Samenvatting powershell commando’s

Hoofdstuk : objecten

* Get-Help [-Name] Get-Command **-Full | More**
* Get-Help [-Name] Get-Command **-Detailed**
* Get-Help [–Name] Get-Command **-Examples**
* Get-Help [-Name] Get-Command **-Online**
* Get-Help [-Name] Get-Command **–Parameter <System.String[]>**   
  example Get-Help Get-Command **name**
* Get-Help [-Name] Get-Command **-ShowWindow**
* help
* help command
* help process
* Werkwoord (**verb) en zelfstandig naamwoord (noun)**
  + Get-Help
  + Write-Host
  + Get-Process
  + Stop-Service
* Windows services : programmas die op **de achtergrond draaien**
* Get-Service -Name w32time
* Get-Service -Name w32time | **Get-Member**
* Get-Service -Name w32time | **Select-Object -Property** **\***
* Get-Service -Name w32time | select \*
* Get-Service -Name w32time | **Get-Member -MemberType Property**
* Get-Service -Name w32time | **Select-Object -Property Status, Name, DisplayName, ServiceType**
* De get-member => geeft **alle mogelijke properties en methodes** van het object
* De select-object of select => geeft de **waarde van de properties en methodes** dus wat er in de methodes en properties zitten
* Get-Process notepad | Stop-Process
* Get-Service -Name BITS | Restart-Service
* Get-ChildItem -Path \*.txt |Where-Object {$\_.length -gt 10000} | Sort-Object -Property length|**Format-Table -Property name, length**
* Methodes tonen : Get-Service -Name w32time | Get-**Member -MemberType Method**
* Methodes gebruiken : **(Get-Service -Name w32time).Start()**
* Get-Service -Name spooler
  + Status van de spooler service
* Stop-Service -Name spooler
  + stopt de print spooler service
* Start-Service -Name spooler
  + start de print spooler service
* Suspend-Service -Name spooler
  + **pauzeert** de print spooler service
* Restart-Service -Name spooler
  + herstart de print spooler service
* *herstart alle running services (maar enkel de running services die kunnen stoppen natuurlijk)*
* **Get-Service | Where-Object -FilterScript {$\_.CanStop} | Restart-Service**

Hoofstuk3 : filters

* Get-Service | Where-Object Status -eq "Stopped"  
  Get-Service | where Status -eq "Stopped"
* Get-Service | where Status -eq "Stopped"
* Get-Service | Where-Object {**$\_.Status -eq** "Stopped"}
* Get-Process | where CPU -**gt** 60
* Get-Process | where CPU -gt 60
* Get-Service | where CanStop -eq **$true => dit betekent dat de object kan stoppen**
* Stapsgewijze filteren :
  + welke properties bestaan er ook al weer?
  + Get-Service | Get-Member
  + Get-Service **| where Status -eq "Running"**
* Get-Service | where Status -eq "Running" | where CanStop -eq $false
* Get-Service | Where-Object { ($\_.Status -eq "Running") -and ($\_.CanStop -eq $false) } => bij and moeten alle 2 voorwaardes juist zijn
  + **-and**
  + **-or**
  + **-not** *of* **!**
* Operatoren:
  + -eq => equal, hoofdletterongevoelig
  + -ceq => equal maar hooflettergevoelig
  + -ne
  + -cne
  + -gt
  + -ge
  + -lt
  + -le
  + -contains
  + -ccontains
  + -notcontains
  + -cnotcontains
  + -in
  + -cin
  + -notin
  + -cnotin => “abc” -cnotin “abc”,”def”
  + -like,-clike
  + -notlike,cnotlike
  + -match,-cmatch
  + -notmatch,-cnotmatch

Hoofdstuk 4 : objecten sorteren , tellen

* Get-Process
* Get-Process | **Sort-Object -Property CPU**
* Get-Process | sort CPU
* Get-Process | Sort-Object -Property CPU –Descending
* Get-Process | **sort CPU -Descending**
* Get-Process | Get-Member
  + kijken welke Properties er allemaal bestaan in een Process object
* Get-Process | select Name,ID,VM,PM
  + we maken een selectie
* Get-Process | **select Name,Id,PM,NPM | sort VM –Descending**
  + en we sorteren
* Get-Process | sort VM -Descending | **select -First 5**
* Get-Process | sort VM -Descending | **select -Last 6**
* Get-ChildItem | Measure-Object
  + tel het aantal files en folders
* Get-ChildItem | **Measure-Object -Character -Line -Word**
* Get-Content filelist.txt | Measure-Object -Character -Line -Word
  + tel het aantal karakters, regels en woorden van het aangereikte object

hoofdstuk 5 : formatering en output

* Write-Host
  + schrijft enkel naar de console
  + maar, formatting met kleuren
  + gebruik Write-Host om de tekst te tonen op het scherm, bv foutboodschappen, die je niet wil pipe
* Write-Host "Hallo daar!"
* Write-Host "Alles goed?" -ForegroundColor Red -BackgroundColor Yellow
* Write-Output of write (of echo)
  + schrijft naar de pipeline (goed!)
  + bv stuur naar het volgende commando
  + bv stuur naar een log file
  + bv stuur naar het scherm
  + maar, geen kleuren
* Clear-Host of clear of cls
* Formatterings commando's
  + Format-Table
  + Format-List
  + Format-Wide
  + Format-Custom
* Get-Service W32Time | select Name, Status, StartType, CanStop, CanShutdown | Format-Table
* Get-Service w32time | Format-List => BELANGRIJK de fotmat commandos moet altijd op het einde het is altijd de laatste stap
* Get-Process | Out-File .\Process.txt
  + Get-Content .\Process.txt
* Get-Process > .\Process.txt
* Get-Process >> .\Processes.txt => je gaat hier appenden
* Get-Service | ConvertTo-Html | Out-File services.html
  + Invoke-Item services.html
* Get-WinEvent 'Application' -MaxEvents 100 | ConvertTo-Html | Out-File appeventlog.html

