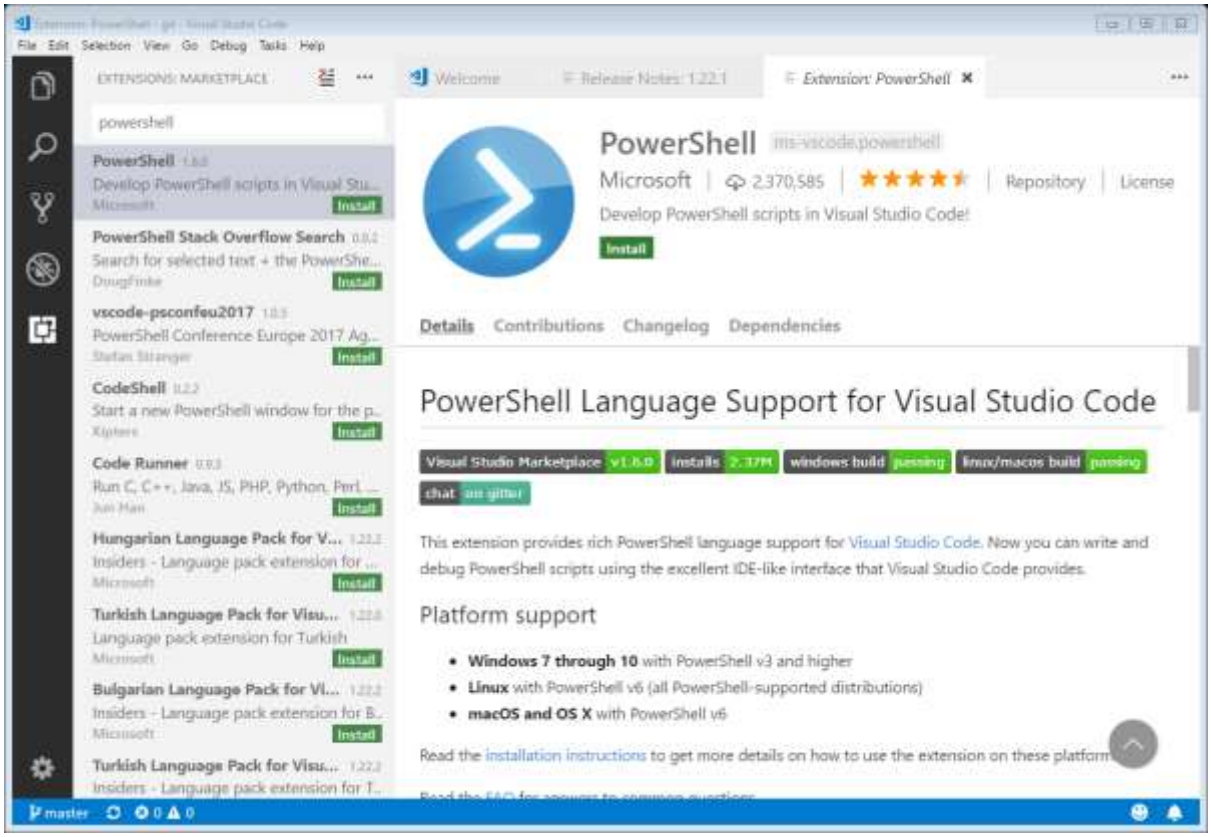
Normaal gesproken zal men ISE starten via de startknop, maar men kan ook het volgende commando intypen:

```
powershell_ise
```

In een PowerShell console is het eenvoudiger omdat er een alias is gedefinieerd: ise.

2. B. Visual Studio Code

De Visual Studio Code editor kan gedownload worden van http://www.visualStudio.com. Het is een editor die voor verschillende programmeertalen gebruikt kan worden. Men mag Visual Studio Code echter niet verwarren met Visual Studio.



Figuur 2. Visual Studio Code

Om PowerShell te kunnen gebruiken in Visual Studio Code moet men een extra *extension* installeren. Deze extension zorgt voor Syntax highlighting, PowerShell Script Analyzer en debugging. Ten opzichte van PowerShell ISE biedt Visual Studio Code een betere integratie

met Git, het gedistribueerde versiebeheersysteem. Het is ook een betere optie wanneer men Visual Studio Code gebruikt voor andere programmeertalen.

3. Help

PowerShell heeft een uitgebreid help-systeem. Vanaf PowerShell 3 is dit nog meer uitgebreid via *Updatable Help*. Auteurs van de onderdelen van PowerShell (modules) kunnen de meest recente versie van het helpsysteem voor de module op een web server zetten.

De eerste maal dat men het helpsysteem gebruikt (door Get-Help uit te voeren) zal men gevraagd worden of men het helpsysteem wil updaten. Wanneer dat het geval is, zal PowerShell proberen om de meest recente versie van de helpbestanden te downloaden voor alle modules die updatable help ondersteunen.

Wanneer dit downloaden niet lukt omdat er geen internet toegang is, zal PowerShell een beperkte versie van de help van een commando tonen.

Men kan meer informatie opvragen over *updatable help* op de volgende manier:

```
PS C:\Users\Jef> Get-Help about_updatable_help
TOPIC
    About_Updatable_Help

SHORT DESCRIPTION
    Describes the updatable help system in Windows PowerShell.

LONG DESCRIPTION
    Windows PowerShell provides several different ways to access
    the most up-to-date help topics for Windows PowerShell cmdlets
    and concepts.

    The Updatable Help system, introduced in Windows PowerShell 3.0,
    is designed to assure that you always have the newest help topics
    on your local computer so that you can read them at the command
    line. It makes it easy to download and install help files and
    to update them whenever newer help files become available.
…
```

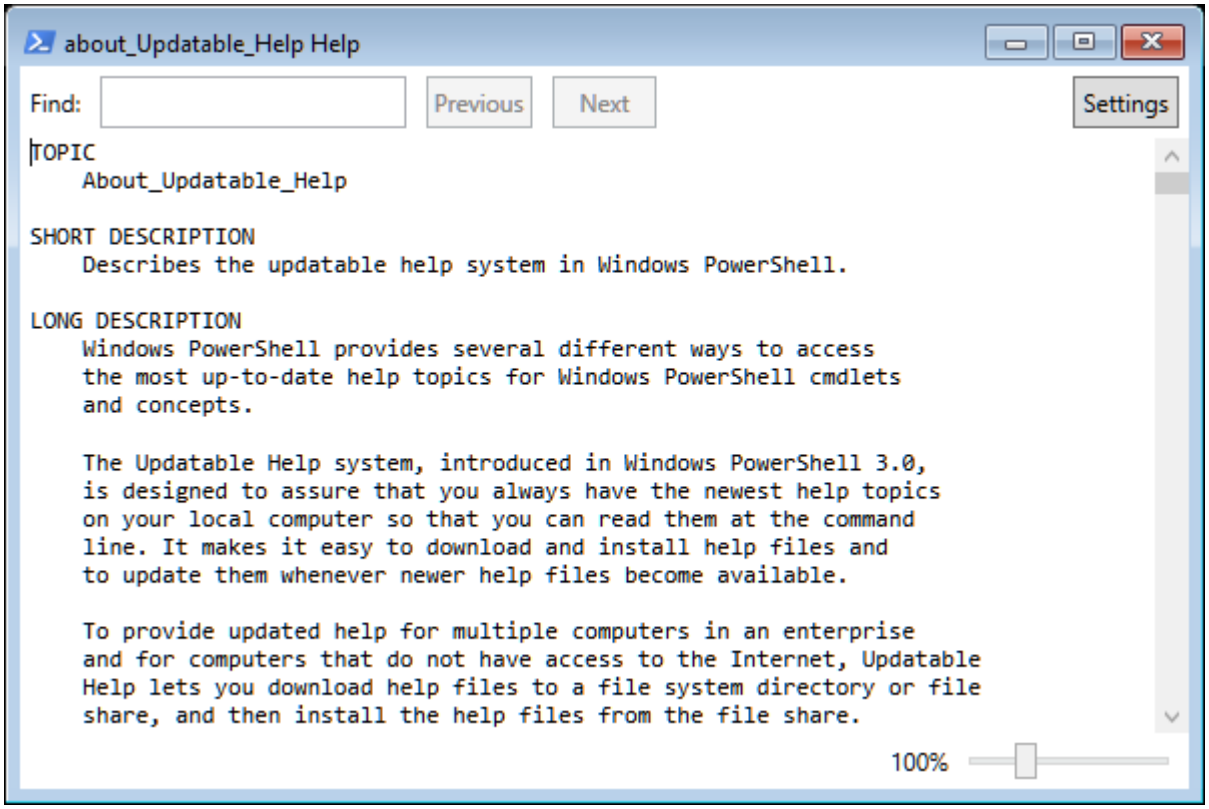
Hiermee hebben we ook kennis gemaakt met het commando dat we kunnen gebruiken om hulp op te vragen: Get-Help. De output van het vorige voorbeeld is afgekapt omdat de informatie te lang is. Men kan door de informatie scrollen, maar men kan ook gebruik maken van een alias: help about_updatable_help. Het voordeel van de help-alias is dat de informatie per scherm getoond wordt (-- More --):

```
PS C:\Users\Jef> help about_updatable_help
TOPIC
    About_Updatable_Help

SHORT DESCRIPTION
    Describes the updatable help system in Windows PowerShell.
```

```
LONG DESCRIPTION
…
    Windows PowerShell 3.0 does not come with Help files. You can use
    the Updatable Help feature to install the help files for all of the
    commands that are included by default in Windows PowerShell and for
    all Windows modules.


  UPDATABLE HELP CMDLETS
-- More  --
```

We kunnen de hulp ook tonen in een apart window:

```
Get-Help about_updatable_help -ShowWindow
```



Figuur 3. Help in een apart window

Om een overzicht te krijgen van de volledige inhoud van Help kunnen we op de volgende twee manieren tewerk gaan:

```
Get-Help *
Get-Help -Category All
```

De syntax om hulp te krijgen voor een bepaald commando is ook voor de hand liggend:

```
Get-Help Get-Variable
```

Het resultaat van de hulp over een bepaald commando is onderverdeeld in verschillende secties:

- Name: de naam van het commando
- Synopsis: korte beschrijving van het commando
- Syntax: hoe men het commando moet gebruiken
- Description: meer uitgebreide beschrijving
- Related links: verwijzingen naar gerelateerde informatie
- Remarks: opmerkingen

De syntax sectie is de meest gecompliceerde. Hierin worden de extra parameters beschreven voor een commando:

```
SYNTAX

    Get-Variable [[-Name] <String[]>] [-Exclude <String[]>] [-Include
<String[]>] [-Scope <String>] [-ValueOnly] [<CommonParameters>]
```

Alles wat tussen vierkante haakjes staat is optioneel. In het geval van het commando Get-Variable zijn alle parameters optioneel. Wanneer we Get-Variable oproepen zonder parameters krijgen we een lijst te zien van alle variabelen die gedefinieerd zijn met hun waarden.

```
PS C:\Users\Jef> Get-Variable

Name                           Value
----                           -----
$                              Get_Variable
?                              False
^                              Get_Variable
args                           {}
ConfirmPreference              High
…
```

Een parameter kan bestaan uit alleen een naam. Een voorbeeld hiervan is -ValueOnly. In dat geval spreken we over een switch. Het is een optie die aan- of afstaat. Wanneer we Get-Variable -ValueOnly uitvoeren, krijgen we alleen de Values te zien (zonder de namen).

De standaardwaarde voor een switch parameter is meestal “uit” of “$false”. Wanneer de standaardwaarde “aan” zou zijn, kan men de parameter uitzetten via “$false”:

```
PS C:\Users\Jef> Remove-Item -Confirm:$false test.txt
```

Een parameter kan ook bestaan uit een naam en een waarde. De naam wordt voorafgegaan door een min-teken (“-Scope” is de naam, “global” is de waarde):

```
Get-Variable -Scope global
```

Er kunnen ook optional positional parameters gedefinieerd zijn. Meestal is dit de -Name parameter. Een positional parameter kan men herkennen aan het feit dat de naam van de parameter optioneel is. De volgende twee commando's zijn dus hetzelfde:

```
Get-Variable -Name PID
Get-Variable PID
```

Naast de optional parameters hebben sommige commando's ook mandatory parameters. Die staan niet tussen vierkante haakjes. Ook hier kunnen we weer een onderscheid maken tussen named mandatory parameters en positional mandatory parameters.

Wanneer er waarden zijn gedefinieerd voor een parameter, wordt het type van de waarde vermeld. In het voorbeeld van Get-Variable zijn de meeste parameters (waaronder ook de -Name parameter) arrays van Strings (String[]). De Nederlandse vertaling voor "array van Strings" is lijst van teksten.

De <CommonParameters> zijn de parameters die we bij de meeste commando's kunnen gebruiken.

We kunnen ook hulp opvragen voor een bepaalde parameter:

```
PS C:\Users\Jef> Get-Help Get-Variable -Parameter scope

-Scope <String>
    Gets only the variables in the specified scope. Valid values are
"Global", "Local", or "Script", or a number relative to the current scope
(0 through the number of scopes, where 0 is the current scope and 1 is its
parent). "Local" is the default. For more information, see about_Scopes.

    Required?                    false
    Position?                    named
    Default value                Local
    Accept pipeline input?       false
    Accept wildcard characters?  false
```

De beschrijving van de syntax kan overweldigend zijn, vooral wanneer men een commando voor de eerste maal wil gebruiken. Meestal zijn de voorbeelden dan duidelijker:

```
PS C:\Users\Jef> Get-Help Get-Variable -Examples

NAME
    Get-Variable

SYNOPSIS
    Gets the variables in the current console.


    -------------------------- EXAMPLE 1 --------------------------

    PS C:\>Get-Variable m*
```

```
    This command gets variables with names that begin with the letter "m".
The command also gets the value of the variables.



    -------------------------- EXAMPLE 2 --------------------------

    PS C:\>Get-Variable m* -Valueonly

    This command gets only the values of the variables that have names that
begin with "m".



    -------------------------- EXAMPLE 3 --------------------------

    PS C:\>Get-Variable -Include M*,P*

    This command gets information about the variables that begin with
either the letter "M" or the letter "P".
```

4. PowerShell commandlets

Een PowerShell commandlet (of commando) bestaat uit een werkwoord, gevolgd door een zelfstandig naamwoord. Beiden worden gescheiden door een minteken. Men moet het woord “werkwoord” hier in de ruime zin opvatten. Niet elk werkwoord in PowerShell is ook een echt Engels werkwoord. Men moet werkwoord opvatten als “iets wat een actie impliceert”. Misschien het meest bekende voorbeeld hiervan in PowerShell is het werkwoord “New”. De mogelijke werkwoorden (of *verbs*) worden bepaald door Microsoft. Men kan een lijst opvragen via Get-Verb:

```
PS C:\Users\Jef> Get-Verb

Verb        Group
----        -----
Add         Common
Clear       Common
Close       Common
Copy        Common
Enter       Common
Exit        Common
Find        Common
Format      Common
Get         Common
Hide        Common
Join        Common
Lock        Common
Move        Common
New         Common
…
```

Elk verb behoort tot een bepaalde groep. De common groep bevat de verbs die algemene acties beschrijven die van toepassing kunnen zijn op eender welke commandlet.

Het zelfstandig naamwoord (of *noun*) definieert waarop de commandlet wordt uitgevoerd. Het kan een enkel woord zijn (Get-Variable , Get-Item) of een samengesteld woord (Get-ChildItem, Send-MailMessage).

Wanneer men doorheeft dat een commandlet bestaat uit een verb en een noun, kan men van die informatie gebruik maken om gemakkelijker commandlets terug te vinden. Stel dat we op zoek willen gaan naar commando's die informatie kunnen opvragen over de firewall:

```
PS C:\Users\Jef> Get-Command Get-*Firewall*
CommandType  Name                                    Version    Source
-----------  ----                                    -------    ------
Function     Get-NetFirewallAddressFilter            2.0.0.0    NetSecurity
Function     Get-NetFirewallApplicationFilter        2.0.0.0    NetSecurity
Function     Get-NetFirewallInterfaceFilter          2.0.0.0    NetSecurity
Function     Get-NetFirewallInterfaceTypeFilter      2.0.0.0    NetSecurity
Function     Get-NetFirewallPortFilter               2.0.0.0    NetSecurity
Function     Get-NetFirewallProfile                  2.0.0.0    NetSecurity
Function     Get-NetFirewallRule                     2.0.0.0    NetSecurity
Function     Get-NetFirewallSecurityFilter           2.0.0.0    NetSecurity
Function     Get-NetFirewallServiceFilter            2.0.0.0    NetSecurity
Function     Get-NetFirewallSetting                  2.0.0.0    NetSecurity
```

5. Confirm, WhatIf en Force parameters

Confirm, WhatIf en Force zijn drie parameters die gebruikt worden met verbs die bestanden, variabelen, data, … veranderen. We komen ze dus meestal tegen bij verbs zoals Set of Remove.

De -Confirm parameter wordt gebruikt om bevestiging te vragen:

```
PS C:\Users\Jef> Set-Location $env:TEMP
PS C:\Users\Jef\AppData\Local\Temp> New-Item dummy.txt


    Directory: C:\Users\Jef\AppData\Local\Temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         9/04/2018     10:12              0 dummy.txt


PS C:\Users\Jef\AppData\Local\Temp> Remove-Item .\dummy.txt -Confirm

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove File" on target
"C:\Users\Jef\AppData\Local\Temp\dummy.txt".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"):
```

De meeste van de antwoorden zijn eenvoudig te begrijpen. Alleen *Suspend* vraagt misschien wat meer uitleg. Wanneer men eerst enkele zaken zou willen controleren voordat men beslist om het bestand te verwijderen, kan men S (Suspend) kiezen.

```
Confirm
```

```
Are you sure you want to perform this action?
Performing the operation "Remove File" on target
"C:\Users\Jef\AppData\Local\Temp\dummy.txt".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"): S
PS C:\Users\Jef\AppData\Local\Temp>>
```

We krijgen vervolgens een *continuation prompt* te zien (>>). We kunnen nu meer informatie opvragen via die prompt. Wanneer we klaar zijn om een beslissing te nemen, kunnen we terugkeren naar de Confirm-prompt via “exit”.

```
PS C:\Users\Jef\AppData\Local\Temp>> exit

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove File" on target
"C:\Users\Jef\AppData\Local\Temp\dummy.txt".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"): Y
```

De -WhatIf parameter vertelt ons wat het commando doet, zonder dat het commando wordt uitgevoerd. Voor eenvoudige commando’s heeft dat niet veel meerwaarde, maar wanner we bijvoorbeeld wildcards gebruiken, kan dat een aantal neveneffecten van een commando duidelijker maken:

```
PS C:\Users\Jef\AppData\Local\Temp> Remove-Item * -WhatIf
What if: Performing the operation "Remove File" on target
"C:\Users\Jef\AppData\Local\Temp\URLE9A.tmp".
What if: Performing the operation "Remove File" on target
"C:\Users\Jef\AppData\Local\Temp\URLEED3.tmp".
What if: Performing the operation "Remove File" on target
"C:\Users\Jef\AppData\Local\Temp\URLFECA.tmp".
What if: Performing the operation "Remove File" on target
"C:\Users\Jef\AppData\Local\Temp\wct3E8D.tmp".
What if: Performing the operation "Remove File" on target
"C:\Users\Jef\AppData\Local\Temp\wct5E5F.tmp".
What if: Performing the operation "Remove File" on target
"C:\Users\Jef\AppData\Local\Temp\wct797A.tmp".
What if: Performing the operation "Remove File" on target
"C:\Users\Jef\AppData\Local\Temp\wct97F9.tmp".
What if: Performing the operation "Remove File" on target
"C:\Users\Jef\AppData\Local\Temp\wct97FA.tmp".
What if: Performing the operation "Remove File" on target
"C:\Users\Jef\AppData\Local\Temp\wctA6F.tmp".
What if: Performing the operation "Remove File" on target
"C:\Users\Jef\AppData\Local\Temp\wctC7FC.tmp".
What if: Performing the operation "Remove File" on target
"C:\Users\Jef\AppData\Local\Temp\wctD839.tmp".
What if: Performing the operation "Remove File" on target
"C:\Users\Jef\AppData\Local\Temp\wctFF81.tmp".
What if: Performing the operation "Remove File" on target
"C:\Users\Jef\AppData\Local\Temp\wctFF82.tmp".
```

De .tmp bestanden zijn in dit geval niet verwijderd. Maar wanneer we het commando zonder -WhatIf zouden uitvoeren, zouden al die bestanden wel verdwenen zijn.

De -Force parameter zorgt ervoor dat we niet meer beschermd worden tegen onszelf en dat we mogelijk domme dingen zullen kunnen uitvoeren. Standaard zal het New-Item commando niet toelaten dat er een tweede bestand met dezelfde naam kan gecreëerd worden en het oorspronkelijke bestand overschreven kan worden:

```
PS C:\Users\Jef\AppData\Local\Temp> New-Item dummy.txt


    Directory: C:\Users\Jef\AppData\Local\Temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         9/04/2018     10:26              0 dummy.txt


PS C:\Users\Jef\AppData\Local\Temp> New-Item dummy.txt
New-Item : Het bestand C:\Users\Jef\AppData\Local\Temp\dummy.txt bestaat
al.
At line:1 char:1
+ New-Item dummy.txt
+ ~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : WriteError:
(C:\Users\Jef\Ap...\Temp\dummy.txt:String) [New-Item], IOException
    + FullyQualifiedErrorId :
NewItemIOError,Microsoft.PowerShell.Commands.NewItemCommand
```

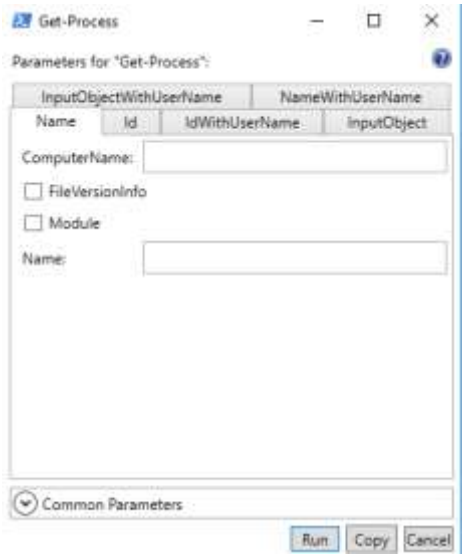
Wanneer we hetzelfde commando uitvoeren met de -Force parameter overschrijft het tweede commando het bestand dummy.txt:

```
PS C:\Users\Jef\AppData\Local\Temp> New-Item dummy.txt


    Directory: C:\Users\Jef\AppData\Local\Temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         9/04/2018     10:28              0 dummy.txt


PS C:\Users\Jef\AppData\Local\Temp> New-Item dummy.txt -Force


    Directory: C:\Users\Jef\AppData\Local\Temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         9/04/2018     10:28              0 dummy.txt
```

6. Show-Command

De Show-Command cmdlet kan gebruikt wordt als hulp om een commando uit te voeren. Er wordt een window getoond waarin men de parameters van het commando kan invullen:

```
Show-Commando Get-Process
```



Figuur 4. Show-Command window

7. Aliassen

Een alias is een alternatieve naam voor een commandlet. Een commandlet kan meer dan één alias hebben:

```
PS C:\Users\Jef> Get-Alias

CommandType     Name                                Version    Source
-----------     ----                                -------    ------
Alias           % -> ForEach-Object
Alias           ? -> Where-Object
Alias           ac -> Add-Content
Alias           asnp -> Add-PSSnapin
Alias           cat -> Get-Content
Alias           cd -> Set-Location
…
```

Een alias kan handig zijn om het typwerk te verminderen of om de overgang naar PowerShell wat gemakkelijker te maken. Een voorbeelde van dit laatste voordeel is de “cd” alias. In de oude command prompt omgeving was dit het commando om de standaard directory te wijzigen. In PowerShell moeten we daarvoor het Set-Location commando gebruiken.

We kunnen een nieuwe alias definiëren met de New-Alias commandlet:

```
PS C:\Users\Jef> New-Alias grep -Value Select-String
```

8. PowerShell Pipeline

De PowerShell Pipeline is een van de meest belangrijke en meest krachtige onderdelen van PowerShell. Wanneer je vroeger al met de command-prompt van Windows hebt gewerkt,

ken je dit concept misschien al. Wanneer je uit een Linux omgeving afkomstig bent, ken je dit concept zeker al. Het kadert in de UNIX filosofie “Do one thing and do it well”.

In plaats van een commandlet continu uit te breiden met extra mogelijkheden, zorgt men ervoor dat men verschillende commandlets heeft die verschillende taken kunnen uitvoeren. Al die commandlets werken samen om een bepaald eindresultaat te bereiken.

