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