Hoofstuk 6 : files and directories

* Get-ChildItem of ls of dir
* Verborgen files (hidden files of system files) worden ook afgegaan met Get-ChildItem -Force
  + Get-ChildItem -Force C:\Windows
* Om de directory of de file te onderzoekn : Get-Item C:\Windows\
* Met Get-Child-Item -Recurse
* Get-ChildItem C:\Windows -Recurse -Force **gaat alles doorheen ook de hidden directories**
* Get-ChildItem C:\Windows -Recurse -Force -ErrorAction Ignore | measure
* Get-ChildItem C:\Windows -Recurse -Force -ErrorAction Ignore | measure
* Get-ChildItem -Hidden **lijst enkel de verborgen files op van het meegegeven pad.**
* Get-ChildItem -System **lijst enkel de system files op van het meegegeven pad**.
* Get-ChildItem C:\Windows\ | **measure Length -Sum** => toont de totale file size
* Get-ChildItem C:\Windows\\*.exe
* Get-ChildItem C:\Techdocs\[a-d]\*.txt => dus de alphabet van a tot en met d moet matchen
  + \* betekent 0 of meer
  + ? match 1 karakter
  + [a-l] van a tot en met l
  + [bc]=> moet met b matchen of met c matchen
* Get-ChildItem C:\Windows\ -Filter \*.exe
* Get-ChildItem C:\Windows\**\*** **-Include \*.ini,\*.dll met -include kun je meerdere wildcard filteren je moet wel een \* in de pad gebriuken**
* Get-ChildItem C:\Windows\**\*** -Include \*.ini,\*.dll **-Exclude w\***
* Get-ChildItem C:\Windows -Directory **geeft enkel directoryinfo terug**
* Get-ChildItem C:\Windows -File
* Get-ChildItem c:\windows\ | select CreationTime, Name | where CreationTime -gt "07/01/21" => **toon alle files en dirs. Die ouder zijn dan 07/01/21 die in de map c:\windows\**
* Get-Childitem -Path 'C:\Program Files' -Recurse | Where-Object ` {$\_.PSIsContainer -eq $True => de psicontainer controleer je of het object een folder is
* **Get-ChildItem 'C:\ Program Files' -Recurse | Where-Object {$\_.PSIsContainer -eq $True } | Where-Object {$\_.GetFiles().Count -eq 0} Select-Object Fullname**
  + Stap 1 : er word eerst door alle objecten gegaan in de map c:\program files
  + Stap2 : dan worden alleen de objecten gekozen die folders zij
  + Stap3 : dan worden alleen de folders getoond op de output scherm als ze geen files hebben in hen folder
* **Get-ChildItem 'C:\Program Files' -Recurse | Where-Object {$\_.PSIsContainer -eq $True -and $\_.GetFiles().Count -eq 0} | `Select-Object Fullname**
* Bestanden selecteren die gemaakt zijn voor 31-12-2013:
  + Get-ChildItem -Path 'c:\data' -Recurse -Force | Where-Object {!$\_.PSIsContainer -and $\_.CreationTime -lt '12-31-2013'}
* **mkdir** C:\powershell\TestDirectory
* cd C:\powershell
* mkdir TestDirectory1,TestDirectory2,TestDirectory3
* copieer files met **Copy-Item** of **copy** of **cp**
  + **cp C**:\Windows\notepad.exe C:\powershell
* overschrijf files of directories tijdens het copieren met **cp -Force**
  + cp -Force C:\Windows\notepad.exe C:\powershell
* Directories met de inhoud copieren doe je met **cp -Recurse**
  + cp -Recurse C:\powershell c:\powershell2
* overschrijven verplicht je met **-Force**
  + cp -Recurse C:\powershell c:\powershell2 -Force
* rm -Recurse -Force C:\powershell2
* mv C:\powershell C:\bazinga => veranderd de naam
* mv C:\bazinga C:\powershell => veranderd de naam
* mkdir c:\testje
* mv C:\powershell C:\testje\nieuwe\_naam => verplaatst
* mv C:\testje\nieuwe\_naam c:\powershell => verplaatst

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

* **Set-Location HKCU:**
  + **Get-childitem**
* **Set-Location ENV:**
  + **Get-childitem**
* $ENV:windir => hoe je $ENV kunt gebruiken
* **Set-Location Alias:** en vervolgens **Get-ChildItem**)
* (**Set-Location Function:** en vervolgens **Get-ChildItem**)
* (**Set-Location Cert:** en vervolgens **Get-ChildItem**)
* Om weer terug te keren typ je **C:** of **Set-Location C:**

Hoofdstuk 7 : variabelen

* $a = “hallo”
* $b = 5
* $a = Get-Date
  + $a | Get-Member
* $a.Date => toont de inhoud van de property date
* $a = Get-Service W32Time
  + $a | Get-Member
  + $a | select Name, StartType, Status
  + $a.Name
  + $a.StartType
  + $a.Status
* $MijnProcessen = Get-Process
* $MijnProcessen | Format-Table Id,Name
* $MijnProcessen | select Id, Name
* $MijnProcessen.Name
* $MijnService = Get-Service W32Time
  + Write-Output "Naam van het object:" $MijnService.Name
  + Write-Output "Status van het object:" $MijnService.Status
* $MijnService = Get-Service W32Time , variable in een string :
  + **Write-Output "De naam van het object is $($MijnService.Name) en de status is $($MijnService.Status)."**
* Write-Output "2 + 3 is $(2 + 3)“
* **GetType()**
  + $MijnVariabele = 5
  + $MijnVariabele.GetType()
* De variabele werkt alleen met double quotes en niet met single quotes de single quote pakt letterlijk over wat er staat
* Int32, double,string
* BasicArray = "John", "Susie", "Jim", "Johnny", "Carrie"
* $BasicArray[0] # John
* $BasicArray[-1] # Carrie
* $BasicArray[0..3] # eerste 4 elementen
* $GetallenArray = 1..10
* $MijnArray.Count()
* $MijnArray += "Test"
* $NieuweBasicArray = $BasicArray -ne "Jim" # elementen verwijderen: nieuwe Array maken
* [Int32]$Getal
* [Float]$Getal
* [string]$Getal
* Voorbeeld : [string]getal = “2”
  + [string]getal + 2 => 22 als output
* Ingebouwde variabelen
  + **$^** Eerste token van de vorige command line
  + **$$** Laatste token van de vorige command line
  + **$?** Boolean status van het laatste **commando**
  + **$\_** of **$PSItem** Huidig pipeline object
  + **$Args** Argumenten van een script of functie
  + **$Error** Array van errors van vorige commando's
  + **$HOME** The user’s home directory
  + **$LastExitCode** Exit code van laatste **programma of script**
* Ingebouwde variabelen :
  + **$PSHOME** installatie locatie van Windows PowerShell
  + **$PROFILE**  PowerShell profile locatie
  + **$PSCulture**  Huidige culture  
     (keyboard layout, getal en datum formaten)
  + **$PSUICulture** Huidige UI culture
  + **$PsVersionTable** PowerShell versie
  + **$PWD** Huidge directory
* Environment variabele : Get-ChildItem env:
  + $env:USERNAME
  + $env:HOMEDRIVE
  + $env:HOMEPATH
  + $env:COMPUTERNAME
  + $env:APPDATA
  + $env:USERDOMAIN
* Read-Host -Prompt
  + $ServerNaam = Read-Host -Prompt "Server name"
  + $GebruikerNaam = Read-Host -Prompt "User name"
  + $Datum = Get-Date
  + Write-Output "Server $ServerNaam, user $GebruikerNaam op $Datum."
* Paswoorden met -AsSecureString
  + $VeiligPaswoord = Read-Host "Geef een paswoord" -AsSecureString

Hoofstuk 8 : commandline parameter

Unamed parameters

* **param1=$args[0]**

**$param2=$args[1]**

**$param3=$args[2]**

Write-Host $param1 $param2 $param3

\Unnamed\_Arguments\_Example\_1.ps1 een twee drie

* write-Host "There are a total of $($args.count) arguments"

for ( $i = 0; $i -lt $args.count; $i++ ) {

Write-Host "Argument $i is $($args[$i])"

}

Named parameters:

* param (

[ParameterAttribute()]

[DataType] $ParameterName1,

[DataType] $ParameterName2 = DefaultValue,

...