We nemen als voorbeeld de Get-Process commandlet. Deze commandlet geeft een lijst terug van alle processen die op de lokale (of remote) computer worden uitgevoerd. We kunnen optioneel ook de naam van een process meegeven:

```
PS C:\Users\Jef> Get-Process powershell

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
    939      48   202712     136244      45,22   9376   1 powershell
```

We krijgen hier informatie te zien over het Powershell proces dat wordt uitgevoerd op deze machine. Maar is dit alles wat we kunnen weten over dit proces-object? PowerShell heeft ervoor gezorgd dat de informatie over het object op één regel past. Maar wanneer we de informatie onder elkaar zouden zetten, kunnen we veel meer informatie zien.

In plaats van een extra parameter te voorzien bij de Get-Process commandlet, bestaat er een aparte commandlet in Powershell om de eigenschappen van een object onder elkaar te tonen: Format-List. (alias: fl). We kunnen de output van de Get-Process commandlet doorgeven (“pipen”) naar Format-List. Beide commando’s doen hier één ding: Get-Process maakt een lijst van Proces objecten en Format-List toont de informatie van een object in een lijst. Het symbool om de output van het ene commando door te geven aan een volgend commando is | :

```
PS C:\Users\Jef> Get-Process powershell | Format-List *


Name                       : powershell
Id                         : 9376
PriorityClass              : Normal
FileVersion                : 10.0.16299.15 (WinBuild.160101.0800)
HandleCount                : 836
WorkingSet                 : 139534336
PagedMemorySize            : 207581184
PrivateMemorySize          : 207581184
VirtualMemorySize          : 919855104
TotalProcessorTime         : 00:00:45.3906250
SI                         : 1
Handles                    : 836
VM                         : 2199943110656
WS                         : 139534336
PM                         : 207581184
NPM                        : 49224
Path                       :
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Company                    : Microsoft Corporation
```

```
CPU                            : 45,390625
…
```

Het sterretje achter Format-List wil zeggen dat we alle eigenschappen van het proces willen zien. Het is een voorbeeld van een optional positional parameter:

```
PS C:\Users\Jef> Get-Help Format-List

NAME
    Format-List

SYNOPSIS
    Formats the output as a list of properties in which each property
appears on a new line.


SYNTAX
    Format-List [[-Property] <Object[]>] [-DisplayError] [-Expand <String>]
[-Force] [-GroupBy <Object>] [-InputObject <PSObject>] [-ShowError] [-View
<String>] [<CommonParameters>]
```

8. A. Paging

De meeste mensen die met de DOS-prompt hebben gewerkt, kennen "|more". Dat is het commando dat we gebruiken om informatie *per pagina* te tonen. We kunnen dit ook gebruiken in PowerShell, maar het is niet de meest efficiënte manier om informatie per pagina te tonen. Wanneer we |more gebruiken, moet PowerShell eerst de volledige output van het vorige commando in de pipeline ophalen. Vervolgens wordt die output aan 'more' gegeven om per pagina getoond te worden. Dat is in feite hoe er ook in de DOS-prompt werd gewerkt.

Het concept van een pipeline is dat informatie die in de pipeline wordt gestoken ogenblikkelijk verwerkt wordt door het volgende commando in de pipeline. Wanneer we paging willen gebruiken op de PowerShell manier, moeten we de Out-Host cmdlet gebruiken:

```
PS C:\Users\Jef> Get-Process |Out-Host -Paging
201      12     2576      10856       0,20 105532  10 HPHotkeyNotification
442      48    21424       4572       1,53   2068   0 HPJumpStartBridge
180      12     3844       1072       0,13 270488  10 HPJumpStartLaunch
<SPACE> next page; <CR> next line; Q quit
195      11     1932       1768       0,33   2416   0 hpMAMSrv
<SPACE> next page; <CR> next line; Q quit
221      15     3816      11664       0,14 214096  10 HPNotifications
```

Nadelen van Out-Host zijn dat er niet kan worden teruggescrolld (maar daarvoor kunnen we de verticale schuifbalk gebruiken) en dat de output onderbreken met 'Q' een foutmelding geeft:

```
The command was stopped by the user.
At line:1 char:1
+ get-process |Out-Host -Paging
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : OperationStopped: (:) [], HaltCommandException
```

```
    + FullyQualifiedErrorId : System.Management.Automation.HaltCommandException
```

De alias voor Out-Host is “oh”.

9. Objecten in PowerShell

9. A. Properties

PowerShell is een objectgeoriënteerde programmeertaal. Daarin lijkt het op andere moderne computertalen zoals C# of Java. We zijn het concept “object” al tegengekomen in de vorige voorbeelden over Format-List. De Format-List commandlet toont de eigenschappen van een object in een lijst. Daarmee weten we dus dat een object eigenschappen moet hebben. Die eigenschappen noemen we ook de properties. We kunnen een overzicht krijgen van alle members van een object via Get-Member:

```
PS C:\Users\Jef> Get-Process powershell |Get-Member -MemberType Property


   TypeName: System.Diagnostics.Process

Name                       MemberType Definition
----                       ---------- ----------
BasePriority               Property   int BasePriority {get;}
Container                  Property   System.ComponentModel.IContainer
Container {get;}
EnableRaisingEvents        Property   bool EnableRaisingEvents {get;set;}
ExitCode                   Property   int ExitCode {get;}
ExitTime                   Property   datetime ExitTime {get;}
Handle                     Property   System.IntPtr Handle {get;}
HandleCount                Property   int HandleCount {get;}
HasExited                  Property   bool HasExited {get;}
Id                         Property   int Id {get;}
MachineName                Property   string MachineName {get;}
…
```

We kunnen de Id van een process opvragen via de .-notatie:

```
PS C:\Users\Jef> (Get-Process powershell).Id
9376
```

Door ronde haakje te gebruiken, voeren we eerst het commando “Get-Process powershell” uit. Het resultaat daarvan is een process object. Dat object heeft een property Id die we kunnen opvragen via de .-notatie. We hadden dit ook in twee stappen kunnen schrijven:

```
PS C:\Users\Jef> $procPowerShell = Get-Process powershell
PS C:\Users\Jef> $procPowerShell.Id
9376
```

In dit geval bewaren we het proces object in een variabele ($procPowerShell). Ook op dat object kunnen we de .-notatie gebruiken.

Een property is op zijn beurt ook weer een object:

```
PS C:\Users\Jef\AppData\Local\Temp> (Get-Process powershell).StartTime|Get-
Member -MemberType Property


   TypeName: System.DateTime

Name        MemberType Definition
----        ---------- ----------
Date        Property   datetime Date {get;}
Day         Property   int Day {get;}
DayOfWeek   Property   System.DayOfWeek DayOfWeek {get;}
DayOfYear   Property   int DayOfYear {get;}
Hour        Property   int Hour {get;}
Kind        Property   System.DateTimeKind Kind {get;}
Millisecond Property   int Millisecond {get;}
Minute      Property   int Minute {get;}
Month       Property   int Month {get;}
Second      Property   int Second {get;}
Ticks       Property   long Ticks {get;}
TimeOfDay   Property   timespan TimeOfDay {get;}
Year        Property   int Year {get;}
```

De StartTime van een proces is een “DateTime”. Wanneer we dus zouden willen weten op welke dag van de week het powershell process gestart is:

```
PS C:\Users\Jef> $procPowerShell.StartTime.DayOfWeek
Monday
```

9. B. Methodes

Naast properties kunnen objecten ook methodes hebben:

```
PS C:\Users\Jef> (Get-Process powershell).StartTime|Get-Member -MemberType
Method


   TypeName: System.DateTime

Name                 MemberType Definition
----                 ---------- ----------
Add                  Method     datetime Add(timespan value)
AddDays              Method     datetime AddDays(double value)
AddHours             Method     datetime AddHours(double value)
AddMilliseconds      Method     datetime AddMilliseconds(double value)
AddMinutes           Method     datetime AddMinutes(double value)
AddMonths            Method     datetime AddMonths(int months)
AddSeconds           Method     datetime AddSeconds(double value)
AddTicks             Method     datetime AddTicks(long value)
AddYears             Method     datetime AddYears(int value)
…
ToLongDateString     Method     string ToLongDateString()
ToLongTimeString     Method     string ToLongTimeString()
```

Een methode kan gebruikt worden om de toestand van een object te veranderen of om het te converteren in iets anders. Een DateTime is bijvoorbeeld geen String. Maar we kunnen het wel omzetten naar een String met de methode ToLongDateString. Om een methode te gebruiken, moeten we ze uitvoeren. Daarvoor gebruiken we de ronde haakjes:

```
PS C:\Users\Jef> $procPowerShell.StartTime.ToLongDateString()
maandag 9 april 2018
```

In dit geval was er geen extra informatie nodig om de omzetting uit te voeren. Voor sommige methodes is dat wel nodig. Daarom kunnen we ook argumenten meegeven tussen de ronde haakjes. Stel dat we de datum zouden willen weten waarop het powershell proces al 7 dagen bezig zal zijn:

```
PS C:\Users\Jef> $procPowerShell.StartTime.AddDays(7)

maandag 16 april 2018 6:29:11
```

Aangezien dit ook weer een DateTime is, zouden we opnieuw de ToLongDateString() methode kunnen uitvoeren:

```
PS C:\Users\Jef> $procPowerShell.StartTime.AddDays(7).ToLongDateString()
maandag 16 april 2018
```

9. C. Overloaded methodes

Voor sommige methodes hebben we verschillende versies. Dat wil zeggen dat ze een verschillende set argumenten hebben. Methodes in een object die dezelfde naam hebben, maar verschillende argumenten, noemen we ook overloaded methodes. We vinden er eentje terug in de String klasse. (De name van een proces is een String)

```
PS C:\Users\Jef> $procPowerShell.Name.Substring(2)
wershell
```

De Substring() methode van een String geeft de string terug die begint op een startindex. Maar er is nog een andere, overloaded versie. We kunnen een lijst van alle overloaded versies opvragen door de ronde haakjes weg te laten:

```
PS C:\Users\Jef> $procPowerShell.Name.Substring

OverloadDefinitions
-------------------
string Substring(int startIndex)
string Substring(int startIndex, int length)
```

Stel dat we de eerste 5 letters van de naam van het process willen zien:

```
PS C:\Users\Jef> $procPowerShell.Name.Substring(0, 5)
power
```

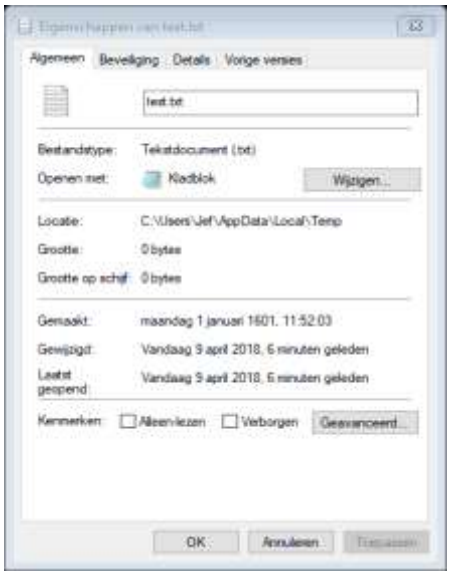
9. D. Read/write vs readonly

Properties van een object kunnen readonly zijn of read/write. In het eerste geval kunnen we ze alleen gebruiken om een waarde op te vragen. In het tweede geval kunnen we ze ook gebruiken om de waarde te wijzigen.

```
PS C:\Users\Jef> Set-Location $env:TEMP
```

```
PS C:\Users\Jef\AppData\Local\Temp> $bestand=New-Item test.txt
PS C:\Users\Jef\AppData\Local\Temp> $bestand | Get-Member -MemberType
Property


   TypeName: System.IO.FileInfo

Name              MemberType Definition
----              ---------- ----------
Attributes        Property   System.IO.FileAttributes Attributes {get;set;}
CreationTime      Property   datetime CreationTime {get;set;}
CreationTimeUtc   Property   datetime CreationTimeUtc {get;set;}
Directory         Property   System.IO.DirectoryInfo Directory {get;}
DirectoryName     Property   string DirectoryName {get;}
Exists            Property   bool Exists {get;}
Extension         Property   string Extension {get;}
FullName          Property   string FullName {get;}
IsReadOnly        Property   bool IsReadOnly {get;set;}
LastAccessTime    Property   datetime LastAccessTime {get;set;}
LastAccessTimeUtc Property   datetime LastAccessTimeUtc {get;set;}
LastWriteTime     Property   datetime LastWriteTime {get;set;}
LastWriteTimeUtc  Property   datetime LastWriteTimeUtc {get;set;}
Length            Property   long Length {get;}
Name              Property   string Name {get;}
```

We zien hier dat de naam en de lengte van het bestand readonly zijn. Ze hebben alleen een “getter” om de waarde op te vragen. De CreationTime is echter read/write want de property heeft een “getter” en een “setter”.



Figuur 5. Een bestand uit de 17de eeuw

```
PS C:\Users\Jef> $langgeleden = Get-Date -Day 1 -Month 1 -Year 1601
PS C:\Users\Jef> $bestand.CreationTime = $langgeleden
```

Het resultaat is niet geloofwaardig, maar het laat wel zien dat de CreationTime een read/write property is. Aangezien de property Name readonly is, kunnen we de naam van een bestand niet wijzigen door de property te veranderen:

```
PS C:\Users\Jef\AppData\Local\Temp> $bestand.Name="testje.txt"
'Name' is a ReadOnly property.
At line:1 char:1
+ $bestand.Name="testje.txt"
+ ~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (:) [], RuntimeException
    + FullyQualifiedErrorId : PropertyAssignmentException
```

Om een bestand te hernoemen, moeten we Rename-Item gebruiken:

```
PS C:\Users\Jef\AppData\Local\Temp> Rename-Item -Path $bestand -NewName
"testje.txt"
PS C:\Users\Jef\AppData\Local\Temp> $bestand.Name
test.txt
PS C:\Users\Jef\AppData\Local\Temp> $bestand2 = Get-Item testje.txt
PS C:\Users\Jef\AppData\Local\Temp> $bestand2.CreationTime

maandag 1 januari 1601 11:52:03
```

Rename-Item heeft de Name-property van de bestaande variabele niet gewijzigd. Maar het bestand op de harde schijf is echter wel veranderd. We kunnen het opvragen via Get-Item en daar zien we dat het bestand dezelfde creatie datum heeft.

10. Lijsten van objecten(collections)

Wanneer een object een collection is, kunnen we die lijst gemakkelijk opvragen. Als voorbeeld maken we een aantal bestanden in de subdirectory:

```
PS C:\Users\Jef> $folder=New-Item subdir -ItemType Directory

PS C:\Users\Jef> New-Item -Path $folder -Name test1.txt


    Directory: C:\Users\Jef\subdir


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        9/04/2018     12:18              0 test1.txt


PS C:\Users\Jef> New-Item -Path $folder -Name test2.txt


    Directory: C:\Users\Jef\subdir


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        9/04/2018     12:18              0 test2.txt
```

De folder subdir bevat nu twee bestanden:

```
PS C:\Users\Jef> $items=Get-ChildItem $folder
PS C:\Users\Jef> $items
```

```
    Directory: C:\Users\Jef\subdir


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        9/04/2018     12:18              0 test1.txt
-a----        9/04/2018     12:18              0 test2.txt
```

De variabele $items vermelden is al genoeg om de lijst te zien van de bestanden in de folder. Wanneer we bepaalde acties willen uitvoeren op alle elementen in een collection kunnen we de ForEach-Object commandlet gebruiken

10. A. ForEach-Object

ForEach-Object kan op twee verschillende manieren gebruikt worden:

- We kunnen de naam van een property of een method gebruiken
- We kunnen een scriptblock schrijven

In het eerste geval vermelden we de naam van de property:

```
PS C:\Users\Jef> $items | ForEach-Object  FullName
C:\Users\Jef\AppData\Local\Temp\subdir\test1.txt
C:\Users\Jef\AppData\Local\Temp\subdir\test2.txt
```

In het tweede geval wordt het scriptblock uitgevoerd voor elk van de items in de collection. We moeten in het block kunnen verwijzen naar het item dat nu verwerkt wordt (het huidige item). Dat doen we via “$_”.

```
PS C:\Users\Jef> $items | ForEach-Object  {$_.Name + ":" + $_.Length/1024}
test1.txt:0
test2.txt:0
```

10. B. Where-Object

De commandlet Where-Object laat toe om te filteren. Wanneer we alleen de bestanden willen zien die gecreëerd zijn na vandaag om 12:18:25:

```
PS C:\Users\Jef> $items |Where-Object CreationTime -gt (Get-Date 12:18:25)


    Directory: C:\Users\Jef\AppData\Local\Temp\subdir


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        9/04/2018     12:18              0 test2.txt
```

Ook hier kunnen we een scriptblock gebruiken. Dat is vooral handig wanneer we meer dan één voorwaarde willen definiëren:

```
PS C:\Users\jef> $items | Where-Object {$_.CreationTime -gt (Get-Date
12:18:25) -and $_.Name  -Like "test*"}



    Directory: C:\Users\Jef\AppData\Local\Temp\subdir


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        9/04/2018     12:18              0 test2.txt
```

In dit geval zien we een lijst met bestanden die na 12:18:25 zijn gemaakt en waarvan de naam begint met “test”. Net zoals bij ForEach-Object verwijzen we naar de objecten via $_.

10. C. Sort-Object

Wanneer we de items willen sorteren, kunnen we gebruik maken van Sort-Object:

```
PS C:\Users\Jef> $items |Sort-Object CreationTime


    Directory: C:\Users\Jef\AppData\Local\Temp\subdir


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        9/04/2018     12:18              0 test1.txt
-a----        9/04/2018     12:18              0 test2.txt


PS C:\Users\Jef> $items |Sort-Object CreationTime -Descending


    Directory: C:\Users\Jef\AppData\Local\Temp\subdir


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        9/04/2018     12:18              0 test2.txt
-a----        9/04/2018     12:18              0 test1.txt
```

10. D. Measure-Object

De Measure-Object cmdlet creëert een GenericMeasureInfo object. Het wordt gebruikt om het aantal, gemiddelde, som, minimum en/of maximum te berekenen. Wanneer we willen weten hoeveel processen er actief zijn op een computer:

```
PS C:\Users\jef> Get-Process |Measure-Object


Count    : 60
Average  :
Sum      :
Maximum  :
Minimum  :
Property :
```

Er zijn 60 processen actief. De andere waarden zijn niet ingevuld. Wanneer we het getal "60" zouden willen gebruiken:

```
PS C:\Users\jef> (Get-Process |Measure-Object).Count
60
```

Standaard wordt alleen het aantal berekend. Wanneer we de gemiddelde CPU-tijd willen weten, kan dat op de volgende manier:

```
PS C:\Users\jef> Get-Process |Measure-Object CPU -Average


Count    : 60
Average  : 19,3270833333333
Sum      :
Maximum  :
Minimum  :
Property : CPU
```

We kunnen ook meerdere berekeningen ineens laten uitvoeren:

```
PS C:\Users\jef> Get-Process |Measure-Object CPU -Average -Sum -Minimum
-Maximum


Count    : 62
Average  : 18,7532762096774
Sum      : 1162,703125
Maximum  : 620,78125
Minimum  : 0,03125
Property : CPU
```

Voor teksten hebben gemiddelde, som, minimum en maximum geen zin. We kunnen Measure-Object ook gebruiken om het aantal lijnen, woorden of letters te zien:

```
PS C:\Users\jef> Get-Content C:\windows\Logs\CBS\CBS.log|Measure-Object
-Line -Word -Character

Lines  Words Characters Property
-----  ----- ---------- --------
15728 259682    3620717
```

10. E. Select-Object

De Select-Object cmdlet kan gebruikt worden om:

1. De eerste/laatste X rijen te selecteren.
2. Bepaalde properties te verwijderen.
3. Met een "uitgepakte" property werken.

Wanneer we bijvoorbeeld de drie processen die de meeste CPU tijd gebruikt hebben willen stoppen, kunnen we het volgende commando gebruiken (omdat we ze niet werkelijk willen stoppen, gebruiken we de -WhatIf switch):

```
PS C:\Users\jef> Get-Process | Sort-Object CPU |Select-Object -Last 3|Stop-
Process -WhatIf
What if: Performing the operation "Stop-Process" on target "System (4)".
What if: Performing the operation "Stop-Process" on target "MsMpEng
(1316)".
What if: Performing the operation "Stop-Process" on target "svchost (940)".
```

De tweede toepassing van Select-Object houdt in dat we een afgeslankte versie van een object willen creëren:

```
PS C:\Users\jef> (Get-Process|Get-Member -MemberType Properties|Measure-
Object).Count
67
PS C:\Users\jef> (Get-Process|Select-Object CPU, Name| Get-Member -
MemberType Properties|Measure-Object).Count
2
```

In het eerste voorbeeld zien we dat een process-object 67 properties heeft. In het tweede voorbeeld heeft Select-Object een object gemaakt met twee properties. Een praktische toepassing hiervan is het bewaren van gegevens in een .CSV-bestand. Wanneer we de output van de proceslijst exporteren naar een .CSV-bestand, worden alle properties weggeschreven. Wanneer we maar een beperkte lijst properties willen bewaren, gebruiken we Select-Object:

```
PS C:\Users\jef> Get-Process|Select-Object CPU,Name,VirtualMemorySize
|Export-Csv proces.csv
PS C:\Users\jef> Get-Content proces.csv -First 4
#TYPE Selected.System.Diagnostics.Process
"CPU","Name","VirtualMemorySize"
"0,3125","ApplicationFrameHost","154583040"
"0,03125","conhost","59842560"
```

Een laatste toepassing van Select-Object is het “uitpakken” van een property. Stel dat we willen weten hoeveel RequiredServices er zijn voor een bepaalde service. Het volgende commando geeft niet het juiste resultaat terug:

```
PS C:\Users\jef> Get-Service LanManServer|Measure-Object RequiredServices


Count    : 1
Average  :
Sum      :
Maximum  :
Minimum  :
Property : RequiredServices
```

Met de -ExpandProperty parameter, pakken we de RequiredServices uit zodat het afzonderlijke waarden worden die gemeten kunnen worden door Measure-Object:

```
PS C:\Users\jef> Get-Service LanManServer|Select-Object -ExpandProperty
RequiredServices |Measure-Object
```

```
Count    : 2
Average  :
Sum      :
Maximum  :
Minimum  :
Property :
```

We kunnen dit ook gebruiken om de afzonderlijke services te zien, zodat we bijvoorbeeld kunnen controleren of ze gestart zijn:

```
PS C:\Users\jef> Get-Service LanManServer|Select-Object -ExpandProperty
RequiredServices

Status   Name               DisplayName
------   ----               -----------
Running  SamSS              Security Accounts Manager
Running  Srv2               Server SMB 2.xxx Driver
```

10. F. Group-Object

Door items te groeperen kunnen we bijvoorbeeld zien hoeveel services er running zijn en hoeveel er gestopt zijn:

```
PS C:\Users\jef> Get-Service|Group-Object Status

Count Name                      Group
----- ----                      -----
  108 Stopped                   {AJRouter, ALG, AppIDSvc, AppMgmt...}
   83 Running                   {Appinfo, BFE, BrokerInfrastructure,
CDPSvc...}
```

De Group-property bevat de lijst van de services in een groep. Wanneer we niet geïnteresseerd zijn in die lijst, kunnen we de Group weglaten met de -NoElement parameter:

```
PS C:\Users\jef> Get-Service|Group-Object Status -NoElement

Count Name
----- ----
  108 Stopped
   83 Running
```

10. G. Compare-Object

De Compare-Object cmdlet vergelijkt twee sets en toont de verschillen. Stel dat we willen zien welke services gestopt zijn sinds een referentiemoment:

```
PS C:\Users\jef> $a=Get-Service|Where-Object status -eq running
PS C:\Users\jef> Stop-Service DnsCache
PS C:\Users\jef> Get-Service|Where-Object status -eq running |Compare-
Object -ReferenceObject $a

InputObject SideIndicator
----------- -------------
Dnscache    <=
```

We beginnen met de lijst van gestarte processen te bewaren in de variabele $a. Vervolgens stoppen we een proces. Wanneer we de lijst van gestarte services doorgeven aan Compare-Object zien we dat Dnscache aanwezig is in de referentielijst en niet in de nieuwe lijst. De SideIndicator geeft aan dat het element voorkomt in de referentielijst. Wanneer het object alleen aanwezig zou zijn in het DifferenceObject zouden we "=>" zien.

10. H. Foreach-Object

De Foreach-Object cmdlet voert een scriptblock uit voor een reeks waarden. Het kan bijvoorbeeld handig zijn wanneer we een aantal dummy waarden willen creëren:

```
PS C:\Users\jef> 1..10|Foreach-Object  {New-Item "bestand$_.txt"}
```

We beginnen met de getallen van 1 tot 10 te genereren. Elk van die getallen wordt doorgegeven aan de New-Item cmdlet om een bestand te maken.

