

PowerShell - Continued

1. Oefeningen

1.1. Pipelining

Het meeste voorkomende gebruik van Powershell gebeurt met pipelines. Waarbij we verschillende commando's na elkaar kunnen plaatsen die sequentieel informatie zal transformeren.

Kijk eens naar de volgende ketting:

```
Get-ChildItem | Where-Object Length -gt 100 | Sort-Object Length -Descending |  
Format-Table Name, Length
```

Probeer in je hoofd voor te stellen hoe de informatie achter deze commando's zou uitzien na elke stap in de ketting. Indien je het niet meteen ziet, kan je de ketting sequentieel opbouwen om te zien welke informatie we initieel hebben en hoe deze zich transformeert.

```
# Stap 1 : Met dit commando krijgen we een lijst van files en directories  
Get-ChildItem  
  
# Stap 2 : Met deze ketting hebben we dezelfde lijst maar gefilterd op grootte  
Get-ChildItem | Where-Object Length -gt 100  
  
# Stap 3 : Dan wordt deze gefilterde lijst ook nog eens gesorteerd  
Get-ChildItem | Where-Object Length -gt 100 | Sort-Object Length -Descending  
  
# Stap 4 : Tenslotte gaan we deze lijst opstellen in een overzichtelijk tabel met naam en grootte  
Get-ChildItem | Where-Object Length -gt 100 | Sort-Object Length -Descending |  
Format-Table Name, Length
```

Je kan alle kettingen opbouwen op deze manier, startende met het eerste commando en zo meer en meer verfijningen toepassen tot je het gewenste resultaat hebt. Met dit resultaat kunnen we dan een output van weergeven, wegschrijven naar een file of gebruiken als deel van een script.

Om dit te oefenen, ga je nu zelf een ketting opbouwen.

1. Lijst alle processen. Zoek eens op met welk cmdlet je dit zou kunnen doen.
2. Lijst alle processen op met een naam die minsten één letter 'a' hebben.
3. Lijst alle processen op met een naam die minsten één letter 'a' hebben en sorteer deze lijst op CPU gebruik van hoog naar laag.
4. Lijst alle processen op met een naam die minsten één letter 'a' hebben en sorteer deze lijst op CPU gebruik van hoog naar laag en selecteer enkel de namen en CPU gebruik van processen als resultaat. *(Gebruik hier Select-Object in plaats van Format-Table)*
5. Schrijf het resultaat van oefening 4 weg naar een file `processes.txt`.
6. Schrijf het resultaat van oefening 4 weg naar een CSV file `processes.csv`. Hiervoor ga je in plaats van redirectie naar een file gebruik moeten maken van een cmdlet.
7. Converteer het resultaat van oefening 4 naar HTML en schrijf dan deze weg naar een file `processes.htm`. Hierbij ga je een cmdlet moeten gebruiken om de output om te zetten naar HTML en daarna redirectie naar een file.

1.2. Scripting

In tegenstelling tot vorige les, gaan we met scripting nu geen functies maken maar alles uitschrijven in het script zelf. Ik wil bijvoorbeeld een script maken dat de omtrek van een vierkant berekent.

```
# Omtrek.ps1
# Gegeven variable $zijde
$zijde = 5

# Berekening en opslagen in variable $omtrek
$omtrek = $zijde * 4

# $omtrek als output tonen
$omtrek
```

Het zou nog beter zijn als we zijde als parameter zouden meegeven. In dit geval gaan we niet gebruik maken van een functie met parameters, maar argumenten die we meegeven wanneer we het script uitvoeren. In het script kunnen we deze argumenten opnemen met `$args[x]` waarbij `x` aangeeft de hoeveelste parameter je wilt hebben startende vanaf 0.