)

* param (

**[DataType] $ParameterName = DefaultValue,**

**...**

)

* paramters types : int,double,bool,datetime,array,switch
* Param (

[string] $firstname,

[int] $age

)

Write-Output "Firstname: $firstname"

Write-Output "Age: $age"

Write-Output $firstname.GetType()

Write-Output $age.GetType()

* Param (

**[string[]] $Numbers**

)

Write-Output "Total numbers are: $($Numbers.Count)"

foreach ($item in $Numbers) {

Write-Host $item

}

Write-Output $Numbers.GetType()

* **Switch als je de parameter gebruikt dat word het beshouwd als true als het niet word gebruikt dan word het beshouwd als false bij het runnne van de script**
* Param(

[string] $Firstname,

**[switch] $Capital**

)

if ($Capital){

$Firstname = $Firstname.ToUpper()

}

Write-Output "Firstname: $Firstname"

* param (

**[Parameter(Mandatory)]**

**[DataType] $MandatoryParameter**

)

* param(

**[ValidateSet("Cow","Dog","Cat",ErrorMessage="Animal {0} is not among list")]**

[String]$Animal = "Dog"

)

Write-Output "$Animal is Pet Animal"

* param(

[ValidatePattern("^\d{3}-\d{2}-\d{4}$",ErrorMessage="{0} is not a correct

number")]

[string] $SocialSecurityNumber

)

Write-Output "$SocialSecurityNumber"

.\DeclareParameterValidatePattern.ps1 123-12-1234

**#de d3 betekent 3decimalen getallen**

* **param (**

**[ValidateScript({**

**$\_ -gt 0 -and $\_ -lt 100**

**})]**

**[int] $Number**

**)**

**Write-Output "$Number"**

* **param (**

**[ValidateRange(1, 100)]**

**[int] $Age**

**)**

**Write-Output "$Age"**

**Hoofdstuk 9 : conditionals**

* If statement
  + **if (<test>) {**
  + **<statements als uitkomst test $true is>**
  + **}**
* if else statement
  + **if (<test>) {**
  + **< statements als uitkomst test $true is >**
  + **} else {**
  + **< statements als uitkomst test $false is >**
  + **}**
* Nesting conditional statements
  + **if (<test>) {**
  + **<conditie1 is true>**
  + **}**
  + **else{**
  + **if (<test>) {**
  + **<conditie2 is true>**
  + **} else {**
  + **<conditie3 is true>**
  + **}**
  + **}**
* **Elseif statement**
  + **if (<test>) {**
  + **<conditie1 is true>**
  + **} elseif (<test>) {**
  + **<conditie2 is true>**
  + **} else {**
  + **<conditie3 is true>**
  + **}**
* Switches : De Switch-constructie is vergelijkbaar met een If-constructie met meerdere ElseIf-secties
  + **Clear-Host**
  + **$number = Read-Host "Vul een getal in tussen 0 en 4"**
  + **Switch ($number){**
  + **1**
  + **{Write-Host "Je hebt het getal 1 gekozen"}**
  + **2**
  + **{Write-Host "Je hebt het getal 2 gekozen"}**
  + **3**
  + **{Write-Host "Je hebt het getal 3 gekozen"}**
  + **Default**
  + **{Write-Host "Je hebt niet het getal 1,2 of 3 gekozen"}**
  + **}**
* **Om uit een switch te komen zonder alle andere waarden te evalue- ren kun je break gebruiken.** 
  + **Voorbeeld:**
  + **{Write-Host "Je hebt het getal 1 gekozen" ; break}**
* **$ip = "10.2.5.6"**

**switch -Regex ($ip){**

**“^10.\*" {Write-Host "This computer is on the internal network"}**

**“^10.1.\*" {Write-Host "This computer is in London"}**

**“^10.2.\*" {Write-Host "This computer is on Vancouver"}**

**Default {Write-Host "This computer is not on the internal network"}**

**}**

**Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving**

* -not of !
  + if ( -not ( Test-Path -Path $meegegevenPad ) ) {}
  + if ( -not $waarde ) {}
  + if ( !$waarde ) {}
* -and
  + if ( ($leeftijd -gt 18) -and ($leeftijd -lt 67) ) {}
  + if ( ($null -ne $meegegevenPad) -and (Test-Path -Path $meegegevenPad) ) {}
* -or
  + if ( ($leeftijd -le 18) -or ($leeftijd -ge 67) ) {}
* **Test-Path -Path "C:\Users" => kijkt of de pad bestaat of niet**
* **$null is een automatische variabele in PowerShell heeft als inhoud NULL.**
  + Checken of een variabele leeg is met $null
  + if ( $null -eq $Aantal ) {  
    }

**hoofdstuk 10 : loops**

* $computerNames = 'WinServer01', 'WinClient-01', 'WinClient-02','WinClient-03'

foreach ($computer in $computerNames) {  
    Get-ADComputer -Identity $Computer  
}

* [array]$numbers = @(1,2,3,4,5,6,7,8,9,10)

Foreach ($number in $numbers){

Write-Host $number

}

* $\_
* $ComputerNamen = 'WinServer01', 'WinClient-01', 'WinClient-02','WinClient-03'

**$Teller = 1**

foreach ($ComputerNaam in $ComputerNamen) {

Write-Output "Computernaam $Teller`: $ComputerNaam"

**$Teller = $Teller + 1**

}

* Foreach(scriptblock expression)
  + $services = get-service \*
  + $services.foreach(‘Name’)
* Met de foreach script block kun je type van het **object coverteren naar bv een string :**
  + $process = get-process
  + $processtostring = $process.foreach([string])
  + Als je de eerste object type wilt zien dan doe je :
    - Processtostring[0].gettype()
* Launch notepad 10x en sluit ze:
  + 1..10 | ForEach-Object {invoke—item c:\windows\notepad.exe}
  + $process = get-process -name “notepad”
  + $process.foreach(“Kill”)
* Where script block
  + $services = get-service \*
  + $services.where($\_.status -eq “running”)
  + Geef de eerste 10 serivces die aan het runnen zijn :
    - $services.where({$\_.status -eq “running”},’Default’,10)