We kunnen dit uitbreiden door eerst een subdirectory te maken waarin de bestanden worden gezet. De Foreach-Object cmdlet kan een script uitvoeren alvorens de items overlopen worden met behulp van de -Begin parameter:

```
PS C:\Users\jef> 1..10|Foreach-Object -Begin{$folder=New-Item subdir
-ItemType Directory;Set-Location $folder } -Process  {New-Item
"bestand$_.txt"}
```

10. I. Tee-Object

Tee-Object maakt een aftakking in de pipeline. We kunnen een tussentijds resultaat bewaren in een variabele of in een bestand:

```
PS C:\Users\jef> Get-Service|Tee-Object -Variable a |Where-Object Status
-eq running| Out-File serv.txt
```

In dit voorbeeld wordt de lijst van services bewaard in de variabele $a. De running services worden weggeschreven naar het bestand serv.txt

11. Remote sessions

Een aantal cmdlets hebben een -ComputerName parameter. Wanneer de firewall "File and Print Sharing" toelaat, kunnen we informatie opvragen van een remote server:

```
PS C:\Users\jef> Get-Service -ComputerName server214|Select-Object -First 3
|ft Status,Name, DisplayName, MachineName -AutoSize

 Status Name     DisplayName                          MachineName
 ------ ----     -----------                          -----------
Stopped AJRouter AllJoyn Router Service               server214
Stopped ALG      Application Layer Gateway Service    server214
Stopped AppIDSvc Application Identity                 server214
```

Maar het is ook relatief eenvoudig om powershell commando's uit te voeren op een andere machine. Aangezien remote Powershell gebruik maakt van WinRM moet die service

geactiveerd zijn en moet de firewall dit soort connectie toelaten. Hiervoor kan men Enable-PSRemoting gebruiken. Dit voert de volgende acties uit:

- Voert Set-WSManQuickConfig uit
  - Start WinRM service
  - Zet het startup type van de WinRM service op “automatic”
  - Maakt een “listener” om via http requests op eender welk IP-adres te ontvangen
  - Voegt een regel toe in de firewall om WinRM verkeer te ontvangen
- Herstart de WinRM service

We kunnen testen of WinRM sessies mogelijk zijn:

```
PS C:\Users\jef> Test-WSMan server214


wsmid           :
http://schemas.dmtf.org/wbem/wsman/identity/1/wsmanidentity.xsd
ProtocolVersion : http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd
ProductVendor   : Microsoft Corporation
ProductVersion  : OS: 0.0.0 SP: 0.0 Stack: 3.0
```

Wanneer we vanuit een Domain computer een sessie proberen te openen op een Workgroup computer, kan dit problemen opleveren. Om dit toe te laten, moeten we de Workgroup computer toevoegen aan de TrustedHosts:

```
PS C:\Users\jef> Set-Item WSMAN:\Localhost\Client\TrustedHosts -Value
testdsc2 -Force
```

In plaats van een specifieke naam, kunnen we natuurlijk ook een wildcard meegeven:

```
PS C:\Users\jef> Set-Item WSMAN:\Localhost\Client\TrustedHosts -Value * -
Force
```

Vervolgens kunnen we een remote session opstarten:

```
PS C:\> $cred=Get-Credential

cmdlet Get-Credential at command pipeline position 1
Supply values for the following parameters:
Credential
PS C:\> Enter-PSSession server214 -Credential $cred
[server214]: PS C:\Users\Jef\Documents> Get-ADUser -Filter *


DistinguishedName : CN=Jef,CN=Users,DC=dom213,DC=lokaal
Enabled           : True
GivenName         :
Name              : Jef
ObjectClass       : user
ObjectGUID        : 226518f5-d8c3-4ce9-bb09-71203718cbb8
```

```
SamAccountName     : Jef
SID                : S-1-5-21-517424593-1313819611-2499439019-500
Surname            :
…
```

Op Server214 is de Active Directory module voor Powershell geïnstalleerd. We kunnen dus AD-commando's uitvoeren. Wanneer we de remote sessie verlaten, zien we dat de AD-module niet op de client geïnstalleerd is. In een remote sessie krijgen we dus toegang tot de Powershell omgeving van de server:

```
[server214]: PS C:\Users\Jef\Documents> Exit-PSSession
PS C:\Users\jef> Get-ADUser
Get-ADUser : The term 'Get-ADUser' is not recognized as the name of a
cmdlet, function, script file, or operable
program. Check the spelling of the name, or if a path was included, verify
that the path is correct and try again.
At line:1 char:1
+ Get-ADUser
+ ~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (Get-ADUser:String) [],
CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException
```

Wanneer we een commando willen uitvoeren op meerdere computers is een interactieve prompt niet de beste oplossing. Via Invoke-Command kunnen we een commando uitvoeren op meerdere computers:

```
PS C:\Users\admin.DOM213> Invoke-Command Server213,Server214 -ScriptBlock
{New-Item c:\ikwashier.txt}


    Directory: C:\


Mode           LastWriteTime    Length Name            PSComputerName
----           -------------    ------ ----            --------------
-a----  12/14/2018   1:16 PM         0 ikwashier.txt   Server213
-a----  12/14/2018   1:16 PM         0 ikwashier.txt   Server214
```

12. Modules en snapins

In PowerShell 1 werden snapins gebruikt om de mogelijkheden van Powershell uit te breiden. Een snapin is een Dynamic Link Library (DLL). Een DLL kan op verschillende systemen geïnstalleerd worden. We kunnen lijstje van de snapins opvragen:

```
PS C:\> Get-PSSnapin


Name        : Microsoft.PowerShell.Core
PSVersion   : 5.1.14393.2248
Description : This Windows PowerShell snap-in contains cmdlets used to
manage components of Windows PowerShell.
```

In de huidige versie van Powershell is er standaard maar 1 snapin geïnstalleerd namelijk de Powershell core. Vanaf versie 2 is men namelijk afgestapt van het systeem van de snapins. Het gebruik ervan was misschien eenvoudig, maar zelf eigen snapins creëren was minder voor de hand liggend. Een snapin (DLL) moet immers gecompileerd worden.

Vandaar dat het module systeem werd geïntroduceerd. In de eenvoudigste vorm is een module een verzameling van items die in een Powershell sessie kunnen gebruikt worden. Dat kunnen commandlets, providers, functies, aliassen, … zijn. Er zijn in feite twee soorten modules: script module en binary modules.

Een script module is een script met een .psm1 extensie. We kunnen de module gebruiken in een sessie via Import-Module. Alle functies in het scripts worden toegankelijk. Wanneer we dat niet willen, kunnen we in het modulebestand de functie Export-ModuleMember gebruiken (-function *naam van de functie*).

Naast de script modules zijn er ook nog binary modules. Dit valt buiten het bestek van deze cursus. Een binary module is een .DLL die in een .NET taal (zoals C#) wordt geschreven. Een binary module wordt gebruikt voor het schrijven van providers of wanneer men gebruik wil maken van eigenschappen die niet zo eenvoudig zijn in een Powershell script zoals multithreading.

Om extra informatie over de module mee te geven, kunnen we een PowerShell Module Manifest schrijven (extensie .psm1).

13. Security policy

Commandlets kunnen in de Powershell omgeving zonder problemen worden uitgevoerd. Wanneer we een script proberen uit te voeren, hangt dat af van de ExecutionPolicy:

```
PS C:\> Get-ExecutionPolicy
RemoteSigned
```

Er zijn verschillende mogelijke execution policies:

- Restricted: alleen interactieve commando's mogen worden uitgevoerd, scripts lukt niet en alleen configuratiebestanden die zijn ondertekend door een vertrouwde publisher mogen worden ingeladen
- AllSigned: alleen interactieve commando's en scripts en configuratiebestanden die zijn ondertekend door een vertrouwde publisher mogen worden uitgevoerd/ingeladen
- RemoteSigned: interactiave commando's en scripts en configuratiebestanden die lokaal zijn gecreëerd mogen worden uitgevoerd. Scripts die zijn gedownload moeten digitaal ondertekend zijn door een vertrouwde publisher. (default in Windows Server)
- Unrestricted: er is geen beperking op dei scripts die kunnen worden uitgevoerd, maar bij remote scripts is er wel een extra waarschuwing en een prompt om te vragen of men zeker is dat het script moet worden uitgevoerd.

- Bypass: er wordt niets geblokkeerd en er zijn ook geen waarschuwingen of prompts.
- Undefined; in de huidige omgeving(scope) is er geen Execution policy

Een execution policy hoort bij een scope. Er zijn 5 mogelijke scopes:

1. LocalMachine: de policy voor alle gebruikers op deze computer
2. CurrentUser: de policy voor de huidige gebruiker
3. Process: de policy voor het huidige Powershell proces
4. UserPolicy: de algemene policy voor alle users, ingesteld via een group policy
5. MachinePolicy: De algemene policy voor alle computers, ingesteld via een group policy

Voor elke scope kan er een aparte policy ingesteld worden. Des te lager een scope in bovenstaande lijst staat, des te meer voorrang die krijgt. Bij het instellen en het opvragen kunnen we scope meegeven:

```
PS C:\> Get-ExecutionPolicy
RemoteSigned
PS C:\> Get-ExecutionPolicy -Scope MachinePolicy
Undefined
PS C:\> Get-ExecutionPolicy -Scope UserPolicy
Undefined
PS C:\> Get-ExecutionPolicy -Scope LocalMachine
RemoteSigned
PS C:\> Get-ExecutionPolicy -Scope CurrentUser
Undefined
PS C:\> Get-ExecutionPolicy -Scope Process
Undefined
```

In dit geval zien we dat de policy bepaald wordt door de LocalMachine scope. Wanneer we de policy wijzigen voor de huidige gebruiker naar Bypass, wordt de uiteindelijke policy ook Bypass:

```
PS C:\> Set-ExecutionPolicy -Scope CurrentUser Bypass

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust.
Changing the execution policy might expose
you to the security risks described in the about_Execution_Policies help
topic at
http://go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the
execution policy?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "N"): Y
PS C:\> Get-ExecutionPolicy
Bypass
```

We hadden trouwens ook ineens alle scopes kunnen opvragen:

```
PS C:\Users\jef\Documents> Get-ExecutionPolicy -List

        Scope ExecutionPolicy
```

```
        ----- ---------------
MachinePolicy       Undefined
   UserPolicy       Undefined
      Process       Undefined
  CurrentUser          Bypass
 LocalMachine    RemoteSigned
```

Men kan een remote script omzetten naar een lokaal script via *unblock-file*.

Om de group policy in te stellen moeten we bij de Computer configuration/User configuration de volgende instelling doen: administrative templates/Windows Components/Winows Powershell/Turn on Script execution

Hoofdstuk 2. Providers

1. Inleiding

In het Windows operating system zijn er een aantal elementen aanwezig die data bevatten in een bepaalde structuur: het bestandssysteem, de registry, de omgevingsvariabelen. Providers laten in essentie toe om die data aan te spreken alsof ze in een filesysteem staan. We kunnen een lijst opvragen van de providers die aanwezig zijn:

```
PS C:\Users\Jef> Get-PSProvider

Name                 Capabilities                              Drives
----                 ------------                              ------
Registry             ShouldProcess, Transactions               {HKLM, HKCU}
Alias                ShouldProcess                             {Alias}
Environment          ShouldProcess                             {Env}
FileSystem           Filter, ShouldProcess, Credentials        {C, D}
Function             ShouldProcess                             {Function}
Variable             ShouldProcess                             {Variable}
Certificate          ShouldProcess                             {Cert}
WSMan                Credentials                               {WSMan}
```

De “Capabilities” hebben de volgende betekenis:

- ShouldProcess: provider ondersteunt -WhatIf en -Confirm
- Filter: items kunnen gefilterd worden zodat niet alle items worden overgehaald
- Credentials: men kan de -credentials parameter gebruiken om een andere identiteit te gebruiken om een connectie te maken met de provider
- Transactions: zorgt ervaar dat alle aanpassingen zijn afgewerkt voordat de actie als volledig afgewerkt wordt beschouwd. Wanneer dit niet lukt worden alle acties teruggezet naar hun oorspronkelijke toestand.

Zoals we kunnen zien, kent elk van die providers het concept van een “drive”. Voor een filesysteem is de betekenis waarschijnlijk wel duidelijk. We kunnen de inhoud van de root van een drive opvragen via de Get-ChildItem commandlet:

```
PS C:\Users\Jef> Get-ChildItem C:\


    Directory: C:\


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----       19/01/2016      7:33                $SysReset
d-----        9/12/2016     12:39                81d417111ef7d8d23aa5
d-----        6/04/2017      6:36                apache-tomcat-8.5.13
d-----       31/08/2013     14:16                Apps
…
```

Wanneer de inhoud van de HKEY_CURRENT_USER hive willen opvragen, kunnen we een gelijkaardig commando gebruiken:

```
PS C:\Users\Jef> Get-ChildItem HKCU:\


    Hive: HKEY_CURRENT_USER


Name                           Property
----                           --------
AppEvents
AppXBackupContentType
Console                        CtrlKeyShortcutsDisabled : 0
                               CursorSize               : 25
                               EnableColorSelection     : 0
                               ExtendedEditKeyCustom    : 0
                               FilterOnPaste            : 1
                               ForceV2                  : 1
                               FullScreen               : 0
```

Figuur 6. De inhoud van HKEY_CURRENT_USER

We kunnen HKEY_CURRENT_USER dus vergelijken met een disk drive.

De standaard provider is de FileSystem provider. Maar door de drive te veranderen, kunnen we de default provider wijzigen:

```
PS C:\Users\Jef> Set-Location HKCU:
PS HKCU:\>
```

We zitten nu in de registry. Het oude “cd” commando is in feite een alias voor Set-Location. We hiermee naar de Software key gaan in HKCU:

```
PS HKCU:\> cd Software
PS HKCU:\Software>
```

Op gelijkaardige manieren kunnen we ook naar de andere providers gaan. Ze hebben niet allemaal een hiërarchische structuur zoals het filesystem. Maar dat wil gewoon zeggen dat Alias, Environment, Function en Variable alleen een “root-directory” hebben.

2. Provider commandlets

2. A. Get-PSDrive

De Get-PSDrive commandlet geeft een overzicht van alle drives in de huidige sessie:

```
PS HKCU:\Software> Get-PSDrive

Name     Used (GB) Free (GB) Provider    Root                CurrentLocation
----     --------- --------- --------    ----                ---------------
Alias                        Alias
C        513,99    403,46    FileSystem  C:\                 Users\Jef
Cert                         Certificate \
D                            FileSystem  D:\
Env                          Environment
Function                     Function
HKCU                         Registry    HKEY_CURRENT_USER   Software
HKLM                         Registry    HKEY_LOCAL_MACHINE
Variable                     Variable
WSMan                        WSMan
```

We krijgen niet alleen de drives te zien, maar ook (voor de hiërarchische systemen) wat hun root is en de huidige locatie.

We kunnen ook alle properties opvragen voor een bepaalde provider:

```
PS HKCU:\Software> Get-PSDrive Env |fl *


Used            :
Free            :
CurrentLocation :
Name            : Env
Provider        : Microsoft.PowerShell.Core\Environment
Root            :
Description     : Drive that contains a view of the environment variables
for the process
MaximumSize     :
Credential      : System.Management.Automation.PSCredential
DisplayRoot     :
```

Het spreekt vanzelf dat de properties Used en Free alleen zin hebben voor de disks.

2. B. Item commandlets

De items commandlets werken met de items in de store. Voor een filesystem zijn dat de bestanden en de directories. Voor de registry zijn dat de keys (values zijn properties van keys in de registry)

- Clear-Item: verwijdert de “content” van een item, maar niet het item zelf
- Copy-Item: kopieert een item
- Get-Item: vraagt een item op
- Get-ChildItem: vraagt de onderliggende items op (in een hiërarchische provider)
- Invoke-Item: voert de default actie uit op een item. Meestal gebruikt voor bestanden om ze te openen of uit te voeren
- Move-Item: verplaatst een item
- Remove-Item: verwijdert een item
- Rename-Item: hernoemt een item
- Set-Item: wijzigt de waarde van een item (environment, variables, functions)

2. C. Item content commandlets

Deze commandlets werken met de content van de items. De items zijn hier meestal bestanden

- Add-Content: voegt content toe aan een bestand
- Clear-Content: verwijdert de inhoud van een bestand (zonder het bestand zelf te verwijderen)
- Get-Content: haalt de content van een bestand op
- Set-Content: wijzigt de inhoud van een bestand

2. D. Item property commandlets

Properties zijn de values van keys in de registry

- Clear-ItemProperty: verwijder de waarde van een property
- Copy-ItemProperty: kopieer een property en de waarde naar een andere locatie
- Get-ItemProperty: vraagt de properties van een item op
- Move-ItemProperty: verplaats een property en de waarde naar een andere locatie
- New-ItemProperty: (alleen gebruikt voor de registry) voegt een value en een waarde toe aan een key. Dit creëert geen nieuwe properties voor een object. (Add-Member)
- Remove-ItemProperty: verwijder een property en de waarde
- Rename-ItemProperty: hernoem een property
- Set-ItemProperty: Stel de properties van een item in.

2. E. Location commandlets

Location commandlets gebruiken we om te navigeren in hiërarchische systemen of om te wisselen van “drive”

- Get-Location: vraagt de huidige locatie op
- Pop-Location: wijzigt de locatie naar de laatst gepushte locatie

- Push-Location: pusht de huidige locatie op de stack
- Set-Location: wijzigt de location

2. F. Path commandlets

Path commandlets hebben alleen zin in hiërarchische providers

- Join-Path: voegt een parent en een child path samen
- Convert-Path: zet een standaard powershell path om naar een provider path
- Split-Path: geeft een onderdeel van een path terug
- Resolve-Path: set de wildcards om naar een path
- Test-Path: controleert of alle onderdelen van het path bestaan

Hoofdstuk 3. PowerShell en WMI

1. WMI

WMI is de afkorting van *Windows Management Instrumentation*. Het bestaat als sinds de vorige eeuw (Windows 95 en Windows NT) en het kan gebruikt worden om informatie te vergaren over een computer: printers, device drivers, user accounts, ODBC datasources, … Programma's kunnen ook eigen WMI klassen toevoegen.

Via de Win32_Process WMI klasse kunnen we informatie te pakken krijgen over de processen op een computer. In PowerShell kunnen we dit ook doen via Get-Process. Het voordeel van WMI is dat het uniforme interface is, waardoor we alle klassen op een gelijkaardige manier kunnen aanspreken. Eenmaal dat men die manier kent, wordt het gebruik van WMI eenvoudig.

De WMI klassen zitten in zogenaamde *namespaces*. Wanneer we een klasse kunnen vergelijken met een bestand, is de namespace de directory waarin het bestand staat. Binnen een namespace moeten alle klassennamen uniek zijn, net zoals alle bestandsnamen in een directory uniek moeten zijn. En net zoals de volledige naam van een bestand ook het path bevat, bevat een WMI klassenaam ook de namespace.

De standaard namespace in WMI is root\cimv2. Een klasse zoals WIN32_OperatingSystem heeft eigenlijk als naam root\cimv2\Win32_OperatingSystem.

Tegenwoordig (vanaf PowerShell 3.0) spreken we WMI aan via de CIM cmdlets. Die hebben als voordeel dat ze automatisch datums kunnen omzetten en op een flexibele manier omgaan met remote connecties.

De oudere WMI cmdlets kunnen datums niet automatisch omzetten en zijn beperkt tot DCOM over RPC voor remote connecties.

2. De WMI querytaal

De WMI Query Language (WQL) wordt gebruikt om de WMI klassen te ondervragen. Ze lijkt sterk op de querytaal die voor databanken wordt gebruikt (SQL). De standaard syntax is

```
SELECT <properties> FROM <WMI Class> [WHERE <voorwaarde>]
```

De WHERE clausule is optioneel. De properties kan een kommagescheiden lijst zijn of de wildcard '*'. Dat wil zeggen alle properties. Bijvoorbeeld:

```
PS C:\> Get-CimInstance -Query "SELECT * FROM Win32_Process WHERE
ProcessiD=$pid"

ProcessId Name            HandleCount WorkingSetSize VirtualSize
```

```
--------- ----           ----------- -------------- -----------
11900     powershell.exe 2388        226803712      2199799685120
```

De variabele $pid is hier ingevuld door PowerShell en is gelijk aan de ProcessID van het huidige proces.

In plaats van de WQL kunnen we ook gebruik maken van filters. Omdat de backslash het escape karakter is in WMI, moeten we die verdubbelen:

```
PS C:\> Get-CimInstance Win32_Process -Filter
"ExecutablePath='C:\\Windows\\Explorer.exe'"

ProcessId Name         HandleCount WorkingSetSize VirtualSize
--------- ----         ----------- -------------- -----------
3540      explorer.exe 1945        90591232       2199617425408
```

We kunnen in Strings ook wildcards gebruiken. Dit zijn dezelfde wildcards als in SQL:

- %: een willekeurig aantal willekeurige letters
- _: één willekeurige letter

```
PS C:\> Get-CimInstance Win32_Process -Filter 'ExecutablePath LIKE
"_:\\%.exe"'

ProcessId Name                    HandleCount WorkingSetSize VirtualSize
--------- ----                    ----------- -------------- -----------
8500      ETDCtrl.exe             493         16793600       172748800
6420      ETDGesture.exe          148         7364608        133083136
2388      sihost.exe              771         25305088       2199182831616
10956     svchost.exe             401         16461824       2199165644800
5844      svchost.exe             408         26394624       2199186964480
3624      taskhostw.exe           343         17448960       2199223336960
3540      explorer.exe            1913        89886720       2199614803968
4284      ShellExperienceHost.exe 1000        52563968       2199433412608
14308     SearchUI.exe            1340        79581184       2234320941056
8508      RuntimeBroker.exe       273         17375232       2199173402624
8552      RuntimeBroker.exe       499         16564224       2199195668480
8404      MSASCuiL.exe            154         8523776        2199150850048
…
```

Dit toont alle processen waarbij het ExecutablePath begint met een willekeurige letter, gevolgd door een dubbele punt en een backslash en eindigt op '.exe'.

Tabel 2. Operatoren voor -Filter en -Query

Naam	Operator	Voorbeeld
Gelijk aan	=	Name='powershell.exe'
Niet gelijk aan	<>	Name <> 'powershell.exe'

Naam	Operator	Voorbeeld
Groter dan	>	WorkingSetSite > $(100MB)
Groter of gelijk aan	>=	WorkingSetSite >= $(100MB)
Kleiner dan	<	WorkingSetSite < $(100MB)
Kleiner of gelijk aan	<=	WorkingSetSite <= $(100MB)
Test op Null	Is	CommandLine IS NULL, CommandLine IS NOT NULL
Wildcard operator	LIKE	CommandLine LIKE "*.exe"
Logische AND	AND	ProcessId=$pid AND Name='powershell.exe'
Logische OR	OR	ProcessId=$pid OR ProcessId=0
Logische NOT	NOT	NOT ProcessId = $pid

Wanneer we strings willen gebruiken in een -Filter of een -Query, moeten die tussen aanhalingstekens staan. Aangezien de Filter/Query string ook al tussen aanhalingstekens staan, zullen er geneste aanhalingstekens moeten worden gebruikt. In principe maakt het niet uit of we enkele aanhalingstekens nesten in dubbele aanhalingstekens of omgekeerd, tenzij we PowerShell variabelen willen gebruiken. Dan moeten er dubbele aanhalingstekens gebruikt worden als buitenste quotes.

3. Associaties tussen WMI klassen

In WMI zijn klassen met elkaar verbonden via zogenaamde associaties. Een proces is bijvoorbeeld verbonden met een logon sessie. De logon sessie bevat informatie over wanneer de gebruiker van het proces zich heeft aangemeld. Maar de logon sessie bevat geen informatie over de naam van de user. Die informatie zit in de user account klasse. Met behulp van Get-CimAssociattedInstance kunnen we de logon sessie te pakken krijgen die verbonden is met het PorwerShell process. Vervolgens kunnen we de user account opvragen die hoort bij die logon sessie. We beginnen met het powershell process op te vragen. Vervolgens gebruiken we dat proces om de geassocieerde Win32_LogonSession op te vragen. Tenslotte vragen van die logon sessie de geassocieerde Win32_UserAccount op.

```
PS C:\> $pwshell = Get-CimInstance -Query "SELECT * FROM Win32_PRocess
WHERE ProcessId=$PID"
PS C:\> $logon= Get-CimAssociatedInstance -InputObject $pwshell -
ResultClassName Win32_LogonSession
PS C:\> Get-CimAssociatedInstance -InputObject $logon -ResultClassName
Win32_UserAccount
```

Dankzij piping kunnen we dit eenvoudiger schrijven. We vervangen de query ook door een filter:

```
PS C:\> Get-CimInstance Win32_Process -Filter "ProcessId=$PID" | Get-
CimAssociatedInstance -ResultClassName Win32_LogonSession | Get-
CimAssociatedInstance -ResultClassName Win32_UserAccount
```

In dit voorbeeld wisten we (op één of andere manier) dat Win32_Process geassocieerd was met Win32_LogonSession en dat Win32_LogonSession geassocieerd was met Win32_UserAccount. Die informatie kunnen we (met enige moeite) ook opvragen. We moeten daarvoor de associatieklasse kennen tussen, bijvoorbeeld Win32_Process en Win32_LogonSession.