* Parallel : In PowerShell 7 is de parameter -Parallel toegevoegd aan de Cmdlet ForEach-Object . Hierdoor kan de pijplijn meerdere objecten tegelijk verwerken. Het gelijktijdig verwerken van meerdere objecten kan betere prestaties
* $users | foreach-object -parallel {set-aduser $user -department “marketing”}
* For loops:
  + for ($i = 1; $i -lt 5; $i++) {
  + Write-Output "Sleeping for $i seconds"
  + Start-Sleep -Seconds $i
  + }
* Import csv
  + $Users = Import-Csv -Path .\users\_small.csv
  + Import-Csv .\lijstje.csv -Delimiter ';'
  + $users = Import-Csv -path .\users\_small.csv -Delimiter ";"

foreach ($user in $users){  
    $voornaam = $user.GivenName

    $achternaam = $user.Surname

    Write-Host ("{0} heeft als achternaam {1}" -f $voornaam, $achternaam)

}

* Import excel
  + Install-module -name ImportExcel -scope CurrentUser
  + $users = Import-Excel -Path .\names.xlsx -WorksheetName 'Blad1'
  + $users | Get-Member
  + $users.GivenName
* Xml
  + $users = [xml](Get-Content ".\personen.xml")  
    $users = $users.SelectNodes("//people//person")

foreach ($user in $users){

  $id = $user.id

    $naam = $user.name

$age = $user.age

Write-Host ("id:{0}`nnaam:{1}`nage:{2}`n" -f $id,$naam,$age)

}

* Do until loops
  + do {

        $notepad = Get-Process Notepad -ErrorAction SilentlyContinue

        Start-Process Notepad.exe

        Write-Host "Starting Notepad, current count is: $($notepad.Count + 1)"

      $notepad = Get-Process Notepad

 } until ($notepad.Count -eq 5)

* do while loop :

$i = 0

do {

    $i

   $i += 5

} while ($i -le 100)

* while loops
  + $i = 0

while ($i -le 100) {

    $i

    $i += 5

}

**Hoofdstuk 11 : exception handling**

* $ErrorActionPreference : vertelt wat er moet gebeuren met de error
  + Er zijn 4 moglijkheden in de variabele :
  + **Continue => is de standaard waard en laat de error zien op de scherm**
  + SilentlyContinue => negeert de error laat het niet op scherm zien
  + Inquire => er word een prompt getoond en gevraagd aan de gerbuiker wat er met error word gedaan
  + **Stop => als er een error is dan stopt het de met runnen van de commando dat je hebt ingegeven**
* **$ErrorActionPreference = 'Inquire'**
* Get-process -name test -ErrorAction stop
* Try {

  1/0

}

Catch {

  Write-Output "Berekening gefaald."

}

* Als de try faald dan word de catch getoond
* Try { DIT WERKT NIET

  Get-ChildItem "c:\ik\besta\niet\" -ErrorAction SilentlyContinue

}

Catch {

Write-Output "Directory bestaat niet"

}

Dit gaat niet werken om dat het geen stopping error is BELANGRIJK

* Try {

  Get-ChildItem "c:\ik\besta\niet\" **-ErrorAction STOP**

}

Catch {

  Write-Output "Directory bestaat niet"

}

* $dirs = “c:\windows”,$env:APPDATA,”c:\ik\besta\niet”
* Foreach($dir in $dirs){

Try{get-childitem $dir -erroraction stop| out-null}

Catch{write-output “$dir niet gevonden”}

* **try** {
* <probeer statements>
* } **catch** [error type] {
* <statements als error>
* } **finally** {
* <statements om altijd uit te voeren op het einede van de opdracht>
* }

Afbeelding met tekst, schermopname, Lettertype, Website

Automatisch gegenereerde beschrijving

* Afbeelding met tekst, schermopname, Lettertype, Website

  Automatisch gegenereerde beschrijving
* Afbeelding met tekst, schermopname, Lettertype

  Automatisch gegenereerde beschrijving
* De meeste recente error die gebeurd na de try word getoond op descherm dit doe je door de $errr[0].exception.message als je alleen de errror bercht wilt zien . gebruik de get-error om de error type te zien
* Param (
* [string][Parameter(Mandatory)]$berekening

)

try{

    Invoke-Expression $berekening

}  
  catch [DivideByZeroException]{

    Write-Host “Divide by zero exception”

} catch [System.Net.WebException],[System.Exception]{

  Write-Host “Other exception”

} finally{

  Write-Host “cleaning up ...”

}

Hoofdstuk 12 : functions

* function Start-App($AppName){

  $AppID = Get-StartApps | Where-Object Name -like $AppName

    Start-Process "explorer.exe" -ArgumentList ("shell:AppsFolder\" + $AppID.AppID)

 }

* **default waarde geven aan de argument zodat je de functie kunt oproepen zonder argument**
* function Start-App([string]$AppName = "notepad"){

    $AppID = Get-StartApps | Where-Object Name -like $AppName

    Start-Process "explorer.exe" -ArgumentList ("shell:AppsFolder\" + $AppID.AppID)

}

Start-App

Start-App notepad++

* function Start-App{

   Param(

        [parameter(Mandatory=$true)][String]$AppName

    )

    $AppID = Get-StartApps | Where-Object Name -like $AppName

    Start-Process "explorer.exe" -ArgumentList ("shell:AppsFolder\" + $AppID.AppID)

}

* function WriteUser

{

    param(

        $user = "Default",

        $message = "Message"

   )

    Write-Host "$user`: $message"

}

WriteUser

WriteUser Tom test123

WriteUser Dominique "Hello everybody"

* Afbeelding met tekst, schermopname, Lettertype

  Automatisch gegenereerde beschrijving
* Afbeelding met tekst, schermopname, Lettertype, software

  Automatisch gegenereerde beschrijving
* Afbeelding met tekst, schermopname, scherm, software

  Automatisch gegenereerde beschrijving
* Afbeelding met tekst, schermopname, Lettertype

  Automatisch gegenereerde beschrijving
* De start-apps en get-start apps zijn beide functies en die kun je pipen met elkaar dus de $apps komt van de functie get-startapps die je ook hebt aangemaakt

Hoofdstuk 13 modules :

* Modules zijn groepen gerelateerde PowerShell-mogelijkheden die in één eenheid zijn gebundeld.
* Als u de cmdlets van een module wilt gebruiken, moet de module worden geladen in de huidige PowerShell-sessie
* Import-Module
* Normaal laad de module automatisch als je de cmdlet gebruikt werkt alleen voor powershel 3.0 en hoger. Maar dit gaat alleen als de module in de mappen onder de laadpaden van de module bevindt : %systemdir%\WindowsPowerShell\**[versie]**\Modules en %userprofiles%\Documents\WindowsPowerShell\Modules.
* $env:PSModulePath
* **Get-Module**
* Vb get-module netadapter
* Get-Command –Module NetAdapter
* Install-module
* Install-module **powershellget**

Automation :

Hoofdstuk : local users

[Afbeelding met tekst, schermopname, Lettertype, nummer

Automatisch gegenereerde beschrijving](https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.localaccounts/?view=powershell-5.1&viewFallbackFrom=powershell-7.2)