Aangezien de namen van associatieklassen de naam bevatten van de klassen waarmee ze geassocieerd zijn, kunnen we op zoek gaan naar de associatieklassen die het woord “Process” bevatten:

```
PS C:\> Get-CimClass Win32*Process* -QualifierName Association


   NameSpace: ROOT/cimv2

CimClassName                        CimClassMethods      CimClassProperties
------------                        ---------------      ------------------
Win32_NamedJobObjectProcess         {}                   {Collection,
Member}
Win32_ComputerSystemProcessor       {}                   {GroupComponent,
PartComponent}
Win32_SystemProcesses               {}                   {GroupComponent,
PartComponent}
Win32_SessionProcess                {}                   {Antecedent,
Dependent}
Win32_AssociatedProcessorMemory     {}                   {Antecedent,
Dependent, BusSpeed}
```

In dit geval zouden we al kunnen raden dat Win32_SessionProcess de klasse is die we zoeken. Maar om zeker te zijn, zouden we eens kunnen kijken of dit inderdaad de associatieklasse is tussen Win32_Process en Win32_LogonSession. Die informatie zit in CimClassProperties. Omdat dit een collection is, moeten we die uitvouwen. Om het overzichterlijker te houden, geven we meteen de juiste klassenaam mee:

```
PS C:\> Get-CimClass Win32_SessionProcess -QualifierName Association|Select
-ExpandProperty cimclassproperties


Name               : Antecedent
Value              :
CimType            : Reference
Flags              : Property, Key, ReadOnly, NullValue
Qualifiers         : {read, key, Override}
ReferenceClassName : Win32_LogonSession

Name               : Dependent
Value              :
CimType            : Reference
Flags              : Property, Key, ReadOnly, NullValue
Qualifiers         : {read, key, Override}
```

```
ReferenceClassName : Win32_Process
```

Door CimClassProperties uit te vouwen (-ExpandProperty) kunnen we de details van Antecedent en Dependent zien. We zien dat Win32_Process de Dependent is en we kunnen die informatie gebruiken om de Win32_SessionProcess klasse te selecteren:

```
PS C:\> Get-CimClass Win32_SessionProcess -QualifierName Association|Select
-ExpandProperty cimclassproperties|Where {$_.Name -eq 'Dependent' -and
$_.ReferenceClassName -eq 'Win32_Process'}


Name               : Dependent
Value              :
CimType            : Reference
Flags              : Property, Key, ReadOnly, NullValue
Qualifiers         : {read, key, Override}
ReferenceClassName : Win32_Process
```

Dit resultaat heeft een niet te onderschatten probleem. We houden namelijk alleen de CimClassProperty over en we zijn de oorspronkelijke klasse kwijt. Met behulp van de -Property parameter kunnen we er echter voor zorgen dat de naam mee wordt opgenomen in het resultaat:

```
PS C:\> Get-CimClass Win32_SessionProcess -QualifierName Association|Select
-ExpandProperty cimclassproperties -Property CimClassName|Where {$_.Name -
eq 'Dependent' -and $_.ReferenceClassName -eq 'Win32_Proc
ess'}


CimClassName       : Win32_SessionProcess
Name               : Dependent
Value              :
CimType            : Reference
Flags              : Property, Key, ReadOnly, NullValue
Qualifiers         : {read, key, Override}
ReferenceClassName : Win32_Process
```

Het resultaat bevat nu niet alleen de eigenschappen van de CimClassProperty maar ook de oorspronkelijk CimClassName. Dit object bewaren we in een tussentijds resultaat:

```
PS C:\> $resultaat = Get-CimClass Win32_SessionProcess -QualifierName
Association|Select -ExpandProperty cimclassproperties -Property
CimClassName|Where {$_.Name -eq 'Dependent' -and $_.ReferenceClassName -e
q 'Win32_Process'}
```

Nu kunnen we met $resultaat verder werken en iets gelijkaardig doen voor de Antecedent CimClassProperty:

```
PS C:\> Get-CimClass $resultaat.CimClassName|Select -ExpandProperty
cimclassproperties|Where {$_.Name -eq 'Antecedent'}|Select
ReferenceClassName

ReferenceClassName
```

```
------------------
Win32_LogonSession
```

Dit levert ons inderdaad de klasse op die geassocieerd is met Win32_Process.

4. Methodes aanroepen

We kunnen WMI niet alleen gebruiken om met de properties te werken. Er kunnen ook methodes gedefinieerd zijn op een WMI-object. Hier moeten we echter wel een onderscheid maken tussen de WMI-methodes en de PowerShell methodes. Wanneer we de PowerShell CmdLet Get-member gebruiken, krijgen we niet de methodes te zien van de CIM-class:

```
PS C:\> (Get-CimClass Win32_Process)|Get-Member


   TypeName: Microsoft.Management.Infrastructure.CimClass

Name                MemberType     Definition
----                ----------     ----------
Dispose             Method         void Dispose(), void
IDisposable.Dispose()
Equals              Method         bool Equals(System.Object obj)
GetHashCode         Method         int GetHashCode()
GetType             Method         type GetType()
ToString            Method         string ToString()
CimClassMethods     Property
Microsoft.Management.Infrastructure.Generic.CimReadOnlyKeyedCollection[Micr
osoft....
CimClassProperties  Property
Microsoft.Management.Infrastructure.Generic.CimReadOnlyKeyedCollection[Micr
osoft....
CimClassQualifiers  Property
Microsoft.Management.Infrastructure.Generic.CimReadOnlyKeyedCollection[Micr
osoft....
CimSuperClass       Property       cimclass CimSuperClass {get;}
CimSuperClassName   Property       string CimSuperClassName {get;}
CimSystemProperties Property
Microsoft.Management.Infrastructure.CimSystemProperties CimSystemProperties
{get;}
CimClassName        ScriptProperty System.String CimClassName
{get=[OutputType([string])]...
```

Hiervoor moeten we gebruik maken van de CimClassMethods:

```
PS C:\> (Get-CimClass Win32_Process).CimClassMethods

Name                    ReturnType Parameters
Qualifiers
----                    ---------- ----------
----------
Create                      UInt32 {CommandLine, CurrentDirectory,
ProcessStartupInformation, ProcessId} {Constructo...
Terminate                   UInt32 {Reason}
{Destructor...
GetOwner                    UInt32 {Domain, User}
{Implemente...
```

```
GetOwnerSid                 UInt32 {Sid}
{Implemente...
SetPriority                 UInt32 {Priority}
{Implemente...
AttachDebugger              UInt32 {}
{Implemente...
GetAvailableVirtualSize     UInt32 {AvailableVirtualSize}
{Implemente...
```

Om de methode op te roepen, moeten we Invoke-CimMethod gebruiken:

```
PS C:\> Get-CimInstance -ClassName Win32_Process -Filter "ProcessId =
$PID"| Invoke-CimMethod -MethodName Terminate
```

Het PowerShell venster wordt nu gesloten.

Hoofdstuk 4. PowerShell scripting

1. Inleiding

Er zijn twee verschillende manieren om PowerShell te gebruiken:

- Aan de command prompt(interactief)
- Als script(batch)

In het eerste geval typen we commando's aan een prompt. Elke keer wanneer we op return drukken, wordt het commando doorgestuurd en uitgevoerd. Wanneer het commando klaar is, kunnen we het volgende commando ingeven. We zijn interactief aan het werken.

Wanneer we PowerShell als een scripting taal willen gebruiken, moeten we een tekstbestand maken (met als extensie .PS1). Alle commandlets die we interactief kunnen gebruiken, kunnen we ook in een script gebruiken. De bedoeling van een script is dat we meerdere commandlets achter elkaar kunnen uitvoeren.

Daar stopt het echter niet. Soms willen we dezelfde reeks van commandlets meerdere malen achter elkaar uitvoeren. We hebben bijvoorbeeld een reeks computers waar we dezelfde configuratie willen doen. In plaats van dezelfde commandlet (of reeks commandlets) via copy-en-paste meerdere malen in het script bestand te zetten, gebruiken we een loop-constructie of iteratie.

Soms willen we bepaalde commandlets alleen uitvoeren wanneer aan een voorwaarde voldaan is. We willen een computer alleen configureren wanneer hij toegankelijk is, bijvoorbeeld. Dan zullen we een selectie-constructie gebruiken.

We kunnen deze constructies ook gebruiken aan de command prompt. Maar daar hebben ze minder zin omdat we maar één commandlet per keer uitvoeren.

Een script kan ook parameters hebben, net zoals commandlets. Wanneer we scripts willen creëren, zullen we ook moeten leren om die parameters uit te lezen.

2. Typische script constructies

2. A. Parameters

Net zoals commandlets kunnen scripts ook gebruik maken van parameters. Met behulp van parameters kunnen we een script flexibeler maken. We voeren immers niet steeds dezelfde reeks commando's uit. Met behulp van een parameter kunnen we het script op verschillende manieren uitvoeren.

De bedoeling van het script is dat de eerste 5 regels van een bestand worden getoond. We zullen beginnen met een versie die steeds hetzelfde bestand toont, namelijk het script zelf:

```
<#
   .SYNOPSIS
   Een eerste voorbeeld van een script
   .DESCRIPTION
   Toont de eerste 5 regels van het script
#>
Get-Content LeesFile1.ps1 -TotalCount 5
```

Bewaar dit bestand als “LeesFile1.ps1”

We beginnen met een block commentaar die gebruikt zal worden door Get-Help:

```
PS C:\Users\Jef\Google Drive\PowerShell> Get-Help .\LeesFile1.ps1

NAME
    C:\Users\Jef\Google Drive\PowerShell\LeesFile1.ps1

SYNOPSIS
    Een eerste voorbeeld van een script


SYNTAX
    C:\Users\Jef\Google Drive\PowerShell\LeesFile1.ps1 [<CommonParameters>]


DESCRIPTION
    Toont de eerste 5 regels van het script


RELATED LINKS

REMARKS
    To see the examples, type: "get-help C:\Users\Jef\Google
Drive\PowerShell\LeesFile1.ps1 -examples".
    For more information, type: "get-help C:\Users\Jef\Google
Drive\PowerShell\LeesFile1.ps1 -detailed".
    For technical information, type: "get-help C:\Users\Jef\Google
Drive\PowerShell\LeesFile1.ps1 -full".
```

Wanneer we het script uitvoeren, krijgen we de eerste vijf regels van het script te zien:

```
PS C:\Users\Jef\Google Drive\PowerShell> .\LeesFile1.ps1
<#
   .SYNOPSIS
   Een eeerste voorbeeld van een script
   .DESCRIPTION
   Toont de eerste 5 regels van het script
```

Om een parameter te definiëren voor het script, maken we gebruik van een Param-block(leesscript2.ps1):

```
<#
   .SYNOPSIS
   Een tweede voorbeeld van een script
   .DESCRIPTION
   Toont de eerste 5 regels van een bestand
#>
Param (
  [string]$bestandsnaam
)
Get-Content $bestandsnaam -TotalCount 5
```

De naam van de parameter is hier "$bestandsnaam". Een parameternaam moet altijd beginnen met een $-teken. Wanneer we het script willen uitvoeren, kunnen we de naam van het bestand meegeven:

```
PS C:\Users\Jef\Google Drive\PowerShell> .\LeesFile2.ps1 .\LeesFile1.ps1
<#
   .SYNOPSIS
   Een eerste voorbeeld van een script
   .DESCRIPTION
   Toont de eerste 5 regels van het script
```

We kunnen ook gebruik maken van de naam door een named parameter mee te geven:

```
.\LeesFile2.ps1 -bestandsnaam .\LeesFile1.ps1
```

Wanneer we geen naam meegeven, loopt het echter mis:

```
PS C:\Users\Jef\Google Drive\PowerShell> .\LeesFile2.ps1
Get-Content : Cannot bind argument to parameter 'Path' because it is an
empty string.
At C:\Users\Jef\Google Drive\PowerShell\LeesFile2.ps1:10 char:13
+ Get-Content $bestandsnaam -TotalCount 5
+             ~~~~~~~~~~~~~
    + CategoryInfo          : InvalidData: (:) [Get-Content],
ParameterBindingValidationException
    + FullyQualifiedErrorId :
ParameterArgumentValidationErrorEmptyStringNotAllowed,Microsoft.PowerShell.
Commands.GetContentCommand
```

Aangezien $bestandsnaam geen waarde heeft gekregen, krijgen we een foutmelding. De oplossing is dat we de parameter *mandatory* maken:

```
<#
   .SYNOPSIS
   Een tweede voorbeeld van een script
   .DESCRIPTION
   Toont de eerste 5 regels van een bestand
#>
Param (
  [parameter(Mandatory=$true)]
  [string]$bestandsnaam
)
Get-Content $bestandsnaam -TotalCount 5
```

2. B. Variabelen

Variabelen in PowerShell worden gebruikt om een waarde bij te houden. De parameter $bestandsnaam is een voorbeeld van een variabele. Zoals we ondertussen al weten, moet de naam beginnen met een $-teken. Een variabele kan op twee verschillende manieren een waarde krijgen:

- Als parameter (zie vorig voorbeeld)
- Via een toewijzing

Een parameter-variabele krijgt zijn waarde wanneer het script wordt opgeroepen. Maar we kunnen in het script zelf ook een waarde toekennen aan een variabele via de toewijzingsoperator '='.

Stel dat we in het vorige voorbeeld zouden willen dat de bestandsnaam zonder de extensie '.ps1' wordt meegegeven. Dan moeten we de extensie zelf toevoegen voordat we Get-Content kunnen uitvoeren:

```
<#
   .SYNOPSIS
   Een derde voorbeeld van een script
   .DESCRIPTION
   Toont de eerste 5 regels van een bestand
#>
Param (
  [parameter(Mandatory=$true)]
  [string]$bestandsnaam
)
$bestandsnaam = $bestandsnaam + ".ps1"
Get-Content $bestandsnaam -TotalCount 5
```

We geven $bestandsnaam hier een nieuwe waarde. De nieuwe waarde is gelijk aan de vorige waarde, gevolgd door ".ps1".

Dat werkt, maar alleen wanneer de gebruiker zelf nog geen extensie heeft toegevoegd. Eigenlijk mag $bestandsnaam = $bestandsnaam + ".ps1" alleen worden toegevoegd wanneer er nog geen extensie is gedefinieerd. Hiervoor kunnen we de if/then constructie gebruiken.

2. C. If/then constructie

Een if moet gevolgd worden door een voorwaarde. Een voorwaarde is iets wat waar of niet waar is ($true of $false). In dit voorbeeld willen we controleren of $bestandsnaam een "." bevat. Hiervoor kunnen we de Contains()-functie gebruiken van een string. De voorwaarde wordt nu:

"Als $bestandsnaam geen punt bevat, voeg de extensie ".ps1" toe"

Omdat we de voorwaarde van de Contains()-functie willen omkeren, gebruiken we de -not operator:

```
<#
    .SYNOPSIS
    Een vierde voorbeeld van een script
    .DESCRIPTION
    Toont de eerste 5 regels van een bestand
#>
Param (
  [parameter(Mandatory=$true)]
  [string]$bestandsnaam
)
if ( -not $bestandsnaam.Contains(".")) {
    $bestandsnaam = $bestandsnaam + ".ps1"
}
Get-Content $bestandsnaam -TotalCount 5
```

Het statement '$bestandsnaam = $bestandsnaam + ".ps1"' wordt nu alleen uitgevoerd wanneer er geen '.' staat in $bestandsnaam.

We kunnen via een if-constructie ook definiëren wat er moet gebeuren wanneer niet aan de voorwaarde voldaan is via else:

```
If (voorwaarde) {
    Statements-als-voorwaarde-waar
}Else{
    Statements-als-voorwaarde-niet-waar
}
```

We kunnen ook meerdere voorwaarden checken via ElseIf:

```
If (voorwaarde1) {
  Statements-als-voorwaarde1-waar
}ElseIf(voorwaarde2) {
  Statements-als-voorwaarde2-waar
}ElseIf(voorwaarde3) {
  Statements-als-voorwaarde3-waar
}Else{
  Statements-als-geen-enkele-voorwaarde-waar
}
```

2. D. ForEach

Stel dat we een reeks extensies willen uitproberen: .ps1, .txt en .bat. We willen telkens controleren of het bestand bestaat en wanneer dat het geval is, willen we de inhoud tonen. We zouden dat als volgt kunnen schrijven (maar dit is niet aan te raden):

```
<#
    .SYNOPSIS
    Een vijfde voorbeeld van een script
    .DESCRIPTION
    Toont de eerste 5 regels van een bestand
#>
Param (
  [parameter(Mandatory=$true)]
  [string]$bestandsnaam
)
```

```
if (Test-Path ($bestandsnaam + ".ps1")){
   Get-Content ($bestandsnaam + ".ps1") -TotalCount 5
}
if (Test-Path ($bestandsnaam + ".bat")){
   Get-Content ($bestandsnaam + ".bat") -TotalCount 5
}
if (Test-Path ($bestandsnaam + ".txt")){
   Get-Content ($bestandsnaam + ".txt") -TotalCount 5
}
```

Het enige waarin de drie reeksen statements verschillen, is in de extensie. We kunnen dit beter schrijven met een array en een ForEach constructie:

```
<#
   .SYNOPSIS
   Een vijfde voorbeeld van een script
   .DESCRIPTION
   Toont de eerste 5 regels van een bestand
#>
Param (
  [parameter(Mandatory=$true)]
  [string]$bestandsnaam
)
$extensies = ".txt",".ps1",".bat"
foreach($extensie in $extensies) {
   if (Test-Path ($bestandsnaam + $extensie)){
     Get-Content ($bestandsnaam + $extensie) -TotalCount 5
   }
}
```

In het Nederlands zou dit worden:

Voor elke string in de array $extensies, bewaar de string in de $extensie en … . We voeren de statements in de foreach-constructie dus uit voor elk element in de array.

3. Een script als module

PowerShell is modulair opgebouwd. Via modules kunnen we extra functionaliteit toevoegen aan de PowerShell omgeving. Wanneer we het vorige script willen gebruiken als module, moeten we rekening houden met het feit dat we een functie zullen moeten maken. Een module bevat immers één of meerdere functies. Een eerste aanpassing zal er dus als volgt uitzien:

```
function Read-Bestand {
   <#
      .SYNOPSIS
      Een vijfde voorbeeld van een script
      .DESCRIPTION
      Toont de eerste 5 regels van een bestand
   #>
   Param (
     [parameter(Mandatory=$true)]
     [string]$bestandsnaam
   )
   $extensies = ".txt",".ps1",".bat"
```

```
   foreach($extensie in $extensies) {
      if (Test-Path ($bestandsnaam + $extensie)){
        Get-Content ($bestandsnaam + $extensie) -TotalCount 5
      }
   }
}
```

Verder moeten we zorgen voor de juiste extensie: .psm1

Vervolgens is het een goed idee om te bepalen welke onderdelen van de module mogen gebruikt worden. Het zou kunnen dat we bepaalde hulpfuncties hebben gedefinieerd die alleen maar bestemd zijn om gebruikt te worden door functies in de module. Hiervoor gebruiken we export-modulemember:

```
Export-ModuleMember -function Read-Bestand
```

Tenslotte moeten we de module bewaren op de juiste plaats. PowerShell zoekt naar modules in $env:PSModulePath. Een van de directories in dit path is documents/WindowsPowerShell/modules in de home directory van de gebruiker.

Een module moet bewaard worden in een subdirectory met dezelfde naam als de module. In ons geval zou dat LeesFile kunnen zijn. Eenmaal dat we dat gedaan hebben, kunnen we de module importeren:

```
Import-module leesfile
```

Vanaf PowerShell 3.0 gebeurt de import in feite automatisch vanaf het moment dat we een cmdlet in de module gebruiken. Maar wanneer we een aanpassing doen aan het bestand, moeten we de module opnieuw importeren via

```
Import-module leesfile -Force
```

4. Nog enkele typische script constructies

4. A. Set-StrictMode

Een van de vervelendste fouten in een script zijn typfouten. In het volgende script staat een kleine fout:

```
param(
      [Parameter(Mandatory=$True)]
      [string[]] $namen
)
foreach ($naam in $namen) {
      Write-Host $naa
}
```

De output van dit script is leeg omdat $naa geen waarde heeft gekregen. Om ervoor te zorgen dat we niet nodeloos beginnen zoeken naar fouten in de logica, kunnen we “Strict Mode” activeren:

```
param(
      [Parameter(Mandatory=$True)]
      [string[]] $namen
)
Set-StrictMode -Version latest
foreach ($naam in $namen) {
      Write-Host $naa
}
```

De Set-StrictMode cmdlet controleert het script op bepaalde fouten. We kunnen verschillende controles definiëren op basis van de PowerShell versie (-Version). Des te hoger het versienummer, des te stricter de controles zijn. Om ook met nieuwere versies van PowerShell rekening te houden, kunnen we in plaats van een versienummer ook “latest” invullen. Het resultaat van het instellen van strictmode in het vorige script heeft het volgende gevolg:

```
PS C:\Users\jef> .\myscript.ps1 -namen eerste,tweede
The variable '$naa' cannot be retrieved because it has not been set.
At C:\Users\jef\myscript.ps1:8 char:13
+      Write-Host $naa
+                 ~~~~
    + CategoryInfo          : InvalidOperation: (naa:String) [],
RuntimeException
    + FullyQualifiedErrorId : VariableIsUndefined

The variable '$naa' cannot be retrieved because it has not been set.
At C:\Users\jef\myscript.ps1:8 char:13
+      Write-Host $naa
+                 ~~~~
    + CategoryInfo          : InvalidOperation: (naa:String) [],
RuntimeException
    + FullyQualifiedErrorId : VariableIsUndefined
```

4. B. Verbose en Debug

Wanneer we de uitvoering van een script willen volgen, kunnen we Write-Host gebruiken om tussentijdse waarden van variabelen te tonen. Het probleem met deze cmdlet is dat de tekst altijd wordt getoond. Het zou misschien handig zijn wanneer we sommige tekst alleen schrijven wanneer we meer uitgebreide output van een script willen tonen. In de bestaande PowerShell cmdlets bestaat dikwijls de mogelijkheid om via de parameters -Verbose of -Debug extra output te tonen. We zouden deze switch-parameters zelf aan ons script kunnen toevoegen. Maar in feite is dit niet nodig:

```
param(
       [Parameter(Mandatory=$True)]
      [string[]] $namen
)
foreach ($naam in $namen) {
      Write-Host $naam
      Write-Debug "Debug info: $naam"
      Write-Verbose "Verbose info: $naam"
}
```

Wanneer we dit script gewoon uitvoeren, krijgen we de verschillende namen te zien:

```
PS C:\Users\jef> .\myscript.ps1 -namen eerste,tweede
eerste
tweede
```

Alleen de output van Write-Host wordt getoond. Dit is de output die we willen krijgen wanneer we het script gewoon uitvoeren. De extra parameter -Verbose, zorgt ervoor dat ook de output van Write-Verbose wordt getoond:

```
PS C:\Users\jef> .\myscript.ps1 -namen eerste,tweede -Verbose
eerste
VERBOSE: Verbose info: eerste
tweede
VERBOSE: Verbose info: tweede
```

De -Debug parameter toont de Write-Debug informatie. Maar het zorgt er ook voor dat er een extra prompt verschijnt:

```
PS C:\Users\jef> .\myscript.ps1 -namen eerste,tweede -Debug
eerste
DEBUG: Debug info: eerste

Confirm
Continue with this operation?
[Y] Yes  [A] Yes to All  [H] Halt Command  [S] Suspend  [?] Help (default
is "Y"): S
PS C:\Users\jef>> $namen
eerste
tweede
PS C:\Users\jef>> exit

Confirm
Continue with this operation?
[Y] Yes  [A] Yes to All  [H] Halt Command  [S] Suspend  [?] Help (default
is "Y"): Y
tweede
DEBUG: Debug info: tweede

Confirm
Continue with this operation?
[Y] Yes  [A] Yes to All  [H] Halt Command  [S] Suspend  [?] Help (default
is "Y"): S
PS C:\Users\jef>> $naam
tweede
PS C:\Users\jef>> exit

Confirm
Continue with this operation?
[Y] Yes  [A] Yes to All  [H] Halt Command  [S] Suspend  [?] Help (default
is "Y"): A
```

Bij elk Write-Debug commando wordt er gevraagd of we verder willen gaan(Yes), vanaf nu altijd verder willen gaan (All), het script willen stoppen(Halt) of het script tijdelijk willen onderbreken (Suspend). In dat laatste geval krijgen we een *continuation prompt* te zien:

```
 PS C:\Users\jef>>
```

(let op de dubbele >>). We zitten nog steeds in het script, dus we kunnen de waarden van de variabelen in het script opvragen. Wanneer we alle nodige informatie hebben opgevraagd, kunnen we de continuation prompt verlaten met “exit”:

```
PS C:\Users\jef>> exit

Confirm
Continue with this operation?
[Y] Yes  [A] Yes to All  [H] Halt Command  [S] Suspend  [?] Help (default
is "Y"): Y
```

4. C. Collections

We hebben al kennisgemaakt met CmdLets die meerdere items teruggeven. Via ForEach-Object kunnen we de afzonderlijke elementen in die collectie van items overlopen. In een script kunnen we ook het *foreach* commando gebruiken. Om zelf een collectie te maken en toe te wijzen kunnen we de *array*-syntax gebruiken:

```
$namen = @(“een”, “twee”, “drie”)
```

PowerShell laat ook toe dat we de array-syntax weglaten:

```
$namen = “een”, “twee”, “drie”
```

De array-synctax is nodig wanneer we een lege array willen declareren waaraan we later elementen toevoegen:

```
$namen = @()
$namen += “een”
$namen += “twee”
$namen += “drie”
```

Een element weghalen moet gebeuren door een nieuwe array te definiëren:

```
$namen = $namen | where {$_ -ne “twee“}
```

Een probleem met gewone arrays is dat ze eigenlijk alleen maar geschikt zijn om lijsten van identieke objecten bij te houden. We kunnen de afzonderlijke items opvragen via hun index ($namen[0]). Maar een index heeft niet veel betekenis. Stel dat we de volgende array zouden gebruiken om de voornaam en achternaam van een persoon bij te houden:

```
$persoon = @(“Karen”, “Damen”)
```

We kunnen de voornaam opvragen via $persoon[0]. Maar het is niet echt duidelijk dat dit de voornaam is. Een hashtable is meer geschikt voor deze situatie. Een hashtable bestaat uit sleutel-waarde paren:

```
PS C:\Users\jef> $persoon = @{voornaam="Karen"; achternaam="Damen"}
PS C:\Users\jef> $ersoon.voornaam
```

```
PS C:\Users\jef> $persoon.voornaam
Karen
```

Een hashtable is een eigen type in .NET met eigen eigenschappen. We kunnen bijvoorbeeld de lijst van alle sleutels opvragen:

```
PS C:\Users\jef> $persoon.keys
achternaam
voornaam
```

Zonder te weten welke keys aanwezig zijn in een object, kunnen we toch alle waarden opvragen door de []-syntax te gebruiken:

```
PS C:\Users\jef> $persoon.keys|foreach { $_ +":"+$persoon[$_]}
achternaam:Damen
voornaam:Karen
```

De volgorde van de keys wordt niet behouden, tenzij we een ordered hashtable gebruiken:

```
PS C:\Users\jef> $persoon = [ordered]@{voornaam="Keren";
achternaam="Damen"}
PS C:\Users\jef> $persoon.keys|foreach { $_ +":"+$persoon[$_]}
voornaam:Keren
achternaam:Damen
```

5. PowerShell jobs

5. A. Background jobs

We kunnen commando's en scripts interactief uitvoeren. Dat wil zeggen dat de command prompt geblokkeerd is tot het commando of het script afgelopen is. We kunnen ze echter ook als een *job* uitvoeren. Dat is te vergelijken met de background jobs van Unix (&). Hiervoor hebben we twee commandlets: Start-Job en Invoke-Command. Het verschil is dat Start-Job het Remoting subsysteem niet gebruikt. De commando's worden lokaal uitgevoerd. Via Invoke-Command kan men commando's sturen naar andere computers.

Stel dat we een script evtlog.ps1 hebben. We kunnen dit script als job uitvoeren via:

```
PS D:\Users\Jef> Start-Job -FilePath .\evtlog.ps1

Id Name PSJobTypeName State   HasMoreData Location  Command
-- ---- ------------- -----   ----------- --------  ------
3  Job3 BackgroundJob Running True        localhost Get-Eventlog -LogName
```

Aan de hand van de JobId kunnen we informatie opvragen over de job:

```
PS D:\Users\Jef> Get-Job 3 | fl


HasMoreData   : True
StatusMessage :
```

```
Location      : localhost
Command       : Get-Eventlog -LogName application
JobStateInfo  : Running
Finished      : System.Threading.ManualResetEvent
InstanceId    : 95c11a09-d66a-43ef-9c99-a71730976371
Id            : 3
Name          : Job3
ChildJobs     : {Job4}
PSBeginTime   : 24/03/2019 8:40:47
PSEndTime     :
PSJobTypeName : BackgroundJob
Output        : {}
Error         : {}
Progress      : {}
Verbose       : {}
Debug         : {}
Warning       : {}
Information   : {}
State         : Running
```

Wanneer de job is uitgevoerd op verschillende computers, kunnen we informatie opvragen over de *child jobs* via -IncludeChildJob

De resultaten van een job worden gecached en kunnen worden opgevraagd met Receive-Job. Wanneer de output eenmaal is opgevraagd, verdwijnt hij uit de cache. De *HasMoreData*-property geeft aan of resultaten kunnen worden opgevraagd.

```
PS D:\Users\Jef> Receive-Job 3
```

Na het uitvoeren van dit commando zijn er geen gecachte data meer:

```
PS D:\Users\Jef> Get-Job 3 | fl


HasMoreData   : False
StatusMessage :
Location      : localhost
Command       : Get-Eventlog -LogName application
…
```

Wanneer we de gecachte gegevens toch zouden willen bewaren, kunnen we de -Keep switch meegeven aan de Receive-Job commandlet.

Informatie over een job die mislukt is, vinden we in de ChildJob-property JobStateInfo. We moeten die expanden en daar vervolgens de Reason van opvragen:

```
PS D:\Users\Jef> Get-Job 2| select -ExpandProperty JobstateInfo|select -
ExpandProperty Reason
Aangevraagde registertoegang is niet toegestaan.
```

Voor sommige commandlets is er een aparte -AsJob parameter:

```
PS D:\Users\Jef> Get-WmiObject -Class Win32_Service -ComputerName win8
-AsJob
```

De commandlet Invoke-Command heeft ook een -AsJob parameter. Hiermee kunnen we elk script als een job uitvoeren. Als voorbeeld voeren we een script uit dat de gestopte service met starttype automatic toont op een reeks computers:

```
Get-Service | where {$_.StartType -eq "Automatic" -and $_.Status -eq
"stopped"}
```

Om dit script als job uit te voeren, kunnen we het volgende commando gebruiken:

```
PS C:\Users\jef> invoke-command -ComputerName localhost,127.0.0.1 -FilePath
.\stoppedservices.ps1 -asjob

Id Name  PSJobTypeName State   HasMoreData Location            Command
-- ----  ------------- -----   ----------- --------            -------
3  Job3  RemoteJob     Running True        localhost,127.0.0.1 Get-Service


PS C:\Users\jef> get-job -IncludeChildJob

Id Name PSJobTypeName State     HasMoreData Location             Command
-- ---- ------------- -----     ----------- --------             -------
3  Job3 RemoteJob     Completed True        localhost,127.0.0.1 Get-
Service | where {$...
4  Job4               Completed True        localhost           Get-
Service | where {$...
5  Job5               Completed True        127.0.0.1           Get-
Service | where {$...
```

Vervolgens kunnen we het resultaat opvragen:

```
PS C:\Users\jef> Receive-Job 4

Status   Name           DisplayName              PSComputerName
------   ----           -----------              --------------
Stopped  MapsBroker     Downloaded Maps Manager  localhost
Stopped  RemoteRegistry Remote Registry          localhost
Stopped  sppsvc         Software Protection      localhost
```

5. B. Scheduled jobs.

We kunnen scripts ook laten uitvoeren op een bepaald moment. Hiervoor hebben we een jobtrigger nodig. Dat is het moment waarop de job moet starten. Dat kan één welbepaalde moment zijn:

```
$trigger = New-JobTrigger -Once -At "1/20/2012 3:00 AM"
```

Maar we kunnen ook meegeven dat een job moet herhaald worden om de vier weken op weekdagen om 23:00 uur:

```
$trigger = New-JobTrigger -Weekly -DaysOfWeek Monday, Wednesday, Friday -At
"23:00" -WeeksInterval 4
```

We kunnen ook opties definiëren. Met het volgende commando creëren we een options object dat ervoor zal zorgen dat de job niet wordt getoond in de job scheduler en dat ze niet zal starten wanneer er geen netwerk aanwezig is. Ze zal op het volgende geschedulde moment starten (op voorwaarde dat er dan wel een netwerk is):

```
$options = New-ScheduledJobOption –HideInTaskScheduler -RequireNetwork
```

Vervolgens kunnen we een job registreren bij de job scheduler:

```
Register-ScheduledJob –Name DailyRestart –ScriptBlock { Get-Process
;Restart-Computer -force } –Trigger $trigger –ScheduledJobOption $options
```

We kunnen een overzicht krijgen van de scheduled jobs:

```
PS C:\> Get-Job -Name DailyRestart
Id     Name            State         HasMoreData     Location
--     ----            -----         -----------     --------
45     DailyRestart    Completed     True            localhost
46     DailyRestart    Completed     True            localhost
47     DailyRestart    Completed     True            localhost
48     DailyRestart    Completed     True            localhost
49     DailyRestart    Completed     True            localhost
50     DailyRestart    Completed     True            localhost
51     DailyRestart    Completed     True            localhost
```

Hoofdstuk 5. Beheertaken op een windows client

1. Gebruikersbeheer

Het creëren van een nieuwe gebruiker is vanaf PowerShell 5.1 vereenvoudigd omdat er een aparte cmdlet voor is gemaakt: New-LocalUser. We kunnen in deze cmdlet de normale opties meegeven zoals AccountneverExpires, PassWordNeverExpires, UserMayNotChangePassword, … . De meeste van deze opties zijn vrij duidelijk, maar we moeten toch uitkijken met het paswoord. Wanneer we het New-LocalUser commando bekijken zien we dat de Password parameter van het type *SecureString* is. Dat wil zeggen dat we die op een speciale, veilige manier moeten behandelen. Wanneer we een gewone String proberen te gebruiken, krijgen we een foutmelding:

```
PS C:\> New-LocalUser -Name Joske -Password Joske
New-LocalUser : Cannot bind parameter 'Password'. Cannot convert the
"Joske" value of type "System.String" to type
"System.Security.SecureString".
At line:1 char:37
+ New-LocalUser -Name Joske -Password Joske
+                                     ~~~~~
    + CategoryInfo          : InvalidArgument: (:) [New-LocalUser],
ParameterBindingException
    + FullyQualifiedErrorId : CannotConvertArgumentNoMessage,
Microsoft.PowerShell.Commands.NewLocalUserCommand
```

De omschrijving van de fout is duidelijk, een String kan niet worden omgezet naar een SecureString. We hebben hier twee mogelijkheden:

- Ofwel vragen we het paswoord aan de gebruiker wanneer het commando wordt uitgevoerd
- Ofwel geven we het paswoord op een “veilige” manier mee in het commando

De volgende cmdlet vraagt het paswoord aan de gebruiker op een veilige manier:

```
PS C\> $paswoord = Read-Host -AsSecureString
****
```

De tekst is nu op een veilige manier bewaard in de variabele $password en we kunnen die nu gebruiken in New-LocalUser:

```
PS C:\> New-LocalUser -Name Joske -Password $paswoord
```

We moeten wel zorgen dat we dit commando uitvoeren in een *elevated command prompt*.

Een alternatief is dat we het paswoord meegeven op de commandline:

```
PS C:\> $paswoord = ConvertTo-SecureString "Rr00ttt" -AsPlainText -Force
PS C:\> New-LocalUser -Name Josje -Password $paswoord
```

De -Force parameter is verplicht als extra bescherming.

We kunnen dit ook in één regel schrijven:

```
PS C:\Users\jef> New-LocalUser Josje -Password (ConvertTo-SecureString
"Rr00ttt" -AsPlainText -Force)
Name  Enabled Description
----  ------- -----------
Josje True
```

Een nieuwe groep maken op een windows client is veel eenvoudiger:

```
PS C:\>New-LocalGroup -Name "K3"
```

De leden van een groep kunnen we ook opvragen:

```
PS C:\> Get-LocalGroupMember
Supply values for the following parameters:
Name: K3

ObjectClass Name                PrincipalSource
----------- ----                ---------------
User        pcbeta\karen        Local
User        pcbeta\kathleen     Local
```

Een gebruiker toevoegen als lid kan op de volgende manier:

```
PS C:\> Add-LocalGroupMember -Group K3 -Member "josje"
```

Deze cmdlets spreken voor zich. Andere cmdlets om lokale gebruikers te beheren zijn:

- Remove-LocalGroupMember
- Remove-LocalGroup
- Remove-LocalUser

Met Set-LocalUser kunnen we het paswoord wijzigen.

2. Schijfbeheer

In het hoofdstuk over de providers hebben we al gezien hoe we bestanden en directories kunnen creëren. We kunnen om te beginnen een directory creëren voor K3:

```
PS C:\>New-Item -Name K3 -Path "c:\documenten\" -ItemType Directory
```

We kunnen vervolgens een kijken welke beveiliging er op die nieuwe folder staat:

```
PS C:\> Get-Acl C:\documenten\K3 |fl


Path   : Microsoft.PowerShell.Core\FileSystem::C:\documenten\K3
Owner  : BUILTIN\Administrators
Group  : Server213\None
Access : NT AUTHORITY\SYSTEM Allow  FullControl
         BUILTIN\Administrators Allow  FullControl
         BUILTIN\Users Allow  ReadAndExecute, Synchronize
         BUILTIN\Users Allow  AppendData
         BUILTIN\Users Allow  CreateFiles
         CREATOR OWNER Allow  268435456
Audit  :
Sddl   : O:BAG:S-1-5-21-895315077-4074179417-748770018-
513D:AI(A;OICIID;FA;;;SY)(A;OICIID;FA;;;BA)(A;OICIID;0x1200a9;;;
         BU)(A;CIID;LC;;;BU)(A;CIID;DC;;;BU)(A;OICIIOID;GA;;;CO)
```

Omdat de beveiliging nogal uitgebreid is, gebruiken we Format-List om de waarden onder elkaar te tonen. SDDL staat voor Security Descriptor Definition Language.

Een SDDL bestaat uit vijf delen:

- De owner (O:) (BA staat voor Builtin\Administrators)
- De primary group (G:)
- De DACL of *descretionary access control list* (D:)
- De SACL of *system access control list* (S:)
- De header

In de praktijk hebben we die meestal niet nodig. Daarom gaan we er niet dieper op in.

Een alternatief voor Get-Acl is de GetAccessControl functie van files en folders:

```
PS C:\> ((Get-Item c:\documenten\k3).GetAccessControl('Access')).Access


FileSystemRights  : FullControl
AccessControlType : Allow
IdentityReference : NT AUTHORITY\SYSTEM
IsInherited       : True
InheritanceFlags  : ContainerInherit, ObjectInherit
PropagationFlags  : None

FileSystemRights  : FullControl
AccessControlType : Allow
IdentityReference : BUILTIN\Administrators
IsInherited       : True
InheritanceFlags  : ContainerInherit, ObjectInherit
PropagationFlags  : None

FileSystemRights  : ReadAndExecute, Synchronize
AccessControlType : Allow
IdentityReference : BUILTIN\Users
IsInherited       : True
InheritanceFlags  : ContainerInherit, ObjectInherit
```

```
PropagationFlags  : None

FileSystemRights  : AppendData
AccessControlType : Allow
IdentityReference : BUILTIN\Users
IsInherited       : True
InheritanceFlags  : ContainerInherit
PropagationFlags  : None

FileSystemRights  : CreateFiles
AccessControlType : Allow
IdentityReference : BUILTIN\Users
IsInherited       : True
InheritanceFlags  : ContainerInherit
PropagationFlags  : None

FileSystemRights  : 268435456
AccessControlType : Allow
IdentityReference : CREATOR OWNER
IsInherited       : True
InheritanceFlags  : ContainerInherit, ObjectInherit
PropagationFlags  : InheritOnly
```

Het argument van GetAccessControl ('Access') bepaalt welke informatie we over de security willen opvragen. Mogelijke waarden zijn:

- Access: de Discretionary Access Control List (DACL)
- All: alle informatie
- Audit: de System Access Control List (SACL)
- Group: De primary group
- None: geen
- Owner: de owner

De optie 'All' komt overeen met Get-Acl.

Wanneer we security willen aanpassen en de nieuwe security wijkt sterk af van de huidige, is het meestal een goed idee om de overerving af te zetten. Daarvoor maken we gebruik van de functie SetAccessRuleProtection(*isProtected,preserveInheritance*).

Het argument *isProtected* bepaalt of we willen dat het item overerft van de parent. True wil zeggen dat het object beschermd is tegen inheritance (geen inheritance). False wil zeggen dat er wel inheritance is.

Het tweede argument (*preserveInheritance*) heeft alleen effect wanneer *isProtected* true is. Het bepaalt wat er moet gebeuren met de overgeërfde regels. De waarde true behoudt de regels, de waarde false verwijdert ze. Wanneer we inheritance willen afzetten en de overgeërfde regels niet willen behouden, kunnen we de volgende reeks commando's uitvoeren:

```
PS C:\> $folder = 'c:\documenten\k3'
PS C:\> $acl = Get-Acl -Path $folder
```

```
PS C:\> $acl.SetAccessRuleProtection($True, $False)
PS C:\> Set-Acl -Path $folder -AclObject $acl
```

We willen deze directory als de root directory definiëren voor alle home directories van de gebruikers. De gebruikers zullen List Folder Contents nodig hebben op deze directory:

```
PS C:\> $ListFolder = New-Object
System.Security.AccessControl.FileSystemAccessRule('Builtin\users',
'ListDirectory','None','None','Allow')
PS C:\Users\jef> $acl.SetAccessRule($ListFolder)
PS C:\Users\jef> Set-Acl -Path $folder -AclObject $acl
```

Het FileSystemAccessRule object heeft de volgende argumenten:

- Builtin\users: de security principal die het recht moet krijgen
- 'ListDirectory': het recht dat moet worden toegekend
- 'None': De inheritance settings. We willen niet dat dit recht ook voor de child objecten geldt
- 'None': Propagation settings. Kan onder meer gebruikt worden om aan te geven dat het recht pas begint te gelden op de childitems
- 'Allow': het recht dat wordt toegekend

Om ervoor te zorgen dat we zelf nog folders kunnen toevoegen, zullen we de administrators full control geven en ervoor zorgen dat dit recht wordt overgeërfd door subdirectories:

```
PS C:\> $FullControl = New-Object
System.Security.AccessControl.FileSystemAccessRule('Builtin\administrators'
,'FullControl','ContainerInherit,ObjectInherit','None', 'Allow')
PS C:\> $acl.SetAccessRule($FullControl)
PS C:\> Set-Acl -Path $folder -AclObject $acl
```

3. NTFS Beveilinging via NTFSSecurity

Instellen van NTFS security vraagt een beetje typwerk. Met behulp van de NTFSSecurity module is het beheer gemakkelijker. We kunnen deze module installeren via Install-Module:

```
PS C:\Users\admin.DOM213> Install-Module NTFSSecurity

NuGet provider is required to continue
PowerShellGet requires NuGet provider version '2.8.5.201' or newer to interact with NuGet-
based repositories. The NuGet
 provider must be available in 'C:\Program Files\PackageManagement\ProviderAssemblies' or
'C:\Users\admin.DOM213\AppData\Local\PackageManagement\ProviderAssemblies'. You can also
install the NuGet provider by
running 'Install-PackageProvider -Name NuGet -MinimumVersion 2.8.5.201 -Force'. Do you want
PowerShellGet to install
and import the NuGet provider now?
[Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"): y

Untrusted repository
You are installing the modules from an untrusted repository. If you trust this repository,
change its
InstallationPolicy value by running the Set-PSRepository cmdlet. Are you sure you want to
install the modules from
'PSGallery'?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"): y
PS C:\Users\admin.DOM213>
```

We kunnen nu de NTFS Access Control List opvragen:

```
PS C:\Users\Jef> Get-NTFSAccess c:\documenten\k3


    Path: c:\documenten\k3 (Inheritance enabled)


Account                 Access Rights  Applies to               Type  IsInherited InheritedFrom
-------                 -------------  ----------               ----  ----------- -------------
BUILTIN\Administrators FullControl    ThisFolderOnly            Allow False
BUILTIN\Administrators FullControl    ThisFolderSubfoldersAn... Allow True          c:
NT AUTHORITY\SYSTEM    FullControl    ThisFolderSubfoldersAn... Allow True          c:
CREATOR OWNER          GenericAll     SubfoldersAndFilesOnly    Allow True          c:
BUILTIN\Users          ReadAndExec... ThisFolderSubfoldersAn... Allow True          c:
BUILTIN\Users          CreateDirec... ThisFolderAndSubfolders   Allow True          c:
BUILTIN\Users          CreateFiles    ThisFolderAndSubfolders   Allow True          c:
```

De overerving afzetten is nu ook eenvoudig:

```
PS C:\Users\Jef> Disable-NTFSAccessInheritance c:\documenten\k3
```

We kunnen ook een nieuw recht toekennen via AddNTFSAccess:

```
PS C:\Users\admin.DOM213> Add-NTFSAccess c:\documenten\k3 -Account
DOM213\K3 -AccessRights readandexecute
```

Om een account te verwijderen uit de lijst moeten we alle rechten vermelden. De gemakkelijkste manier om dit te doen is de output van Get-NTFSAccess te gebruiken:

```
PS C:\Users\Jef> Get-NTFSAccess c:\documenten\k3
    Path: c:\documenten\k3 (Inheritance disabled)
Account                 Access Rights  Applies to               Type IsInherited InheritedFrom
-------                 -------------  ----------               ---- ----------- -------------
CREATOR OWNER          GenericAll     SubfoldersAndFilesOnly    Allow False
NT AUTHORITY\SYSTEM    FullControl    ThisFolderSubfoldersAn... Allow False
BUILTIN\Administrators FullControl    ThisFolderSubfoldersAn... Allow False
BUILTIN\Users          CreateFiles    ThisFolderAndSubfolders   Allow False
BUILTIN\Users          ReadAndExec... ThisFolderSubfoldersAn... Allow False
BUILTIN\Users          CreateDirec... ThisFolderAndSubfolders   Allow False
DOM213\K3              ReadAndExec... ThisFolderSubfoldersAn... Allow False

PS C:\Users\Jef> Get-NTFSAccess c:\documenten\k3 -Account BUILTIN\Users|Remove-NTFSAccess
PS C:\Users\Jef> Get-NTFSAccess c:\documenten\k3


    Path: c:\documenten\k3 (Inheritance disabled)


Account                  Access Rights  Applies to               Type IsInherited
InheritedFrom
-------                  -------------  ----------               ---- -----------
--
CREATOR OWNER           GenericAll     SubfoldersAndFilesOnly    Allow False
NT AUTHORITY\SYSTEM     FullControl    ThisFolderSubfoldersAn... Allow False
BUILTIN\Administrators  FullControl    ThisFolderSubfoldersAn... Allow False
DOM213\K3               ReadAndExec... ThisFolderSubfoldersAn... Allow False
```

4. Eventlog lezen

De windows eventlog bevat meestal veel data. Het is dus belangrijk om goed te filteren, zeker wanneer we een eventlog zouden lezen over het netwerk. We maken gebruik van de meer moderne Get-WinEvent cmdlet. De oudere Get-EventLog heeft minder filtering

mogelijkheden waardoor resultaten eerst moeten worden overgehaald en vervolgens pas lokaal gefilterd kunnen worden.