* Get-LocalUser
* Get-LocalUser -Name student | Select-object \*
* Get-LocalGroup
* Get-LocalGroupMember Administrators
* $Password = Read-Host -AsSecureString
* New-LocalUser "USER\_ACCOUNT\_NAME"   
   -Password $password   
   -FullName "USER\_FULL\_NAME"   
   -Description "DESCRIPTION"
* Add-LocalGroupMember -Group “Users" (Gebruikers)  
  -Member “USER\_ACCOUNT\_NAME"
* Add-LocalGroupMember -Group "Administrators"   
  -Member “USER\_ACCOUNT\_NAME
* $Password = Read-Host "Enter the new password" –AsSecureString
* $UserAccount = Get-LocalUser -Name “student“
* $UserAccount | Set-LocalUser -Password $Password => dus je zoekt de object student en als je die hetb gevonden dan neem je die object over de pipeline naar de naar de volgende commando
* Get-LocalUser student | Disable-LocalUser of disable-localuser student
* Remove-LocalUser -Name "USER\_ACCOUNT\_NAME"

Hoofdstuk : windows information

* Windows Information – Get-ComputerInfo
* Windows Information – CIM
* Examples
  + Processor / Baseboard / Bios Information
  + Windows System Information
  + Windows Process Information
  + Windows Disk Information
  + Account Information
  + Physical Memory / Network / Graphics Information
  + Software Installed
* get-ciminstance -classname => Selecteer een CIM-klasse: Je begint door een CIM-klasse te specificeren. Een CIM-klasse vertegenwoordigt een bepaald type object, zoals een proces (Win32\_Process), service (Win32\_Service), of hardwarecomponent (Win32\_LogicalDisk).
* Get-CimClass:
  + Uitleg: Geeft de definitie van een CIM-klasse weer, inclusief de beschikbare eigenschappen en methoden.
* Get-CimSession:
  + Uitleg: Maakt een CIM-sessie-object aan, waarmee je verbinding kunt maken met andere computers om CIM-operaties uit te voeren.
  + Dus als je computerinformatie wilt krijgen van een andere computer dan gebruik je cim session
* Cim querie => om cim commandos uit te voeren
  + $specificProcess = Get-CimInstance -Query "SELECT \* FROM Win32\_Process WHERE Name = 'explorer.exe'"
* Get-CimAssociatedInstance:
  + Uitleg: Haalt instanties op die zijn geassocieerd met een specifieke CIM-instantie, bijvoorbeeld, haal alle processen op die zijn gestart door een bepaalde service
  + $associatedProcesses = Get-CimAssociatedInstance -InputObject $serviceInstance -ResultClassName Win32\_Process
* Invoke-CimMethod:
  + Uitleg: Roept een methode aan op een CIM-instantie, bijvoorbeeld, start een proces.
  + $process = Get-CimInstance -ClassName Win32\_Process -Filter "Name='notepad.exe'"
  + Invoke-CimMethod -InputObject $process -MethodName Terminate
* WMI (Windows Management Instrumentation) en Common Information Model (CIM) zijn gerelateerde technologieën. Beide technologieën zijn gebaseerd op industriestandaarden die de Distributed Management Task Force (DMTF) definieert
* CIM is een nieuwere technologie die is gebaseerd op open, platformoverschrijdende standaarden.
* CIM-cmdlets kunnen op drie manieren verbindingen maken:
* Lokale verbindingen.
* Wanneer u geen externe computer opgeeft, gebruiken de CIM-cmdlets een lokale COM-sessie om toegang te krijgen tot en te communiceren met de opslagplaats.
* Ad-hocverbindingen.
* Wanneer u een externe computer opgeeft met behulp van de parameter - ComputerName met een CIM-cmdlet, gebruikt de externe verbinding WS-MAN.
* CIM-sessies.
* U kunt een CIM-sessie vooraf maken met specifieke opties voor connectiviteit met externe computers. Wanneer u een CIM-sessie maakt, kunt u opgeven of U WS-MAN of DCOM wilt gebruiken.
* Als u de CIM-cmdlets op **een externe computer wilt gebruiken, hoeft powerShell niet op de externe computer te zijn geïnstalleerd**
* De externe computer moet WS-MAN ondersteunen om er verbinding mee te maken met CIM-cmdlets.
* U kunt cim cmdlet gerbuiken om linux computers
* Over het algemeen moet u CIM-cmdlets gebruiken in plaats van de oudere WMI-cmdlets.
* De cim-cmdlet gebruiken ws-Man voor externe connectiviteit
* **Als externe computers geen ondersteuning bieden voor het gebruik van WS-MAN**, kunt u het **gebruik van DCOM** opgeven voor connectiviteit met behulp van **een CIM-sessie.**
* **De opslogplaats zijn niet goed gedocumenteerd** waardoorr het detectern van klassen moelijk is daar als eer **een powershell cmdlet bestaan dan zou beter** zijn om die cmdlet te gebruiken in de plaats van de klassenmethodes van cim
* Get-ciminstance -Namespace
  + In een computersysteem zijn verschillende soorten informatie georganiseerd in verschillende "namespaces**" of ruimtes**. Een namespace kan worden gezien als een **specifiek gebied waarin bepaalde soorten gegevens worden georganiseerd**.
  + Get-CimInstance -ClassName Win32\_Process -Namespace "root/CIMv2"
* Om alle ruimtes(namespace) te zien van de cimcmdlet waar all uw computerinfo staat
* Afbeelding met tekst, schermopname, software

  Automatisch gegenereerde beschrijving

Afbeelding met tekst, software, Multimediasoftware, schermopname

Automatisch gegenereerde beschrijving

* Op een externe computer maak je een notepad aan en vervolgens delete je de notepad
* Invoke-CimMethod -ComputerName desktop-02 -ClassName Win32\_Process -MethodName Create -Arguments @{CommandLine='Notepad.exe'}
* Get-CimInstance -Class Win32\_Processor
* Get-CimInstance -ClassName Win32\_Baseboard => geeft info over de moederbord
* Get-CimInstance -ClassName Win32\_BIOS

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

* Get-CimInstance -Namespace root\cimv2 -ClassName Win32\_OperatingSystem
* Get-CimInstance -Namespace root\cimv2 -ClassName Win32\_OperatingSystem | Get-Member
* Os software installed :
  + Get-CimInstance -Class Win32\_InstalledWin32Program | Select-Object Name
* Windows process information
  + Get-CimInstance -ClassName Win32\_Process -Filter "Nam
  + Get-CimInstance -Query "SELECT \* from Win32\_Process WHERE name LIKE 'P%’”
  + Get-CimInstance -ClassName Win32\_Product -Filter 'Name like "Microsoft%“’
  + Get-CimInstance -ClassName Win32\_Service -filter "name='McpManagementService'" | select \*
  + Get-CimInstance -ClassName Win32\_Service -filter "name='McpManagementService’”
  + Get-CimInstance -ClassName Win32\_Service -filter 'name like "McpManagementService“’
  + Get-CimInstance -ClassName Win32\_Service -filter "name like 'McpManagementService'"
  + Get-CimInstance -ClassName Win32\_Service -filter 'name="McpManagementService“’
* “p%” = naam begint met p
* Windows disk information
  + Get-CimInstance -ClassName CIM\_LogicalDisk | Format-Table DeviceID, DriveType, ProviderName, Size, FreeSpace
  + Get-CimInstance -ClassName CIM\_LogicalDisk | Where-Object {$\_.DriveType -eq 3} | Format-Table DeviceID, DriveType,Size, FreeSpace
* Account information
  + Get-CimInstance -Class Win32\_UserAccount

Afbeelding met tekst, schermopname, Lettertype, document

Automatisch gegenereerde beschrijving

* Physical Memory / Network / Graphics Information
  + Pyhiscal ; Write-Warning 'Get-CimInstance -ClassName Win32\_PhysicalMemory'
  + Get-CimInstance -ClassName Win32\_PhysicalMemory
  + Network adamper : Get-CimInstance -ClassName Win32\_NetworkAdapter
  + Info installed grahpics : Get-CimInstance -ClassName Win32\_VideoController

Hoofdstuk : windows management

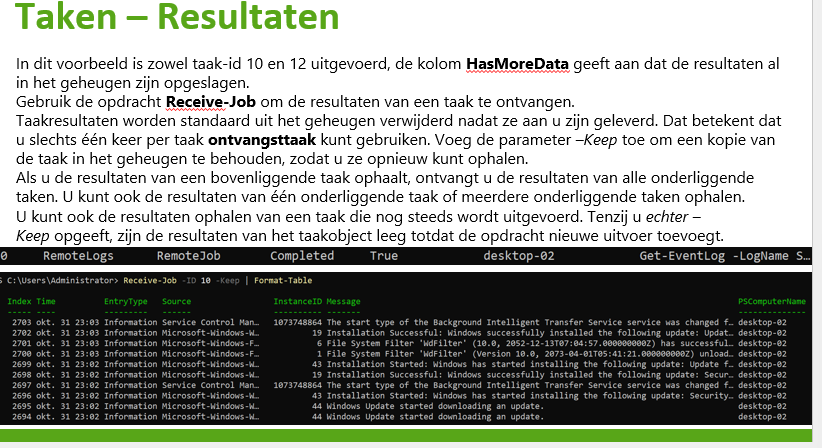
* de **module Microsoft.PowerShell.Management** bevat veel ingebouwde cmdlets die kunnen worden gebruikt om informatie te verkrijgen en specifieke bewerkingen uit te voeren op een lokale computer
* Get-command -module Microsoft.PowerShell.Management
* Get-Module Microsoft.PowerShell.Management | select -ExpandProperty ExportedCommands

Afbeelding met tekst, schermopname, Lettertype, ontwerp

Automatisch gegenereerde beschrijving

* Beheers cmdlet :
  + Get-computerinfo
  + Get-winevent -logname application -maxevent 20
* Machtigingen beheren
  + Get-command -module Microsoft.PowerShell.Security
  + Get-acl
  + Set-acl
* Get-Acl -Path C:\windows |Format-List
* (Get-Acl -Path C:\windows).Access | ft
* Afbeelding met tekst, schermopname, Lettertype, Website

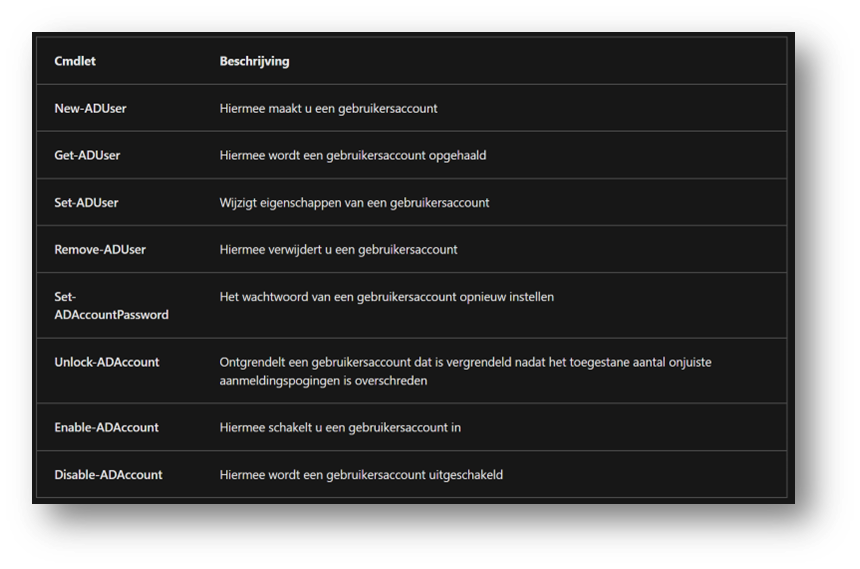
  Automatisch gegenereerde beschrijving
* Afbeelding met tekst, schermopname, Lettertype

  Automatisch gegenereerde beschrijving
* $ACL = Get-Acl -Path C:\powershell\
* $ACL = Get-Acl -Path C:\powershell\
* $AccessRule = New-Object System.Security.AccessControl.FileSystemAccessRule("Ine","Modify","Allow")
* $ACL.RemoveAccessRule($AccessRule)
* 
* Locale taken : Start-Job -ScriptBlock { Dir C:\ -Recurse } -Name LocalDirectory
* Locale taken : Start-Job -FilePath C:\test.ps1 -Name TestScript
* Externe taken : Invoke-Command -ScriptBlock { Get-EventLog -LogName System -Newest 10 } -ComputerName desktop-02 -AsJob -JobName RemoteLogs
* -asjob => laat het worden uitgevoerd op in het achetgrond
* -jobname geef de taak een naam
* Taken ophalen : get-job
* Recieve-job
* 

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

Hoofdstuk : active directory

* Active Directory Domain Services (AD DS) en de bijbehorende services vormen de kern van Windows Server-netwerken.
* De AD DS-**database slaat informatie op over de objecten die deel uitmaken van de netwerkomgeving,** zoals accounts voor gebruikers, computers en groepen.
* Install-WindowsFeature AD-Domain-Services –IncludeManagementTools
* Cmdlets voor gebruikersaccounts hebben het **woord 'Gebruiker' of 'Account'** in het zelfstandig naamwoordgedeelte van de naam. Als u de beschikbare cmdlets wilt identificeren, neemt u deze op in zoekopdrachten met jokertekens wanneer u Get-Help of Get-Command gebruikt.
* Get-Command -Module ActiveDirectory
* 
* Sam acount => security account manager
* Get-ADUser -Identity Administrator -Properties \*
* Get-ADUser -Filter {Name -eq "Administrator"} -Properties logonCount
* Get-ADUser -Identity Administrator -Properties logonCount
* Controleren of een AD-user bestaat:
  + $Name = "administrator"