De aangewezen manier om gebruik te maken van Get-WinEvent en te filteren is een hashtable. De volgende keys worden voorzien:

Key	Data type	Wildcards?
LogName	<String[]>	Ja
ProviderName	<String[]>	Ja
Path	<String[]>	Nee
Keywords	<Long[]>	Nee
Id	<Int32[]>	Nee
Level	<Int32[]>	Nee
StartTime	<DateTime>	Nee
EndTime	<DateTime>	Nee
userID	<SID>	Nee
Data	<String[]>	Nee
*****	<String[]>	Nee

Stel dat we een overzicht willen krijgen van alle errorevents (level:2) van de application eventlog(LogName:"Application") voor vandaag (StartTime:[datetime]::Today). Een hashtable wordt opgebouwd binnen @{…}:

```
PS C:\> Get-WinEvent
@{logname="application";starttime=[datetime]::today;level=2}
```

Wanneer we alle fouten van gisteren willen zien, moeten we eerst wat datums instellen:

```
PS C:\> $vandaag = [datetime]::today
PS C:\> $gisteren = $vandaag.AddDays(-1)
PS C:\> Get-WinEvent
@{LogName='Application','System';starttime=$gisteren;endtime=$vandaag;level
=2}
```

We kunnen ook de eventlog van een andere computer opvragen:

```
PS C:\> Get-WinEvent
@{logname='application','System';starttime=[datetime]::today;level=2}
-ComputerName 10.0.1.4
```

5. Netwerkbeheer

Er bestaan verschillende commando's onder windows om informatie op te vragen over het netwerk en die informatie eventueel te wijzigen: ipconfig, tracert, netsh. Powershell bouwt verder op die tools.

5. A. ipconfig

De informatie die we te pakken kunnen krijgen via ipconfig is ook in Powershell voorhanden:

```
PS C:\> Get-NetIpconfiguration


InterfaceAlias       : Ethernet 2
InterfaceIndex       : 7
InterfaceDescription : Microsoft Hyper-V Network Adapter #2
NetProfile.Name      : dom213.lokaal
IPv4Address          : 10.0.1.4
IPv6DefaultGateway   :
IPv4DefaultGateway   : 10.0.1.1
DNSServer            : ::1
                       127.0.0.1
```

Wanneer we meer info willen, kunnen we de -Detailed parameter meegeven:

```
PS C:\> Get-NetIPConfiguration -Detailed


ComputerName                            : server214
InterfaceAlias                          : Ethernet 2
InterfaceIndex                          : 7
InterfaceDescription                    : Microsoft Hyper-V Network Adapter
#2
NetCompartment.CompartmentId            : 1
NetCompartment.CompartmentDescription   : Default Compartment
NetAdapter.LinkLayerAddress             : 00-0D-3A-39-01-BF
NetAdapter.Status                       : Up
NetProfile.Name                         : dom213.lokaal
NetProfile.NetworkCategory              : DomainAuthenticated
NetProfile.IPv6Connectivity             : NoTraffic
NetProfile.IPv4Connectivity             : Internet
IPv6LinkLocalAddress                    : fe80::ac70:f19a:70bd:5d2b%7
IPv4Address                             : 10.0.1.4
IPv6DefaultGateway                      :
IPv4DefaultGateway                      : 10.0.1.1
NetIPv6Interface.NlMTU                  : 1500
NetIPv4Interface.NlMTU                  : 1500
NetIPv6Interface.DHCP                   : Enabled
NetIPv4Interface.DHCP                   : Enabled
DNSServer                               : ::1
                                          127.0.0.1
```

Wanneer we informatie willen over alle interfaces (virtuele, loopback, disconnected) kunnen we de -All parameter gebruiken:

```
PS C:\Users\Jef> Get-NetIPConfiguration -All


InterfaceAlias       : Ethernet
InterfaceIndex       : 18
InterfaceDescription : Realtek PCIe GBE Family Controller
NetProfile.Name      : dbicts.lokaal
IPv4Address          : 192.168.23.102
IPv6DefaultGateway   :
IPv4DefaultGateway   : 192.168.23.254
DNSServer            : 192.168.23.54
                       8.8.8.8

InterfaceAlias       : VirtualBox Host-Only Network #2
InterfaceIndex       : 7
InterfaceDescription : VirtualBox Host-Only Ethernet Adapter #2
IPv4Address          : 192.168.56.1
IPv6DefaultGateway   :
IPv4DefaultGateway   :
DNSServer            : fec0:0:0:ffff::1
                       fec0:0:0:ffff::2
                       fec0:0:0:ffff::3

InterfaceAlias       : LAN-verbinding* 13
InterfaceIndex       : 44
InterfaceDescription : Microsoft Teredo Tunneling Adapter
NetAdapter.Status    : Disconnected

InterfaceAlias       : LAN-verbinding* 3
InterfaceIndex       : 11
InterfaceDescription : Microsoft Wi-Fi Direct Virtual Adapter
NetAdapter.Status    : Disconnected

InterfaceAlias       : Wi-Fi
InterfaceIndex       : 3
InterfaceDescription : Intel(R) Centrino(R) Wireless-N 2230
NetAdapter.Status    : Disconnected
```

Het ipconfig commando wordt ook gebruikt om de DNS cache op de client te beheren. Daarvoor hebben we de volgende commando's:

```
PS C:\> Get-DnsClientCache

Entry                    RecordName               Record Status    Section TimeTo Data L
Data                                              Type                    Live   engthe
-----                    ----------               ------ ------    ------- ------ ------ ---
www.betavzw.org          www.betavzw.org          CNAME  Success   Answer     981      8
betavzw.org
www.betavzw.org          betavzw.org              AAAA   Success   Answer     981     16
2a01:7c8:aaac:3e9:5054:ff...
www.betavzw.org          www.betavzw.org          CNAME  Success   Answer     981      8
betavzw.org
www.betavzw.org          betavzw.org              A      Success   Answer     981      4
95.170.95.196
```

En

```
PS C:\> Clear-DnsClientCache
PS C:\> Get-DnsClientCache
```

5. B. Ping en tracert

Het ping-commando kan vervangen worden door:

```
PS C:\> Test-NetConnection www.google.be


ComputerName            : www.google.be
RemoteAddress           : 216.58.213.195
InterfaceAlias          : Ethernet 2
SourceAddress           : 10.0.1.4
PingSucceeded           : True
PingReplyDetails (RTT)  : 18 ms
```

Maar Test-NetConnection kan meer dan het ping-commando. We kunnen ook de toegang tot een poort controlleren:

```
PS C:\> Test-NetConnection www.betavzw.org -Port 443 -InformationLevel
Detailed


ComputerName            : www.betavzw.org
RemoteAddress           : 95.170.95.196
RemotePort              : 443
NameResolutionResults   : 95.170.95.196
MatchingIPsecRules      :
NetworkIsolationContext : Internet
InterfaceAlias          : Ethernet 2
SourceAddress           : 10.0.1.4
NetRoute (NextHop)      : 10.0.1.1
TcpTestSucceeded        : True
```

Voor HTTP, RDP, SMB en WINRM moeten we zelfs het poortnummer niet kennen:

```
PS C:\> Test-NetConnection server213 -CommonTCPPort RDP -InformationLevel
Detailed


ComputerName            : server213
RemoteAddress           : 10.0.1.5
RemotePort              : 3389
NameResolutionResults   : 10.0.1.5
MatchingIPsecRules      :
NetworkIsolationContext : Private Network
InterfaceAlias          : Ethernet 2
SourceAddress           : 10.0.1.4
NetRoute (NextHop)      : 0.0.0.0
TcpTestSucceeded        : True
```

Wanneer we Test-NetConnection uitvoeren zonder verdere informatie, wordt de connectie met een default server op het Internet getest. (vgl met ping www.google.be)

```
PS C:\> Test-NetConnection


ComputerName            : internetbeacon.msedge.net
RemoteAddress           : 13.107.4.52
InterfaceAlias          : Ethernet 2
SourceAddress           : 10.0.1.4
PingSucceeded           : True
PingReplyDetails (RTT)  : 4 ms
```

Via Test-NetConnection kunnen we ook het Tracert commando vervangen:

```
PS C:\Users\Jef> Test-NetConnection -TraceRoute


ComputerName            : internetbeacon.msedge.net
RemoteAddress           : 13.107.4.52
InterfaceAlias          : Ethernet
SourceAddress           : 192.168.23.102
PingSucceeded           : True
PingReplyDetails (RTT)  : 12 ms
TraceRoute              : 192.168.23.254
                          192.168.1.1
                          80.200.255.14
                          91.183.241.44
                          0.0.0.0
                          0.0.0.0
                          94.102.160.36
                          94.102.160.49
                          198.200.130.166
                          0.0.0.0
                          0.0.0.0
                          13.107.4.52
```

Voor de hops met een timeout is het IP-adres 0.0.0.0

5. C. Nslookup

Het alternatief voor nslookup is Resolve-DnsName:

```
PS C:\> Resolve-DnsName -Name betavzw.org -Type MX

Name                                           Type   TTL   Section
NameExchange                                    Preference
----                                           ----   ---   -------    ----------
--                                  ----------
betavzw.org                                    MX     10    Answer     betavzw-
org.mail.protection.outlook.com   0

Name       : betavzw-org.mail.protection.outlook.com
QueryType  : A
TTL        : 10
Section    : Additional
IP4Address : 94.245.120.74
```

```
Name       : betavzw-org.mail.protection.outlook.com
QueryType  : A
TTL        : 10
Section    : Additional
IP4Address : 213.199.154.138
```

Wanneer we een andere DNS server willen gebruiken, kunnen we dat doen via de -Server parameter:

```
PS C:\> Resolve-DnsName -Name betavzw.org -Type MX -Server 8.8.8.8

Name                                   Type   TTL   Section
NameExchange                            Preference
----                                   ----   ---   -------    ----------
--                                ----------
betavzw.org                            MX     21599 Answer     betavzw-
org.mail.protection.outlook.com   0
```

We kunnen de resolving doen zonder recursion. Wanneer het resultaat niet in de DNScache zit of niet lokaal geregistreerd is, zullen we geen antwoord krijgen:

```
PS C:\> Resolve-DnsName -Name betavzw.org -Type MX -Server 8.8.8.8

Name                                   Type   TTL   Section
NameExchange                            Preference
----                                   ----   ---   -------    ----------
--                                ----------
betavzw.org                            MX     21599 Answer     betavzw-
org.mail.protection.outlook.com   0
```

Wanneer we alleen de lokale DNS cache op de client willen gebruiken, kunnen we dat ook meegeven:

```
PS C:\> Clear-DnsClientCache
PS C:\> Resolve-DnsName -Name www.betavzw.org -CacheOnly
Resolve-DnsName : www.betavzw.org : DNS record does not exist
At line:1 char:1
+ Resolve-DnsName -Name www.betavzw.org -CacheOnly
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ResourceUnavailable: (www.betavzw.org:String)
[Resolve-DnsName], Win32Exception
    + FullyQualifiedErrorId :
RECORD_DOES_NOT_EXIST,Microsoft.DnsClient.Commands.ResolveDnsName
```

5. D. Net use

Het net use commando wordt gebruikt om drive letters te mappen. Hiervoor hebben we we de New-SmbMapping cmdlet.

```
PS C:\> New-SmbMapping -LocalPath K: -RemotePath \\server213\c$

Status Local Path Remote Path
------ ---------- -----------
```

```
OK     K:         \\server213\c$
```

De parameters -Username en -Password kunnen eventueel gebruikt worden om de connectie via een andere gebruikersnaam te maken. De parameter -Persistent kan gebruikt worden om ervoor te zorgen dat de connectie onthouden wordt.

De mappings opvragen doen we via Get-SmbMapping

```
PS C:\> Get-SmbMapping

Status Local Path Remote Path
------ ---------- -----------
OK     K:         \\server213\c$
```

6. Services beheren

Services worden in de achtergrond uitgevoerd. We kunnen een lijst opvragen via Get-Service. Wanneer we informatie willen over een bepaalde service, geven we de naam mee:

```
PS C:\ > Get-Service Dnscache

Status   Name               DisplayName
------   ----               -----------
Running  Dnscache           DNS Client
```

Wanneer we niet zeker zijn over de naam, kunnen we de where-commandlet gebruiken:

```
PS C:\> Get-Service |where {$_.Name -like '*dns*'}

Status   Name               DisplayName
------   ----               -----------
Running  Dnscache           DNS Client
```

Wanneer we de volledige informatie willen over een service, gebruiken we de Format-List commandlet:

```
PS C:\> Get-Service DnsCache |fl


Name                  : DnsCache
DisplayName           : DNS Client
Status                : Running
DependentServices     : {NcaSvc}
ServicesDependedOn    : {nsi, Tdx}
CanPauseAndContinue   : False
CanShutdown           : False
CanStop               : False
ServiceType           : Win32OwnProcess, Win32ShareProcess
```

Voor sommige eigenschappen zijn er aparte parameters voorzien:

```
PS C:\> Get-Service Dnscache -DependentServices
```

```
Status   Name               DisplayName
------   ----               -----------
Stopped  NcaSvc             Network Connectivity Assistant
```

En

```
PS C:\> Get-Service DnsCache -RequiredServices

Status   Name               DisplayName
------   ----               -----------
Running  nsi                Network Store Interface Service
Running  Tdx                Stuurprogramma voor ondersteuning v...
```

Om service te beheren, hebben we de Start-Service, Stop-Service en Restart-Service cmdlet:

```
PS C:\> restart-service browser
WARNING: Waiting for service 'Computer Browser (browser)' to stop...

PS C:\> get-service browser|fl


Name                : browser
DisplayName         : Computer Browser
Status              : Running
DependentServices   : {}
ServicesDependedOn  : {LanmanServer, LanmanWorkstation}
CanPauseAndContinue : False
CanShutdown         : False
CanStop             : True
ServiceType         : Win32OwnProcess, Win32ShareProcess
```

Met behulp van de Set-Service commandlet kunnen we bepaalde eigenschappen wijzigen, zoals bijvoorbeeld het starttype:

```
PS C:\> Get-service browser|ft name, starttype

Name    StartType
----    ---------
browser    Manual

PS C:\> Set-service browser -StartupType disabled
PS C:\> Get-service browser|ft name, starttype

Name    StartType
----    ---------
browser  Disabled
```

De Set-Service commandlet is beperkt in de mogelijkheden. Om bijvoorbeeld de logon naam te wijzigen voor een service moeten we WMI gebruiken. We zullen om te beginnen eens kijken wat we met een Win32_service object kunnen doen:

```
PS C:\> Get-WmiObject Win32_service | Get-Member -MemberType Method


   TypeName: System.Management.ManagementObject#root\cimv2\Win32_Service
```

```
Name                   MemberType Definition
----                   ---------- ----------
Change                 Method     System.Management.ManagementBaseObject
Change(System.String DisplayNam
ChangeStartMode        Method     System.Management.ManagementBaseObject
ChangeStartMode(System.String S
Delete                 Method     System.Management.ManagementBaseObject
Delete()
GetSecurityDescriptor Method     System.Management.ManagementBaseObject
GetSecurityDescriptor()
InterrogateService     Method     System.Management.ManagementBaseObject
InterrogateService()
PauseService           Method     System.Management.ManagementBaseObject
PauseService()
ResumeService          Method     System.Management.ManagementBaseObject
ResumeService()
SetSecurityDescriptor Method     System.Management.ManagementBaseObject
SetSecurityDescriptor(System.Ma
StartService           Method     System.Management.ManagementBaseObject
StartService()
StopService            Method     System.Management.ManagementBaseObject
StopService()
UserControlService     Method     System.Management.ManagementBaseObject
UserControlService(System.Byte
```

We zien hier niet allleen StartService() en StopService() methodes verschijnen. Er is ook een Change methode om een aantal eigenschappen te wijzigen. De change methode verwacht te volgende argumenten:

- DisplayName(string)
- PathName(string)
- Servicetype(int)
- ErrorControl(int)
- Startmode(string)
- Desktopinteract(boolean)
- StartUsername (string)
- StartPassword(String)
- LoadOrderGroup(string)
- LoadOrderGroupDependencies(string)
- Servicedependencies(string)

De argumenten moeten in deze volgorde worden doorgeven. Voor de argumenten die we niet willen wijzigen, gebruiken we $null als waarde:

```
PS C:\> $service.Change($null,$null,$null,$null,$null,$null,'.\Jef','pwd')


__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
```

```
__PROPERTY_COUNT : 1
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
ReturnValue      : 0
PSComputerName   :
```

Wanneer we het resultaat willen bekijken:

```
PS C:\> Get-WMIObject Win32_Service -Filter "name='xmail'" | ft
DisplayName, StartName

DisplayName  StartName
-----------  ---------
XMail Server .\Jef
```

Wanneer we de oude toestand willen terugzetten (Local System), moeten we geen paswoord meegeven voor deze ingebouwde account:

```
PS C:\>
$service.Change($null,$null,$null,$null,$null,$null,'Localsystem','')


__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 1
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
ReturnValue      : 0
PSComputerName   :


PS C:\> Get-WMIObject Win32_Service -Filter "name='xmail'" | Select -
Property DisplayName, StartName

DisplayName  StartName
-----------  ---------
XMail Server LocalSystem
```

Hoofdstuk 6. Netwerkshares beheren

1. SMB configuratie

Netwerk shares worden onder windows afhandeld via het *Server Messaging Block* protocol (SMB). We kunnen de configuratie hiervan opvragen:

```
PS C:\> Get-SmbServerConfiguration


AnnounceComment                   :
AnnounceServer                    : False
AsynchronousCredits               : 512
AuditSmb1Access                   : False
AutoDisconnectTimeout             : 15
AutoShareServer                   : True
AutoShareWorkstation              : True
CachedOpenLimit                   : 10
DurableHandleV2TimeoutInSeconds   : 180
EnableAuthenticateUserSharing     : False
EnableDownlevelTimewarp           : False
EnableForcedLogoff                : True
EnableLeasing                     : True
EnableMultiChannel                : True
EnableOplocks                     : True
EnableSecuritySignature           : True
EnableSMB1Protocol                : True
EnableSMB2Protocol                : True
EnableStrictNameChecking          : True
EncryptData                       : False
IrpStackSize                      : 15
KeepAliveTime                     : 2
MaxChannelPerSession              : 32
MaxMpxCount                       : 50
MaxSessionPerConnection           : 16384
MaxThreadsPerQueue                : 20
MaxWorkItems                      : 1
NullSessionPipes                  : ,netlogon,samr,lsarpc
NullSessionShares                 :
OplockBreakWait                   : 35
PendingClientTimeoutInSeconds     : 120
RejectUnencryptedAccess           : True
RequireSecuritySignature          : True
ServerHidden                      : True
Smb2CreditsMax                    : 8192
Smb2CreditsMin                    : 512
SmbServerNameHardeningLevel       : 0
TreatHostAsStableStorage          : False
ValidateAliasNotCircular          : True
ValidateShareScope                : True
ValidateShareScopeNotAliased      : True
ValidateTargetName                : True
```

SMB bestaat in verschillende versies. De oudste versie (SMB1) staat op de *deprecated* lijst bij Microsoft. Bij een nieuwe installatie van Windows 10 wordt het niet meer geactiveerd.

Bij een upgrade van het besturingssysteem is het echter wel aanwezig. Omdat het protocol gebruikt werd bij de installatie van *ransomware* zoals *WannaCry* en alleen verouderde besturingssystemen (Windows XP) het nog nodig hebben, is het een goed idee om het te disablen. Vanaf Windows Server 2012 R2 kan dit door een feature te verwijderen:

```
PS C:\> Remove-WindowsFeature FS-SMB1
```

Het SMB2 protocol verwijdert men best niet omdat SMB3 hiervan afhankelijk is.

2. SMB shares opvragen

We kunnen een lijst opvragen van alle gedeelde directories op de locale machine:

```
PS C:\> Get-SmbShare

Name     ScopeName Path                                        Description
----     --------- ----                                        -----------
ADMIN$   *         C:\Windows                                  Remote Admin
C$       *         C:\                                         Default share
D$       *         D:\                                         Default share
IPC$     *                                                     Remote IPC
NETLOGON *         C:\Windows\SYSVOL\sysvol\dom213.lokaal\SCRIPTS Logon server share
SYSVOL   *         C:\Windows\SYSVOL\sysvol                    Logon server share
```

Dit zijn allemaal standaard shares die worden gemaakt bij de installatie van het besturingssysteem(ADMIN$, C$, D$, IPC$) of bij het promoveren naar een Domain Controller (NETLOGON en SYSVOL).

Men kan deze default shares eventueel afzetten, maar hou er wel rekening mee dat het Distributed File System (Dfs) ze nodig heeft. Vergeet niet om de LanManServer service te herstarten:

```
PS C:\> Set-SmbServerConfiguration -AutoShareServer:$false -Confirm:$false
PS C:\> Restart-service lanmanserver
```

3. SMB share maken en configureren

We kunnen een nieuwe fileshare maken via New-SmbShare:

```
PS C:\> mkdir gedeeld


    Directory: C:\


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         5/14/2018   6:42 AM                gedeeld


PS C:\> New-SmbShare -Name delen -Path c:\gedeeld

Name  ScopeName Path       Description
----  --------- ----       -----------
delen *         c:\gedeeld
```

```
PS C:\> Set-SmbShare delen -FolderEnumerationMode AccessBased -
Confirm:$false
```

Met dit laatste commando zetten we de AccessBasedEnumeration aan. Vervolgens kunnen we de share permissions instellen. Microsoft raadt aan om iedereen Full Control te geven en de permissions in te stellen op folder niveau. Wanneer men dit niet wil, kan men de share permissions instellen via Revoke-SmbShareAccess en Grant-SmbShareAccess:

```
PS C:\> Revoke-SmbShareAccess delen -AccountName Everyone -Confirm:$flale

Name  ScopeName AccountName AccessControlType AccessRight
----  --------- ----------- ----------------- -----------
delen *         Everyone    Deny              Full


PS C:\> Grant-SmbShareAccess delen -AccountName Everyone -AccessRight
Change -Confirm:$false

Name  ScopeName AccountName AccessControlType AccessRight
----  --------- ----------- ----------------- -----------
delen *         Everyone    Allow             Change


PS C:\> Grant-SmbShareAccess delen -AccountName Administrators -AccessRight
Full -Confirm:$false

Name  ScopeName AccountName            AccessControlType AccessRight
----  --------- -----------            ----------------- -----------
delen *         Everyone               Allow             Change
delen *         BUILTIN\Administrators Allow             Full
```

Die laatste informatie kan men ook apart opvragen via GetSmbShareAccess:

```
PS C:\> Get-SmbShareAccess delen

Name  ScopeName AccountName            AccessControlType AccessRight
----  --------- -----------            ----------------- -----------
delen *         Everyone               Allow             Change
delen *         BUILTIN\Administrators Allow             Full
```

Wanneer we die informatie willen zien voor alle shares:

```
PS C:\> Get-SmbShare * | Get-SmbShareAccess | Sort-Object -Property Name |
ft -GroupBy Name


   Name: delen

Name  ScopeName AccountName            AccessControlType AccessRight
----  --------- -----------            ----------------- -----------
delen *         Everyone               Allow             Change
delen *         BUILTIN\Administrators Allow             Full


   Name: IPC$
```

```
Name ScopeName AccountName              AccessControlType AccessRight
---- --------- -----------              ----------------- -----------
IPC$ *         NT AUTHORITY\INTERACTIVE Allow             Full
IPC$ *         BUILTIN\Backup Operators Allow             Full
IPC$ *         BUILTIN\Administrators   Allow             Full
```

4. SMB connectiebeheer

Om op te vragen welke SMB connections er gemaakt zijn vanop de eigen machine kunnen we Get-SmbConnection gebruiken:

```
PS C:\> Get-SmbConnection

ServerName ShareName UserName   Credential        Dialect NumOpens
---------- --------- --------   ----------        ------- --------
server213  delen     DOM213\jef DOM213.LOKAAL\Jef 3.1.1   1
```

Vanop de server kunnen connectie informatie over de clients opvragen:

```
PS C:\> Get-SmbSession

SessionId    ClientComputerName ClientUserName NumOpens
---------    ------------------ -------------- --------
257698037789 10.0.1.4           DOM213\Jef     2
```

We kunnen ook opvragen welke bestanden geopend zijn:

```
PS C:\> Get-SmbOpenFile

FileId       SessionId    Path                             ShareRelativePath
ClientComputerName ClientUserName
------       ---------    ----                             -----------------     -------------
----- --------------
257698037829 257698037789 c:\gedeeld\                                            10.0.1.4
DOM213\Jef
257698037837 257698037789 c:\gedeeld\                                            10.0.1.4
DOM213\Jef
257698037849 257698037789 c:\gedeeld\New Text Document.txt New Text Document.txt 10.0.1.4
DOM213\Jef
```

Hoofdstuk 7. Powershell in een domain

1. Powershell en ADSI

ADSI is de afkorting van “Active Directory Service Interface”. Tot en met Windows Server 2008 was dit de manier om active directory aan te spreken via een programmeertaal zoals PowerShell. Vanaf Windows Server 2008 R2 kan men de PowerShell Active Directory Module gebruiken. Maar voor meer gevorderde manipulaties van Active Directory is kennis van ADSI nog steeds belangrijk.