$User = Get-ADUser -Filter {SAMAccountName -eq $Name}

If ($User -eq $Null)

{write-host "User $name does not exist in AD"}

Else

{write-host "User $User found in AD"}

* Wanneer u de cmdlet New-ADUser gebruikt om nieuwe gebruikersaccounts te maken, is de **parameter -Name vereist.**
* **Je moet bij een wachtwoord parameter wnr je user aanmaakt een variable als waarde geven omdat je de wachtwoord secured moet zijn. Of anders vraag je het op in console om de wachtwoord in te geven wnr je een user aanmaakt**
* Als u een wachtwoord instelt, kunt u het gebruikersaccount inschakelen door de parameter -**Enabled in te stellen als $true**
* Als je een gebruiker geen wachtwoord toevoegd dan is het de gebuiker uitgeshakeld
* Set-ADAccountPassword -Identity 'dominique houbrechts' -Reset -NewPassword (ConvertTo-SecureString -AsPlainText "pxlserver\_007" -Force)
* set-ADUser "Dominique Houbrechts" -Enabled $true
* New-ADUser -Name "Dominque Houbrechts" -GivenName "Dominique" -Surname "Houbrechts" -SamAccountName "dhoubrechts" -UserPrincipalName "dominique.houbrechts@pxl.be" -AccountPassword(Read-Host -AsSecureString "Input Password") -Enabled $true
* User principal name => is de user logon name
* Groepen beheren:
  + get-command \*-ADGroupMember
  + get-command \*-ADPrincipalGroupMembership
  + New-ADGroup -Name Digital -GroupScope Global
  + New-ADGroup -Name Business -GroupScope Global
  + Get-ADGroup
  + Get-ADGroup -Filter \* | Where-Object {$\_.Name -like "Di\*"}
* get-command \*-ADGroupMember
* Group members toevoegen
  + Add-ADGroupMember -Identity "Digital" -Members dhoubrechts
  + Remove-ADGroupMember -Identity "Digital" -Members dhoubrechts
  + Get-ADGroupMember -Identity "Digital“

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

* OU’s
  + New-ADOrganizationalUnit "Digital" -Path "dc=pxl,dc=local" -ProtectedFromAccidentalDeletion $false
  + New-ADOrganizationalUnit "SNE" -Path "ou=digital,dc=pxl,dc=local" -ProtectedFromAccidentalDeletion $false
  + Get-ADOrganizationalUnit -Filter {Name -like "Digi\*"}
  + remove-ADOrganizationalUnit "OU=SNE,OU=Digital,DC=PXL,DC=local" -Confirm:$false
  + remove-ADOrganizationalUnit "OU=Digital,DC=PXL,DC=local" -Confirm:$false
* DISM.exe /Online /Get-CapabilityInfo /CapabilityName:Rsat.ActiveDirectory.DS-LDS.Tools~~~~0.0.1.0 => RSAT dit is om te kijken of het geinstalleerd is
* DISM /Online /Add-Capability /CapabilityName:Rsat.ActiveDirectory.DS-LDS.Tools~~~~0.0.1.0

Dit is om te instaleren

* Add client local computer to domain
  + Add-Computer -ComputerName Desktop-02 -LocalCredential Desktop-02\student -DomainName pxl.local -Credential pxl\administrator -restart

Hoofdstuk : remote access

* The Enable-PSRemoting cmdlet configures the computer to receive PowerShell remote commands that are sent by using the WS-Management technology.
* You have to run this command only one time on each computer that will receive commands. You do not have to run it on computers that only send commands.
* Enable-PSRemoting -force SkipNetworkProfileCheck
* Get-PSSessionConfiguration
* Disable Ps remote
  + Stap 1 : # Run as admin
  + Stap 2 : enable-psremoting
  + Stap3 : Get-PSSessionConfiguration | Format-Table -Property Name, Permission
  + Stap4 : Disable-PSRemoting -Force
  + Stop-Service WinRM -PassThru
  + Set-Service WinRM -StartupType Disabled -PassThru
  + Get-Service WinRM | Select-Object Name, Status, StartupType
  + #dir wsman:\localhost\listener
  + #Once you know the name of the listener, you can remove it with the next PowerShell command.
  + #Remove-Item -Path WSMan:\Localhost\listener\<Listener name>
  + #Remove all listeners
  + #Remove-Item -Path WSMan:\Localhost\listener\listener\* -Recurse
  + #Disable the firewall exceptions
  + Set-NetFirewallRule -DisplayName 'Windows Remote Management (HTTP-In)' -Enabled False -PassThru
  + | Select-Object -Property DisplayName, Profile, Enabled
* Viewu windows computers in active directory
  + Afbeelding met tekst, schermopname, software, Multimediasoftware

    Automatisch gegenereerde beschrijving
* New-PSSession -ComputerName winserver01,desktop-02
* Get-PSSession
* New-PSSession -ComputerName (Get-ADComputer -Filter \* | Select-Object -ExpandProperty Name)
* #Creating new sessions from Text file
  + $servers = Get-content -Path .\servers.txt
  + $sessions = New-PSSession -ComputerName $servers
  + or
  + $sessions = New-PSSession -ComputerName (Get-content -Path .\Servers.txt)
* In de servers.txt zitten de namen van de computers van de ad dit kun je zelf maken via notepad
* Remove session
  + Get-PSSession | Remove-PSSession
  + Get-PSSession

Afbeelding met tekst, schermopname, software, Website

Automatisch gegenereerde beschrijving

* $remoteSessions = Get-PSSession
* Invoke-Command -Command { Get-CimInstance -Class Win32\_OperatingSystem } -Session $remoteSessions | ft
* Afbeelding met tekst, schermopname, Lettertype, software

  Automatisch gegenereerde beschrijving
* De doel om de lijst van processoren van de computers die in lijst zitten
* Afbeelding met tekst, schermopname, Lettertype, Website

  Automatisch gegenereerde beschrijving
* **CIM Sessie-objecten maken**
  + $s = New-CimSession -ComputerName WINSERVER01
  + $sessions = New-CimSession -ComputerName WINSERVER01,DESKTOP-02
* Wanneer u een sessie maakt, **wordt de verbinding niet onmiddellijk tot stand gebracht in** PowerShell. Wanneer een cmdlet de CIM-sessie gebruikt, maakt PowerShell verbinding met de opgegeven computer en beëindigt PowerShell de verbinding wanneer de cmdlet is voltooid..
* Dus een cimsessie houd de verbinding niet open het sluit direct wnr uw cmdlet voltooid is

Het is beter om een ander vorm van externe bereik te gebuiken op powershell om de verbing tot stand te houden