Met ADSI kan men de volgende acties uitvoeren in Active Directory: een object lezen, een object wijzigen, een object creëren en een object verwijderen.

1. A. Een object opvragen via ADSI

Active Directory is een hiërarchische structuur waarbij elk object een *Distinguished Name* of DN heeft. Zoals de naam al aangeeft is een DN uniek. Hij bestaat uit verschillende onderdelen of *Relative Distinguished Names* (RDN), gescheiden door een komma. Een RDN bestaat uit een attribuut en een waarde. Enkele veel voorkomende attributen zijn:

- Domain Component (DC): een onderdeel van een domeinnaam. De domeinnaam betavzw.org heeft twee DC’s: DC=betavzw en DC=org
- Organizational Unit (OU): een organizational unit in active directory
- Common Name(CN): containers en objecten

Om te verwijzen naar een gebruiker “Joske Vermeulen” in de container “users” in het domein DOM213.LOKAAL gebruiken we de volgende Distinguished Name:

```
CN=Joske Vermeulen,CN=users,DC=DOM213,DC=LOKAAL
```

Om dit object te bewaren in een PowerShell variabele kunnen we de volgende syntax gebruiken:

```
$user = [ADSI]"LDAP:// CN=Joske Vermeulen,CN=users,DC=DOM213,DC=LOKAAL"
```

De variabele $user is een System.DirectoryServices.DirectoryEntry object. We kunnen opvragen welke members dit object heeft:

```
C:\> $user | Get-Member
   TypeName: System.DirectoryServices.DirectoryEntry
Name                        MemberType Definition
----                        ---------- ----------
ConvertDNWithBinaryToString CodeMethod static string ConvertDNWithBinaryToSt...
ConvertLargeIntegerToInt64  CodeMethod static long ConvertLargeIntegerToInt6...
accountExpires              Property   System.DirectoryServices.PropertyValu...
badPasswordTime             Property   System.DirectoryServices.PropertyValu...
badPwdCount                 Property   System.DirectoryServices.PropertyValu...
```

```
cn                          Property   System.DirectoryServices.PropertyValu...
codePage                    Property   System.DirectoryServices.PropertyValu...
countryCode                 Property   System.DirectoryServices.PropertyValu...
description                 Property   System.DirectoryServices.PropertyValu...
distinguishedName           Property   System.DirectoryServices.PropertyValu...
dSCorePropagationData       Property   System.DirectoryServices.PropertyValu...
instanceType                Property   System.DirectoryServices.PropertyValu...
isCriticalSystemObject      Property   System.DirectoryServices.PropertyValu...
lastLogoff                  Property   System.DirectoryServices.PropertyValu...
lastLogon                   Property   System.DirectoryServices.PropertyValu...
lastLogonTimestamp          Property   System.DirectoryServices.PropertyValu...
logonCount                  Property   System.DirectoryServices.PropertyValu...
logonHours                  Property   System.DirectoryServices.PropertyValu...
memberOf                    Property   System.DirectoryServices.PropertyValu...
name                        Property   System.DirectoryServices.PropertyValu...
nTSecurityDescriptor        Property   System.DirectoryServices.PropertyValu...
objectCategory              Property   System.DirectoryServices.PropertyValu...
objectClass                 Property   System.DirectoryServices.PropertyValu...
objectGUID                  Property   System.DirectoryServices.PropertyValu...
objectSid                   Property   System.DirectoryServices.PropertyValu...
primaryGroupID              Property   System.DirectoryServices.PropertyValu...
pwdLastSet                  Property   System.DirectoryServices.PropertyValu...
sAMAccountName              Property   System.DirectoryServices.PropertyValu...
sAMAccountType              Property   System.DirectoryServices.PropertyValu...
…
```

We kunnen elk van die properties opvragen:

```
C:\>$user.mail
Joske.Vermeulen@schoten.be
```

Het is belangrijk om te beseffen dat het $user object een kopie is van het eigenlijke Active Directory object:

```
$user.givenName="Marieke"
$user = [ADSI]"LDAP:// CN=Joske Vermeulen,CN=users,DC=DOM213,DC=LOKAAL"
$user.givenName
Joske
```

Wanneer we het object opnieuw opvragen uit Active Directory hebben nog steeds de oude waarde. Om de wijzigingen weg te schrijven, kunnen we gebruik maken van de CommitChanges() methode:

```
$user.givenName="Marieke"
$user.CommitChanges()
$user = [ADSI]"LDAP:// CN=Joske Vermeulen,CN=users,DC=DOM213,DC=LOKAAL"
$user.givenName
Joske
```

1. B. Een object creëren via ADSI

Om een object te creëren, moeten we eerst de container opvragen waarin het object moet worden gemaakt:

```
$container= [ADSI] "LDAP://CN=users,DC=DOM213,DC=lokaal"
```

Vervolgens kunnen we de Create()-methode loslaten op die container en het object bewaren:

```
$user = $container.Create("user", "CN=Marieke Vermeulen")
$user.CommitChanges()
```

De user is nu gedisabled gecreëerd.

1. C. Een object verwijderen via ADSI

Om een object te verwijderen, moeten we ook een verwijzing opvragen naar de container:

```
$container= [ADSI] "LDAP://CN=users,DC=DOM213,DC=lokaal"
```

Vervolgens kunnen we het object verwijderen met behulp van de RDN (Relative Distinguished Name)

```
$container.Delete("user", "CN=Marieke Vermeulen")
```

2. Installatie van Active Directory

We kunnen een server omvormen tot een domain controller via powershell. We beginnen met op zoek te gaan naar de Active Directory service die nog moet geïnstalleerd worden:

```
PS C:\> Get-WindowsFeature |Where InstallState -eq available|ft
DisplayName, Name

DisplayName                                             Name
-----------                                             ----
Active Directory Certificate Services                   AD-Certificate
Certification Authority                                 ADCS-Cert-Authority
Certificate Enrollment Policy Web Service               ADCS-Enroll-Web-Pol
Certificate Enrollment Web Service                      ADCS-Enroll-Web-Svc
Certification Authority Web Enrollment                  ADCS-Web-Enrollment
Network Device Enrollment Service                       ADCS-Device-Enrollment
Online Responder                                        ADCS-Online-Cert
Active Directory Domain Services                        AD-Domain-Services
Active Directory Federation Services                    ADFS-Federation
Active Directory Lightweight Directory Services         ADLDS
Active Directory Rights Management Services             ADRMS
Active Directory Rights Management Server               ADRMS-Server
…
```

De feature die we nodig hebben is AD-Domain-Services. We kunnen eventueel de parameter -IncludeManagementTools meegeven. Maar aangezien we alle beheer via PowerShell zullen doen, hebben we die GUI tools niet nodig.

```
PS C:\> Install-WindowsFeature AD-DOmain-Services

Success Restart Needed Exit Code      Feature Result
------- -------------- ---------      --------------
True    No             Success        {Active Directory Domain Services,
Remote ...
```

Dit heeft de bestanden geïnstalleerd om de domain controller te creëren. Maar de server is nog geen domain controller. Daarvoor zullen we eerst een module importeren. Die module is mee geïnstalleerd bij de installatie van de WindowsFeature. We kunnen die zien via:

```
PS C:\> Get-Module -ListAvailable
```

Het import commando is:

```
PS C:\> Import-Module ADDSDeployment
```

Vervolgens kunnen we een nieuwe domain controller in een nieuwe forest creëren via:

```
PS C:\> Install-ADDSForest

cmdlet Install-ADDSForest at command pipeline position 1
Supply values for the following parameters:
DomainName: dom213.lokaal
SafeModeAdministratorPassword: *******
Confirm SafeModeAdministratorPassword: *******

The target server will be configured as a domain controller and restarted
when this operation is complete.
Do you want to continue with this operation?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"):
```

We krijgen nu eventueel een paar waarschuwingen te zien en vervolgens herstart de server.

Na het herstarten, kunnen we informatie opvragen over het domein zoals bijvoorbeeld het forest functional level en het domain functional level:

```
Get-ADRootDSE |ft defaultNamingContext, domainFunctionality,
forestFunctionality

defaultNamingContext domainFunctionality forestFunctionality
-------------------- -------------------- --------------------
DC=dom213,DC=lokaal    Windows2016Domain   Windows2016Forest
```

We kunnen deze gegevens ook apart opvragen:

```
(Get-ADForest).ForestMode
Windows2016Forest
(Get-ADDOmain).DomainMode
Windows2016Domain
```

Via Set-ADForestMode kan men het forest functional level wijzigen. Sinds Windows Server 2012 R2 kan men het zelfs terug downgraden (wanneer men tenminste geen features gebruikt die niet ondersteund worden in het lagere niveau zoals de AD recycle bin):

```
Set-ADForestMode -ForestMode Windows2012R2Forest
```

```
cmdlet Set-ADForestMode at command pipeline position 1
Supply values for the following parameters:
Identity: dom213.lokaal

Confirm
Are you sure you want to perform this action?
Performing the operation "Set" on target
"CN=Partitions,CN=Configuration,DC=dom213,DC=lokaal".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"): Y
PS C:\Users\Jef> (Get-ADForest).ForestMode
Windows2012R2Forest
```

Wanneer we gebruik maken van de ingebouwde Active Directory objecten, kunnen we informatie opvragen over de *Flexible Single Master Operation Roles* (FSMO):

```
[System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()|
Select-Object *owner

PdcRoleOwner            RidRoleOwner            InfrastructureRoleOwner
------------            ------------            -----------------------
server214.dom213.lokaal server214.dom213.lokaal server214.dom213.lokaal
[System.DirectoryServices.ActiveDirectory.Forest]::GetCurrentForest()|
Select-Object *owner

SchemaRoleOwner         NamingRoleOwner
---------------         ---------------
server214.dom213.lokaal server214.dom213.lokaal
```

We kunnen dit eenvoudiger schrijven via de Get-ADDomain en Get-ADForest commandlets:

```
Get-ADDomain | select Infrastructuremaster, PDCEmulator, RIDMaster

Infrastructuremaster    PDCEmulator             RIDMaster
--------------------    -----------             ---------
server214.dom213.lokaal server214.dom213.lokaal server214.dom213.lokaal

Get-ADForest |Select DomainNamingMaster, SchemaMaster

DomainNamingMaster      SchemaMaster
------------------      ------------
server214.dom213.lokaal server214.dom213.lokaal
```

3. De Set-Location commandlet en Active Directory

Via de Set-Location commandlet (voor sommigen beter bekend als “cd”) kunnen we naar verschillende folders gaan op de harde schijf. We kunnen het echter ook voor Active Directory gebruiken. Op de harde schijf gebruiken we als driveletter C:, D:, … Voor Active Directory gebruiken we als “driveletter” “AD:”

```
PS C:\> cd ad:
PS AD:\> dir

Name                ObjectClass         DistinguishedName
----                -----------         -----------------
dom213              domainDNS           DC=dom213,DC=lokaal
Configuration       configuration       CN=Configuration,DC=dom213,DC=lokaal
Schema              dMD                 CN=Schema,CN=Configuration,DC=dom213,DC=lokaal
```

```
DomainDnsZones       domainDNS            DC=DomainDnsZones,DC=dom213,DC=lokaal
ForestDnsZones       domainDNS            DC=ForestDnsZones,DC=dom213,DC=lokaal
```

Dat maakt het werken met active directory (bijna) even eenvoudig als het werken met folders op de harde schijf. Het enige waar we rekening mee moeten houden, is dat we voor het cd-commando gebruik moeten maken van de Distinguished Name:

```
PS AD:\> cd "dc=dom213,dc=lokaal"
PS AD:\dc=dom213,dc=lokaal> dir

Name                 ObjectClass          DistinguishedName
----                 -----------          -----------------
Builtin              builtinDomain        CN=Builtin,DC=dom213,DC=lokaal
Computers            container            CN=Computers,DC=dom213,DC=lokaal
Domain Controllers   organizationalUnit   OU=Domain Controllers,DC=dom213,DC=lokaal
ForeignSecurityPr... container            CN=ForeignSecurityPrincipals,DC=dom213,DC=lokaal
Infrastructure       infrastructureUpdate CN=Infrastructure,DC=dom213,DC=lokaal
Keys                 container            CN=Keys,DC=dom213,DC=lokaal
LostAndFound         lostAndFound         CN=LostAndFound,DC=dom213,DC=lokaal
Managed Service A... container            CN=Managed Service Accounts,DC=dom213,DC=lokaal
NTDS Quotas          msDS-QuotaContainer  CN=NTDS Quotas,DC=dom213,DC=lokaal
Program Data         container            CN=Program Data,DC=dom213,DC=lokaal
System               container            CN=System,DC=dom213,DC=lokaal
TPM Devices          msTPM-Information... CN=TPM Devices,DC=dom213,DC=lokaal
Users                container            CN=Users,DC=dom213,DC=lokaal


PS AD:\dc=dom213,dc=lokaal> cd "cn=users"
PS AD:\cn=users,dc=dom213,dc=lokaal>
```

4. Computers van een domein beheren

4. A. Controleren of een computer lid is van een domein

Via de Windows Management Instrumentation kunnen we opvragen of een computer lid is van een domein:

```
PS C:\> (Get-WmiObject win32_computersystem).partofdomain
False
```

Wanneer een computer geen lid is van een domein, is hij lid van een workgroup. We kunnen de naam van die workgroup opvragen:

```
PS C:\Users\jef> (Get-WmiObject win32_computersystem).workgroup
WORKGROUP
```

In het geval van een domeincomputer, is de waarde van “workgroup” leeg.

4. B. Computer lid maken van een domein

Wanneer de firewall settings het toelaten, kunnen we een computer vanop afstand lid maken van een domein. Hiervoor moeten we eerst zorgen dat de DNS-settings in orde zijn. Dat zal meestal gebeuren via DHCP, maar we kunnen dat ook via Powershell doen. We beginnen met een lijst van de netwerkadapters op te vragen:

```
PS C:\> Get-Netadapter

Name                      InterfaceDescription                ifIndex Status
MacAddress             LinkSpeed
```

```
----                    --------------------                    ------- ------       -------
---             ---------
Ethernet 2              Microsoft Hyper-V Network Adapter #2          2 Up           00-0D-
3A-28-40-16        10 Gbps
```

De index van de adapter is twee. Vervolgens kunnen we opvragen wat het adres is van de DNS server:

```
PS C:\> Get-DnsClientServerAddress -InterfaceIndex 2

InterfaceAlias               Interface Address ServerAddresses
                             Index     Family
--------------               --------- ------- ---------------
Ethernet 2                           2 IPv4    {168.63.129.16}
Ethernet 2                           2 IPv6    {}
```

Tenslotte kunnen we de machine lid maken van het domein.

```
PS C:\> Add-Computer -ComputerName server213 -LocalCredential
"Server213\jef" -DomainName DOM213.lokaal -Credential "DOM213\a
inistrator" -Restart
```

De machine wordt automatisch herstart. Wanneer we achteraf aanmelden, zien we dat de machine lid is geworden van het domein:

```
PS C:\> (Get-WmiObject Win32_computersystem).partofdomain
True
```

4. C. Computer verwijderen uit het domein

Bij het verwijderen moet ook het computer object uit active directory verwijderd worden. Standaard wordt het object alleen gedisabled:

```
PS C:\> Remove-Computer -ComputerName server213 -UnjoinDomainCredential
"dom213\jef" -LocalCredential "server213\Jef" -Restart
Confirm
After you leave the domain, you will need to know the password of the local
Administrator account to log onto this
computer. Do you wish to continue?
[Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"): Y
```

Bij het verwijderen van een computerobject kunnen we soms een foutmelding krijgen:

```
PS C:\> Remove-ADComputer server213

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove" on target "CN=server213,CN=Computers,DC=dom213,DC=lokaal".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): Y
Remove-ADComputer : The directory service can perform the requested operation only on a leaf
object
At line:1 char:1
+ Remove-ADComputer server213
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (server213:ADComputer) [Remove-ADComputer],
ADException
```

```
    + FullyQualifiedErrorId :
ActiveDirectoryServer:8213,Microsoft.ActiveDirectory.Management.Commands.RemoveADComputer
```

Dat heeft te maken met het feit dat er nog Active directory object bewaard worden in het computer object. We lossen dat ook door die objecten ook te verwijderen:

```
PS C:\> Get-ADComputer server213 |Remove-ADObject -Recursive

Are you sure you want to remove the item and all its children?
Performing recursive remove on Target:
'CN=server213,CN=Computers,DC=dom213,DC=lokaal'.
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"): Y
```

4. D. Offline domain join

Alhoewel het niet via Powershell gebeurt, behandelen we hier ook de optie om machines lid te maken van een domein zonder dat ze een netwerkconnectie hebben. We beginnen met het volgende commando uit te voeren op de domain controller:

```
PS C:\> djoin /provision /domain dom213.lokaal /machine server213 /savefile
c:\server213.txt

Provisioning the computer...
Successfully provisioned [server213] in the domain [dom213.lokaal].
Provisioning data was saved successfully to [c:\server213.txt].
Computer provisioning completed successfully.
The operation completed successfully.
```

Dit creëert een bestand “server213.txt” dat overgebracht moet worden naar de client (server213).

Op de client voeren we het volgende commando uit. De parameter /windowspath verwijst naar de windows directory.

```
C:\>djoin /requestodj /loadfile server213.txt /windowspath %systemroot%
/localos
Loading provisioning data from the following file: [server213.txt].

The provisioning request completed successfully.
A reboot is required for changes to be applied.
The operation completed successfully.
```

We kunnen dit commando ook gebruiken om een VHD-bestand lid te maken van het domein. Dat geven we de parameter /localos niet mee.

4. E. Bepalen hoeveel computers een user kan toevoegen aan het domein

Soms vinden beheerders het raar dat gewone gebruikers een computer lid kunnen maken van het domein. Dat wordt gezien als een beveiligingsrisico. Dat is het misschien wel, maar men mag hierbij niet vergeten dat men de gebruikersnaam en het paswoord moet kennen van een domein-user om een machine lid te maken van het domein. Wanneer men die informatie al heeft (en men zou die niet mogen hebben), is er al een groter probleem.

Standaard mag een gewone gebruiker 10 machines lid maken van het domein. Wanneer men dit wat veel vindt, kan men dit aanpassen. We kunnen beginnen met de huidige instelling op te vragen:

```
PS C:\> Get-ADObject (Get-ADRootDSE).defaultnamingcontext -Properties
ms-DS-Machineaccountquota


DistinguishedName          : DC=dom213,DC=lokaal
ms-DS-Machineaccountquota : 10
Name                       : dom213
ObjectClass                : domainDNS
ObjectGUID                 : bccd4fae-8a39-4a78-a2f9-a1913cc7a1e5
```

Via Set-ADObject kunnen we de instelling aanpassen:

```
PS C:\> Set-ADObject (Get-ADRootDSE).defaultnamingcontext -Replace
@{"ms-DS-Machineaccountquota"=5}
PS C:\> Get-ADObject (Get-ADRootDSE).defaultnamingcontext -Properties
ms-DS-Machineaccountquota


DistinguishedName          : DC=dom213,DC=lokaal
ms-DS-Machineaccountquota : 5
Name                       : dom213
ObjectClass                : domainDNS
ObjectGUID                 : bccd4fae-8a39-4a78-a2f9-a1913cc7a1e5
```

De lijst is een HashTable, vandaar dat we de syntax @{…} moeten gebruiken.

5. Gebruikersbeheer via commandlets

De *-ADUser commandlets kunnen gebruikt worden om de gebruikers in een AD-domein te beheren: New-ADUser, Get-ADuser, Set-ADUser en Remove-ADuser.

5. A. Een nieuwe gebruiker creëren via New-ADUser

We kunnen veel eigenschappen meegeven bij de creatie van een gebruiker. De enige parameter die verplicht is, is de Name. Echter niet alle attributen die bestaan in active directory kunnen worden ingevuld via New-ADuser. Voor de attributen waarvoor geen parameter bestaat, kunnen we gebruik maken van -OtherAttributes:

```
New-ADUser glenjohn  -GivenName "Glen" -Surname "John" -DisplayName "Glen
John" -Path 'CN=Users,DC=fabrikam,DC=local'
-OtherAttributes @{'msDS-PhoneticDisplayName'="GlenJohn"}
```

De extra attributnamen moeten tussen enkele aanhalingstekens worden gezet.

Het paswoord van een gebruiker vraagt een aparte behandeling. We kunnen deze tekst niet meegeven zoals we dat met andere parameters kunnen doen:

```
New-ADUser Joske -AccountPassword J0skeJ0ske
```

```
New-ADUser : Cannot bind parameter 'AccountPassword'. Cannot convert the "J0skeJ0ske" value of
type "System.String" to
type "System.Security.SecureString".
At line:1 char:51
+ New-ADUser -SamAccountName Joske -AccountPassword J0skeJ0ske
+                                                   ~~~~~~~~~~
    + CategoryInfo          : InvalidArgument: (:) [New-ADUser], ParameterBindingException
    + FullyQualifiedErrorId :
CannotConvertArgumentNoMessage,Microsoft.ActiveDirectory.Management.Commands.NewADUser
```

De AccountPassword parameter is van het type SecureString en niet van het type String. We kunnen dit oplossen met de Read-Host commandlet:

```
PS C:\ > New-ADUser Joske -AccountPassword  (Read-Host -AsSecureString
"Paswoord")
```

Read-Host leest een tekst in van de command prompt. Met de parameter AsSecureString wordt de string omgezet naar een SecureString.

Standaard wordt een nieuwe gebruiker gecreëerd in de “Users” container. Met behulp van *redirusr* kunnen we die default container wijzigen (*redircmp* voor computers).

5. B. Get-ADUser en filtering

Wanneer we informatie over één of meerdere gebruikers willen opvragen, kunnen we dat doen via Get-ADuser. Wanneer we geïnteresseerd zijn in de gegevens van één gebruiker kunnen we Get-ADUser gebruiken met de Name:

```
PS C:\> Get-ADUser joske


DistinguishedName : CN=Joske,CN=Users,DC=dom213,DC=lokaal
Enabled           : False
GivenName         :
Name              : Joske
ObjectClass       : user
ObjectGUID        : 4aa9b7f6-bf65-480b-b672-b0df6c91a779
SamAccountName    : Joske
SID               : S-1-5-21-517424593-1313819611-2499439019-1104
Surname           :
UserPrincipalName : joske@dom213.lokaal
```

Maar de ware kracht van Get-ADUser zit in de filtering mogelijkheden. Er zijn drie parameters die hierbij een rol spelen:

- -SearchBase: wat is de startfolder waarin gezocht moet worden
- -SearchScope: hoe “diep” moeten we zoeken:
    - 0: Base of alleen het object zelf (een beetje raar)
    - 1: OneLevel of de folder zelf
    - 2: Subtree of de volledige (sub)hiërarchie
- -Filter: een beperking van de lijst op basis van bepaalde eigenschappen

Wanneer we alle accounts in de container “users” willen zien, kunnen we als volgt te werk gaan:

```
PS C:\> Get-ADUser -Filter * -SearchBase "CN=users,dc=dom213,dc=lokaal"


DistinguishedName : CN=Jef,CN=Users,DC=dom213,DC=lokaal
Enabled           : True
GivenName         :
Name              : Jef
ObjectClass       : user
ObjectGUID        : 226518f5-d8c3-4ce9-bb09-71203718cbb8
SamAccountName    : Jef
SID               : S-1-5-21-517424593-1313819611-2499439019-500
Surname           :
UserPrincipalName :

DistinguishedName : CN=Guest,CN=Users,DC=dom213,DC=lokaal
Enabled           : False
GivenName         :
Name              : Guest
ObjectClass       : user
ObjectGUID        : 00e64624-0aa1-4554-8912-947501b7b27a
SamAccountName    : Guest
SID               : S-1-5-21-517424593-1313819611-2499439019-501
Surname           :
UserPrincipalName :
…
```

Via de parameter -SearchBase geven we aan dat we in de container users willen kijken. De -Filter parameter legt geen verdere beperking op aan het resultaat.

Om het effect van de -SearchScope te bekijken, maken we drie organizational units:

```
PS C:\> New-ADOrganizationalUnit ou
PS C:\> New-ADOrganizationalUnit subou -Path "ou=ou,dc=dom213,dc=lokaal"
PS C:\> New-ADOrganizationalUnit subsubou -Path
"ou=subou,ou=ou,dc=dom213,dc=lokaal"
```

Vervolgens maken we twee gebruikers in elk van de sub-ou’s:

```
PS C:\> New-ADUser subjoske -Path "ou=subou,ou=ou,dc=dom213,dc=lokaal"
PS C:\> New-ADUser subsubjoske -Path
"ou=subsubou,ou=subou,ou=ou,dc=dom213,dc=lokaal"
```

Wanneer we Get-ADUser uitvoeren zonder -SearchScope, krijgen we alle gebruikers te zien :

```
PS C:\> Get-ADUser -Filter * -SearchBase "ou=ou,dc=dom213,dc=lokaal"


DistinguishedName :
CN=subsubjoske,OU=subsubou,OU=subou,OU=ou,DC=dom213,DC=lokaal
Enabled           : False
GivenName         :
```

```
Name              : subsubjoske
ObjectClass       : user
ObjectGUID        : 1eec0757-e870-4b47-b1f0-fe35f1cad777
SamAccountName    : subsubjoske
SID               : S-1-5-21-517424593-1313819611-2499439019-1602
Surname           :
UserPrincipalName :

DistinguishedName : CN=subjoske,OU=subou,OU=ou,DC=dom213,DC=lokaal
Enabled           : False
GivenName         :
Name              : subjoske
ObjectClass       : user
ObjectGUID        : 306584f0-53d1-4c8e-b958-1d77fd25c300
SamAccountName    : subjoske
SID               : S-1-5-21-517424593-1313819611-2499439019-1601
Surname           :
UserPrincipalName :
```

Dit is hetzelfde als -SearchScope “Subtree”

Met -SearchScope “Base” krijgen we niets te zien wanneer de -SearchBase een container is. We krijgen alleen iets te zien wanneer de -SearchBase een user zou zijn:

```
PS C:\> Get-ADUser -Filter * -SearchBase "ou=ou,dc=dom213,dc=lokaal" -
SearchScope Base
PS C:\> Get-ADUser -Filter * -SearchBase
"ou=subou,ou=ou,dc=dom213,dc=lokaal" -SearchScope 0
PS C:\> Get-ADUser -Filter * -SearchBase
"cn=subjoske,ou=subou,ou=ou,dc=dom213,dc=lokaal" -SearchScope Base


DistinguishedName : CN=subjoske,OU=subou,OU=ou,DC=dom213,DC=lokaal
Enabled           : False
GivenName         :
Name              : subjoske
ObjectClass       : user
ObjectGUID        : 306584f0-53d1-4c8e-b958-1d77fd25c300
SamAccountName    : subjoske
SID               : S-1-5-21-517424593-1313819611-2499439019-1601
Surname           :
UserPrincipalName :
```

In feite controleren we hier of het object bestaat. Maar veel praktisch nut heeft dit niet.

Met -SearchScope OneLevel doorzoeken we alleen de container zelf :

```
PS C:\> Get-ADUser -Filter * -SearchBase
"ou=subou,ou=ou,dc=dom213,dc=lokaal" -SearchScope OneLevel


DistinguishedName : CN=subjoske,OU=subou,OU=ou,DC=dom213,DC=lokaal
Enabled           : False
GivenName         :
Name              : subjoske
ObjectClass       : user
```

```
ObjectGUID        : 306584f0-53d1-4c8e-b958-1d77fd25c300
SamAccountName    : subjoske
SID               : S-1-5-21-517424593-1313819611-2499439019-1601
Surname           :
UserPrincipalName :
```

Get-ADusers is in feite een speciaal geval van de meer algemene Get-ADObject commandlet. We kunnen bijvoorbeeld alle ou's opvragen in het ou=ou,dc=dom213,dc=lokaal hiërarchie:

```
PS C:\> Get-ADObject -Filter { Objectclass -eq 'organizationalunit' } -
SearchBase "ou=ou,dc=dom213,dc=lokaal"

DistinguishedName                                 Name     ObjectClass
-----------------                                 ----     -----------
OU=ou,DC=dom213,DC=lokaal                         ou       organizationalUnit
OU=subou,OU=ou,DC=dom213,DC=lokaal                subou    organizationalUnit
OU=subsubou,OU=subou,OU=ou,DC=dom213,DC=lokaal    subsubou organizationalUnit
```

Het resultaat van een Get-commandlet wordt dikwijls gecombineerd met een Set-commandlet. We zouden de vorige lijst kunnen gebruiken om de optie "protected from accidental deletion" kunnen afzetten:

```
PS C:\> Get-ADObject -Filter { Objectclass -eq 'organizationalunit' } -
SearchBase "ou=ou,dc=dom213,dc=lokaal"|Set-ADOrga
nizationalUnit -ProtectedFromAccidentalDeletion:$false
```

Vervolgens kunnen we de volledige hiërarchie (met alle gebruikers) verwijderen.

De -Filter parameter heeft een ietwat speciale syntax. Een filterwaarde moet tussen accolades staan. Een filterwaarde bestaat uit één of meerdere voorwaarden. Wanneer er meerdere voorwaarden zijn, worden die gecombineerd met een "-and" of een "-or" operator.

Een voorwaarde bestaat uit een attribuut, een operator en een waarde. De volgende operators worden ondersteund:

Filter	**Betekenis**
-eq	Gelijk aan
-ne	Niet gelijk aan
-gt	Strikt groter dan
-ge	Groter of gelijk aan

Filter	Betekenis
-lt	Strikt kleiner dan
-le	Kleiner of gelijk aan
-like	Zoals, maakt gebruikt van de * wildcard voor een willekeurig aantal willekeurige letters
-notlike	Niet zoals, maakt gebruikt van de * wildcard voor een willekeurig aantal willekeurige letters

5. C. Search-ADAccount

Voor bepaalde filtervoorwaarden op het gebied van de status van de account en het paswoord kan men gebruik maken van Search-ADAccount. Enkele voorbeelden:

Toon een lijst van alle gedisablede gebruikers:

```
Search-ADAccount -AccountDisabled -UsersOnly
```

Toon een lijst met alle expired gebruikers:

```
Search-ADAccount -AccountExpired -UsersOnly
```

Toon een lijst met alle gebruikers die binnen 6 dagen expired zullen zijn:

```
Search-ADAccount -AccountExpiring -TimeSpan 6.00:00:00 -UsersOnly
```

Toon een lijst met alle gebruikers waarvan het paswoord verlopen is:

```
Search-ADAccount -PasswordExpired -UsersOnly
```

Toon een lijst met alle gelockte gebruikers:

```
Search-ADAccount -LockedOut -UsersOnly
```

Toon een lijst met alle gebruikers die zich de laatste 60 dagen niet hebben aangemeld:

```
Search-ADAccount -AccountInactive -UsersOnly -TimeSpan "60"
```

5. D. Paswoord van een gebruiker veranderen

De eenvoudigste manier om het paswoord van een gebruiker te wijzigen is via Set-ADPassword:

```
PS C:\> Set-ADAccountPassword joske
Please enter the current password for 'CN=Joske,CN=Users,DC=dom213,DC=lokaal'
Password: *******
Please enter the desired password for 'CN=Joske,CN=Users,DC=dom213,DC=lokaal'
Password: ********
Repeat Password: ********
```

Hier hebben we echter nog steeds het oude paswoord nodig. Via de -reset parameter hoeft dat niet meer. Maar dan moeten we natuurlijk wel het recht hebben om het paswoord te resetten:

```
PS C:\> Set-ADAccountPassword joske -reset
Please enter the desired password for 'CN=Joske,CN=Users,DC=dom213,DC=lokaal'
Password: *******
Repeat Password: *******
```

Wanneer we het paswoord niet willen intypen via een prompt, moeten we het meegeven als een SecureString:

```
PS C:\> Set-ADAccountPassword joske  -Reset -NewPassword (ConvertTo-
SecureString -AsPlainText "Rr00ttt" -Force)
```

De -Force parameter is verplicht wanneer we de -AsPlainText parameter gebruiken.

6. Beheer van groepen

6. A. Standaardbeheer

We kunnen een nieuwe groep maken in Active Directory via New-ADGroup. De -GroupScope parameter is verplicht:

```
PS C:\> New-ADGroup K3

cmdlet New-ADGroup at command pipeline position 1
Supply values for the following parameters:
GroupScope: Global
```

Via de -Path parameter kunnen we de container bepalen waarin de groep terecht moet komen.

De leden van een groepen kunnen we beheren via Add-ADGroupMember. De parameter -Members kan een lijst van gebruikers bevatten (gescheiden door een komma):

```
PS C:\> Add-ADGroupMember K3 -Members joske
PS C:\> Get-ADGroupMember K3


distinguishedName : CN=Joske,CN=Users,DC=dom213,DC=lokaal
name              : Joske
```

```
objectClass       : user
objectGUID        : 4aa9b7f6-bf65-480b-b672-b0df6c91a779
SamAccountName    : Joske
SID               : S-1-5-21-517424593-1313819611-2499439019-1104
```

Het commando Get-ADGroupMember toont standaard alleen de directe leden. Wanneer een groep lid is van een andere groep kunnen we leden van die "membergroep" opvragen via -Recursive:

```
PS C:\> Get-ADGroupMember administrators -Recursive


distinguishedName : CN=administrator,CN=Users,DC=dom213,DC=lokaal
name              : administrator
objectClass       : user
objectGUID        : af5f7fa7-c5e2-478d-bacf-4f0cbc51191e
SamAccountName    : administrator
SID               : S-1-5-21-517424593-1313819611-2499439019-1603

distinguishedName : CN=Jef,CN=Users,DC=dom213,DC=lokaal
name              : Jef
objectClass       : user
objectGUID        : 226518f5-d8c3-4ce9-bb09-71203718cbb8
SamAccountName    : Jef
SID               : S-1-5-21-517424593-1313819611-2499439019-500

distinguishedName : CN=Joske,CN=Users,DC=dom213,DC=lokaal
name              : Joske
objectClass       : user
objectGUID        : 4aa9b7f6-bf65-480b-b672-b0df6c91a779
SamAccountName    : Joske
SID               : S-1-5-21-517424593-1313819611-2499439019-1104
```

Gebruikers verwijderen uit een groep gebeurt via Remove-ADGroupMember:

```
PS C:\> Remove-ADGroupMember K3 -Members joske

Confirm
Are you sure you want to perform this action?
Performing the operation "Set" on target
"CN=K3,CN=Users,DC=dom213,DC=lokaal".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"): Y
PS C:\> Get-ADGroupMember K3
```

We kunnen ook opvragen van welke groepen een gebruiker lid is:

```
PS C:\> Get-ADPrincipalGroupMembership Joske


distinguishedName : CN=Domain Users,CN=Users,DC=dom213,DC=lokaal
GroupCategory     : Security
GroupScope        : Global
name              : Domain Users
objectClass       : group
objectGUID        : 10499f60-18ec-4687-826a-a3d7334df8aa
```

```
SamAccountName    : Domain Users
SID               : S-1-5-21-517424593-1313819611-2499439019-513

distinguishedName : CN=K3,CN=Users,DC=dom213,DC=lokaal
GroupCategory     : Security
GroupScope        : Global
name              : K3
objectClass       : group
objectGUID        : f67baa94-c86b-428e-a27f-54fb28f4f0e8
SamAccountName    : K3
SID               : S-1-5-21-517424593-1313819611-2499439019-1604
```

6. B. Groepen in Windows Server 2016: Time-To-Live

Vanaf Forest functional level Windows Server 2016 kunnen we een maximum duurtijd meegeven voor het lidmaatschap van groepen. Hiervoor moet de Privileged Access Management Feature geactiveerd worden:

```
PS C:\> Enable-ADOptionalFeature 'Privileged Access Management Feature' -
Scope ForestOrConfigurationSet -target dom213.lokaal
WARNING: Enabling 'Privileged Access Management Feature' on
'CN=Partitions,CN=Configuration,DC=dom213,DC=lokaal' is
an irreversible action! You will not be able to disable 'Privileged Access
Management Feature' on
'CN=Partitions,CN=Configuration,DC=dom213,DC=lokaal' if you proceed.

Confirm
Are you sure you want to perform this action?
Performing the operation "Enable" on target "Privileged Access Management
Feature".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"): Y
```

We kunnen dit controleren:

```
PS C:\> Get-ADOptionalFeature -filter {name -like "Privi*"}


DistinguishedName  : CN=Privileged Access Management Feature,CN=Optional
Features,CN=Directory Service,CN=Windows NT,CN
                     =Services,CN=Configuration,DC=dom213,DC=lokaal
EnabledScopes      : {CN=Partitions,CN=Configuration,DC=dom213,DC=lokaal,
CN=NTDS Settings,CN=server214,CN=Servers,CN=D
                     efault-First-Site-
Name,CN=Sites,CN=Configuration,DC=dom213,DC=lokaal}
FeatureGUID        : ec43e873-cce8-4640-b4ab-07ffe4ab5bcd
FeatureScope       : {ForestOrConfigurationSet}
IsDisableable      : False
Name               : Privileged Access Management Feature
ObjectClass        : msDS-OptionalFeature
ObjectGUID         : 64fb17a7-afcf-4b8b-ba3b-009cdc83804b
RequiredDomainMode :
RequiredForestMode : Windows2016Forest
```

Vervolgens kunnen we een time-to-live object en een gebruiker lid maken gedurende een bepaalde periode:

```
PS C:\> $ttl = New-TimeSpan -Minutes 1
PS C:\> Add-ADGroupMember K3 -Members joske -MemberTimeToLive $ttl
PS C:\> Get-ADGroup k3 -Properties Member -ShowMemberTimeToLive


DistinguishedName : CN=K3,CN=Users,DC=dom213,DC=lokaal
GroupCategory     : Security
GroupScope        : Global
Member            : {<TTL=32>,CN=Joske,CN=Users,DC=dom213,DC=lokaal}
Name              : K3
ObjectClass       : group
ObjectGUID        : f67baa94-c86b-428e-a27f-54fb28f4f0e8
SamAccountName    : K3
SID               : S-1-5-21-517424593-1313819611-2499439019-1604
```

Na een minuut zal Joske geen lid meer zijn van de groep.

7. Gebruikers in bulk toevoegen

Eén gebruiker creëren gaat waarschijnlijk het eenvoudigste via de GUI-tools ADUC (Active Directory Users and Computers) of ADAC (Active Directory Administrative Center). Maar wanneer we meerdere gebruikers moeten toevoegen, kan een script dat een lijst van gebruikers leest handiger zijn.

De structuur van het importbestand is een CSV-formaat:

```
Voornaam;Achternaam
Karen;Damen
Kristel;Verbeke
Kathleen;Aerts
```

We zullen beginnen met een Organizational Unit (K3) te maken voor de drie gebruikers in de root van het domein Dom213.Lokaal:

```
PS C:\> New-ADOrganizationalUnit K3 -Path "DC=DOM213,DC=lokaal"
```

Vervolgens maken we het script:

```
Set-StrictMode -Version Latest
$gebruikers = Import-Csv -Delimiter ";" .\users.csv
$domein = Get-ADDomain -Current LocalComputer
$OU = "OU=K3,"+$domein.DistinguishedName
$paswoord = ConvertTo-SecureString "Rr00ttt" -AsPlainText -Force
foreach ($gebruiker in $gebruikers){
   $Displayname = $gebruiker.Voornaam + " " + $gebruiker.Achternaam
   $Firstname = $gebruiker.Voornaam
   $Lastname = $gebruiker.Achternaam
   $SAM = $gebruiker.Voornaam
   $UPN = $gebruiker.Voornaam + "." + $gebruiker.Achternaam + "@" + $domein.DNSRoot
   New-ADUser -Name $Displayname -DisplayName $Displayname -SamAccountName $SAM `
              -UserPrincipalName $UPN -GivenName $Firstname -Surname $Lastname `
              -AccountPassword $paswoord -Path $OU -ChangePasswordAtLogon $True `
              -Enabled $true
}
```

- We beginnen met de “strict mode” te enablen. Hierdoor voorkomen we onder meer dat er niet-geïnitialiseerde variabelen worden gebruikt.
- Vervolgens importeren we het bestand. Het heet een comma-separated-value betand, maar wij gebruiken meestal een puntkomma als scheidingsteken
- Het domein object bevat informatie over het huidige domein. We hebben de DNS-naam nodig (dom213.lokaal) en de distinguishedname (DC=DOM213,DC=lokaal)
- De naam van de OU bestaat uit “OU=K3” gevolgd door de distinguishedname van het domein.
- Elke gebruiker krijgt hetzelfde paswoord. We zorgen ervoor dat een gebruiker zijn/haar paswoord moet wijzigen bij de eerste aanmelding.

Dit zijn alle instellingen die algemeen zijn voor de gebruikers. Vervolgens overlopen we de collection die ingelezen is uit het CSV-bestand. De collection bestaat uit een aantal objecten die als properties de headings hebben uit het CSV-bestand (Voornaam en Achternaam). Op basis van die informatie creëren we de items die we nodig hebben voor de New-ADUser cmdlet:

- De Fullname en de DisplayName bestaan uit de voornaam en de achternaam, gescheiden door een spatie
- De samaccountname bestaat uit de voornaam
- De User Principal Name is opgebouwd als voornaam.achternaam@dnsdomainnaam

Het New-ADUser commando maakt gebruikt van deze items om de users te creëren. We zorgen ervoor dat ze enabled zijn en dat ze hun paswoord moeten aanpassen bij de eerste aanmelding. We controleren via Get-ADUser of de gebruikers effectief zijn gemaakt:

```
PS C:\> .\bulkimport.ps1
PS C:\> Get-ADUser -Filter * -SearchBase "OU=K3,DC=DOM213,DC=lokaal"


DistinguishedName : CN=Karen Damen,OU=K3,DC=dom213,DC=lokaal
Enabled           : True
GivenName         : Karen
Name              : Karen Damen
ObjectClass       : user
ObjectGUID        : 70287e1d-61dd-4559-9e41-9ade13252e23
SamAccountName    : Karen
SID               : S-1-5-21-895315077-4074179417-748770018-1109
Surname           : Damen
UserPrincipalName : Karen.Damen@dom213.lokaal

DistinguishedName : CN=Kristel Verbeke,OU=K3,DC=dom213,DC=lokaal
Enabled           : True
GivenName         : Kristel
Name              : Kristel Verbeke
ObjectClass       : user
ObjectGUID        : 0ea5f7c7-6184-43c6-a70c-c167d93a35da
SamAccountName    : Kristel
SID               : S-1-5-21-895315077-4074179417-748770018-1110
Surname           : Verbeke
UserPrincipalName : Kristel.Verbeke@dom213.lokaal
```

```
DistinguishedName : CN=Kathleen Aerts,OU=K3,DC=dom213,DC=lokaal
Enabled           : True
GivenName         : Kathleen
Name              : Kathleen Aerts
ObjectClass       : user
ObjectGUID        : 65deaaff-0eb9-4fce-a0db-b157e0dd0d09
SamAccountName    : Kathleen
SID               : S-1-5-21-895315077-4074179417-748770018-1111
Surname           : Aerts
UserPrincipalName : Kathleen.Aerts@dom213.lokaal
```

In een aangepase versie zouden we rekening kunnen houden met het feit dat sommige gebruikers al bestaan. We zullen die namen wegschrijven in errors.csv:

```
Set-StrictMode -Version Latest
$gebruikers = Import-Csv -Delimiter ";" .\users.csv
$domein = Get-ADDomain -Current LocalComputer
$OU = "OU=K3,"+$domein.DistinguishedName
$paswoord = ConvertTo-SecureString "Rr00ttt" -AsPlainText -Force
$errors= @()
foreach ($gebruiker in $gebruikers){
   $Displayname = $gebruiker.Voornaam + " " + $gebruiker.Achternaam
   $Firstname = $gebruiker.Voornaam
   $Lastname = $gebruiker.Achternaam
   $SAM = $gebruiker.Voornaam
   $UPN = $gebruiker.Voornaam + "." + $gebruiker.Achternaam + "@" + $domein.DNSRoot
   try{
      New-ADUser -Name $Displayname -DisplayName $Displayname -SamAccountName $SAM`
      -UserPrincipalName $UPN -GivenName $Firstname -Surname $Lastname `
      -AccountPassword $paswoord -Path $OU -ChangePasswordAtLogon $True `
      -Enabled $true
   }Catch [Microsoft.ActiveDirectory.Management.ADIdentityAlreadyExistsException]{
        Write-Error "User $Displayname bestaat al"
        $errors += $gebruiker
   }
}
If ($errors.Count -gt 0){
   $errors |Export-Csv -Path "Errors.csv" -Delimiter ";"
}
```

We vangen de “IdentityAlreadyExistsException” op en voegen de gebruiker toe aan een array van errors. Achteraf schrijven we de array weg op een manier die achteraf terug gebruikt kan worden als input voor het script.

Hoofdstuk 8. Overzicht Powershell syntax

Dit hoofdstuk is opgevat als een naslagwerk voor de Powershell syntax. De details van de verschillende onderdelen wordt in andere hoofdstukken beschreven.

1. Commentaar

Commentaar wordt niet gelezen door de Powershell Engine. We kunnen hiermee extra informatie toevoegen aan een script voor menselijke gebruikers. Er zijn twee soorten commentaar:

- Lijncommentaar met het #-teken. Alles vanaf het #-teken tot het einde van de regels wordt beschouwd als commentaar
- Block commentaar: <#...#>. Alles vanaf het <#-teken tot en met het #>-teken wordt beschouwd als commentaar. Een block kan zich uitstrekken over meerdere regels.

2. Speciale tekens

NAAM	TEKEN
STATEMENT SEPARATOR	;
CALL OPERATOR	&
DOT-SOURCE OPERATOR	.
LINE CONTINUATION	`
ESCAPE CHARACTER	`

De *backtick* wordt niet alleen gebruikt om een commando over meerdere regels te spreiden, maar is ook het escape character in strings

BESCHRIJVING	TEKST	ASCII CODE
NULL	`0	0
BELL	`a	7
BACKSPACE	`b	8
TAB	`t	9
LINE FEED	`n	10

BESCHRIJVING	TEKST	ASCII CODE
CARRIAGE RETURN	`r	13

3. Enkele operatoren

NAAM	OPERATOR
EQUAL TO	-eq
NOT EQUAL TO	-ne
AND	-and
OR	-or
OPTELLEN/CONCATENATION	+
SUBEXPRESSION OPERATOR	$( )

4. Arrays en hashtabellen

NAAM	SYNTAX	VOORBEELD
ARRAY OPERATOR	@()	$array = @() #lege array $array = @(1, 2, 3)
IMPLICIETE ARRAY	Waarde1, waarde2	$array = 1, 2, 3, 4 $array = "een", "twee", "drie"
HASHTABEL OPERATOR	@{ }	$hashtable = @{} #lege hashtable $hashtable = @{Key1 = "Value"; Key2 = "Value2"}

5. Strings

NAAM	SYNTAX	VOORBEELD
EXPANDING STRING	" "	"Waarde" $tekst="Dag" $begroeting = "$tekst allemaal"
EXPANDING HERE STRING	@" "@	$var="variabele" @" Tekst op meerdere regels Met $var (kan " en ' bevatten) "@

NAAM	SYNTAX	VOORBEELD
NON EXPANDING STRING	' '	$tekst='Dag' $geenbegroeting = '$tekst allemaal'
NON EXPANDING HERE STRING	@' '@	$var="variabele" @' Tekst op meerdere regels Zonder $var (kan " en ' bevatten) '@
QUOTES IN STRINGS	"" `" '' `'	$string = """Dag `"' $string = '''Dag `''

6. Voorbehouden variabelen

variabele	Beschrijving
$Error	Array met errors
$Host	Informatie over de huidige Powershell host
$Matches	Overeenkomsten die gevonden werden na de -match operator
$Ofs	Output field operator. Bepaalt hoe array elementen worden samengevoegd (standard is een spatie)
$PID	Proces ID van de huidige Powershell
$PROFILE	Path naar elk van de profile bestanden
$PSVersiontable	Tabel met versie informatie van Powershell
$PWD	Huidige directory

Bibliografie

Battalio, Erix. 2017. Windows Powershell ISE Add-ON Tools. *Technet.* [Online] Microsoft, 21 november 2017. [Citaat van: 6 april 2018.] https://social.technet.microsoft.com/wiki/contents/articles/2969.windows-powershell-ise-add-on-tools.aspx.

Hunter, Laura E. en St. Cyr, Ken. 2001. *Automating Active Directory Administration with Windows Powershell 2.0.* sl : Sybex, 2001. 978-1-118-02731-8.

Lee, Thomas. 2017. *Windows Server 2016 Automation with Powershell Cookbook.* sl : Pack Publishing, 2017. 978-1-787-12204-8.

Modimoto, Rand, Shapiro, Jeffrey en Yardeni, Guy. 2017. *Windows Server 2016 Unleashed.* sl : Pearson Education, Inc., 2017. 978-0-13-458375-4.

Yellapragada, Uma. 2015. *Active Directory with Powershell.* sl : Packt Publising, 2015. 978-1-782-17599-5.

De meest recente versie van deze referentielijst kan geraadpleegd worden op http://www.betavzw.org/wnopleidingen/powershell