* Afbeelding met tekst, schermopname, Lettertype

  Automatisch gegenereerde beschrijving
* C
* Cim session met dcom in de plaats van Ws-Man
  + $opt = New-CimSessionOption -Protocol Dcom
  + $DcomSession = New-CimSession -ComputerName winserver01 -SessionOption $opt
  + Get-CimInstance -ClassName Win32\_BIOS -CimSession $DcomSession
* Cim session verwijderen
  + $sessions | Remove-CimSession
  + Get-CimSession -ComputerName winserver01 | Remove-CimSession
  + Get-CimSession | Remove-CimSession

Hoofdstuk : netwerken

* met PowerShell zijn ONDER andere TCP/IP, Domain Name System (DNS), firewall en routeringstabelconfiguraties.
* Ping en tracert
* PowerShell bevat de NETTCPIP-module => tcp/ip
* Get-Command -Module NetTCPIP
* Get-NetIPAddress
* New-NetIPAddress -IPAddress 192.168.1.10 -InterfaceAlias "Ethernet0" -PrefixLength 24 -DefaultGateway 192.168.1.1i
* Afbeelding met tekst, schermopname, Lettertype

  Automatisch gegenereerde beschrijving
* New-netipaddress parmaeters
  + Afbeelding met tekst, schermopname, Lettertype

    Automatisch gegenereerde beschrijving
* Get-NetIPAddress -InterfaceAlias "Ethernet0" -AddressFamily "IPv4“
* Get-netroute
* Get-DnsClient -InterfaceAlias Ethernet0 | select \*
* Afbeelding met tekst, schermopname, Lettertype, software

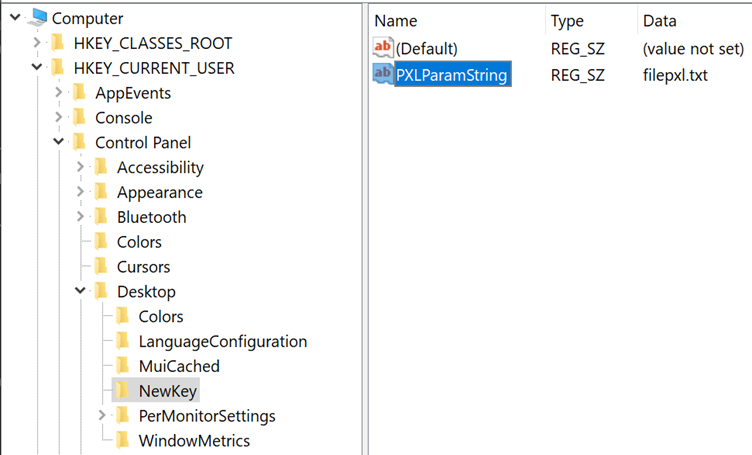
  Automatisch gegenereerde beschrijving
* De module voor de bovenste foto is de netsecurity.module => firewall en ip beveiliging

Hoofdstuk : registry

* Registry keys
* Working with the registry in PowerShell is similar to working with common files on a local disk.
* Display the list of available drives on your computer:  Get-PSDrive
* Special devices available through the **Registry provider** – HKCU (HKEY\_CURRENT\_USER) and   
  HKLM (HKEY\_LOCAL\_MACHINE).
* Registry referenties
  + **Get-Item**   
     Get a registry key
  + **New-Item**
  + Create a new registry key
  + **Remove-Item**
  + Delete a registry key
* Get-ItemProperty  get the value of a registry parameter
* Set-ItemProperty  change the value of a registry parameter
* New-ItemProperty  create registry parameter
* Rename-ItemProperty rename parameter
* Remove-ItemProperty  remove registry parameter

Afbeelding met tekst, schermopname, software, Website

Automatisch gegenereerde beschrijving

* Get-Item -Path Registry::HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion
* Get-Item -Path Registry::HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion | Select-Object -ExpandProperty property
* Get-ItemProperty -Path Registry::HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion
* Set-Location -Path hklm:\SOFTWARE\Microsoft\Windows\CurrentVersion
  + Get-childitem
* Changing registry value
  + Get-ItemProperty -Path Registry::HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\ -Name Hidden
  + Set-ItemProperty -Path Registry::HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\ -Name Hidden -Value 0
  + Get-ItemProperty -Path Registry::HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\ -Name Hidden
* Create Registry Key or Parameter – New-Item / Property
  + **$HKCU\_Desktop= "HKCU:\Control Panel\Desktop“**
  + **New-Item –Path $HKCU\_Desktop –Name NewKey**
* De parameret is uw property create parameter
  + **New-ItemProperty -Path $HKCU\_Desktop\NewKey -Name "PXLParamString" -Value ”filepxl.txt” -PropertyType "String"**
* Resultaat
* 
* Check if a registry key exists – Test-Path
  + $regkey='HKCU:\Control Panel\Desktop\NewKey'
  + $regparam='PXLParamString'
  + if (Get-ItemProperty -Path $regkey -Name $regparam -ErrorAction Ignore){
  + write-host 'The registry entry already exist' }
  + else{
  + New-ItemProperty -Path $regkey -Name $regparam -Value ”woshub\_test”  -PropertyType "String"
  + }
* Delete a registry key and property
  + $HKCU\_Desktop= "HKCU:\Control Panel\Desktop"
  + Remove-ItemProperty -Path $HKCU\_Desktop\NewKey -Name "PXLParamString"
  + Remove-Item -Path $HKCU\_Desktop\NewKey -Recurse
* Rename a registry key
  + Rename-Item -path 'HKCU:\Control Panel\Desktop\NewKey' OldKey
  + Rename-ItemProperty -path "HKCU:\Control Panel\Desktop\NewKey" -name "PXLParamString" -newname “oldPXLParamString
* Find a registry key / Parameter
  + Find the registry key

Get-ChildItem -path HKCU:\ -recurse -ErrorAction SilentlyContinue | Where-Object {$\_.Name -like "\*oldkey\*"}

* + Finde de registry parameter

$Path = (Get-ItemProperty "HKCU:\Control Panel\Desktop\OldKey")

$Path.PSObject.Properties | ForEach-Object {

    If($\_.Name -like '\*pxl\*'){

        Write-Host $\_.Name ' = ' $\_.Value

    }

}

* Afbeelding met tekst, schermopname, Lettertype

  Automatisch gegenereerde beschrijving
* {
* Push-Location

  Set-Location HKCU:\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced

  Set-ItemProperty . HideFileExt "1"

* Pop-Location
* Stop-Process -processName: Explorer -force # This will restart the Explorer service to make this work.
* }
* function ShowFileExtensions()
* {
* Push-Location
* Set-Location HKCU:\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced
* Set-ItemProperty . HideFileExt "0"
* Pop-Location
* Stop-Process -processName: Explorer -force # This will restart the Explorer service to make this work.
* }
* #Show file extensions
* ShowFileExtensions
* Afbeelding met tekst, schermopname, software, Lettertype

  Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, Lettertype, Website

Automatisch gegenereerde beschrijving