Maw als ik een script uitvoer als `.\Omtrek.ps1 aba ada`, dan zal `aba` in `$args[0]` en `ada` in `$args[1]` zitten. Als we `Omtrek.ps1` aanpassen met `$args[0]` om een zijde mee te geven dan krijgen we het volgende:

```
# Omtrek.ps1
# Gegeven argument $zijde
$zijde = $args[0]

# Berekening en opslagen in variable $omtrek
$omtrek = $zijde * 4

# $omtrek als output tonen
$omtrek
```

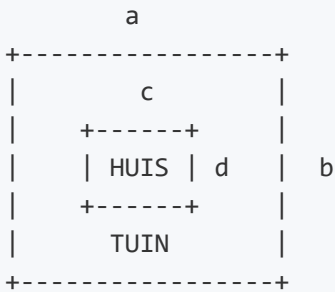
Nu krijgen we enkel een getal als output, misschien willen we wat duidelijker zijn van wat er gebeurt in het script. Gelukkig is het mogelijk om in het script meerdere outputs te geven en eventueel ook tekst toe te voegen.

```
# Omtrek.ps1
# Gegeven argument $zijde
$zijde = $args[0]

# Berekening en opslagen in variable $omtrek
$omtrek = $zijde * 4

# Output de hele bewerking
"zijde = " + $zijde.ToString() + " m"
"omtrek = " + $zijde.ToString() + " m * 4 = " + $omtrek.ToString() + " m"
```

1. Maak een script dat met een argument als zijde de oppervlakte berekent van een vierkant.
2. Maak een script dat met 2 argumenten breedte en hoogte de oppervlakte berekent van een balk.
3. Bekijk even het volgende plaatje, we zien een grondgebied met huis en tuin. De breedte en hoogte van het grondgebied zijn a en b. De breedte en hoogte van het huis zijn c en d. Maak een script dat met argumenten a, b, c, d de oppervlakte van de tuin berekent.



```

      a
  +-----+
  |               |
  |      c         |
  |  +-----+     |
  |  | HUIS | d    | b
  |  +-----+     |
  |      TUIN      |
  |               |
  +-----+
  
```

a = 20
 b = 15
 c = 10
 d = 8
 Oppervlakte grond = 20 m * 15 m = 300 m²
 Oppervlakte huis = 10 m * 8 m = 80 m²
 Oppervlakte tuin = 300 m² - 80 m² = 220 m²

4. Breid het script van 3 uit zodat je mooi de hele bewerking ziet als output.

```

Oppervlakte grond = 300 m^2
Oppervlakte huis = 80 m^2
Oppervlakte tuin = 300 m^2 - 80 m^2 = 220 m^2
  
```

5. Maak een script dat voor een gegeven argument `getal` laat zien of dit getal deelbaar is door 2. Dit kan je bereiken met de `%` operator, deze operator berekent de restwaarde, hieronder een voorbeeld.

```

9 % 3 = 0 # 9 / 3 = 3 met rest 0
10 % 3 = 1 # 10 / 3 = 3 met rest 1
11 % 3 = 2 # 11 / 3 = 3 met rest 2
12 % 3 = 0 # 12 / 3 = 4 met rest 0
12 % 4 = 0 # 12 / 4 = 3 met rest 0
  
```

1.3. If-Else Statements

If-else statements ken je wellicht nog, ook deze kunnen we rechtstreeks gebruiken in een script.

```
$zijdes = $args[0]

if ($zijdes -le 1) {
    "Dit is geen veelhoek"
}

elseif ($zijdes -eq 4) {
    "Dit is een vierhoek"
}

elseif ($zijdes -eq 5) {
    "Dit is een vijfhoek"
}

else {
    "Dit is een veelhoek"
}
```

1. Maak een script dat voor een gegeven argument `bedrag` 21% BTW berekent. Hiervoor heb je nog geen if-else nodig.
2. Breid het script van 1 uit zodat we BTW berekenen op basis van een gegeven `land` in plaats van enkel België. De mogelijke landen kan je in onderstaande tabel terugvinden.

Land	% BTW
BE	21
NL	21
FR	20
DE	19
LU	17

1.4. Foreach

Voor herhalingen gaan we dit lab enkel verder met foreach, aangezien for en while minder vaak voorkomen binnen scripting. De foreach loop kunnen we gebruiken om te itereren over een lijst van objecten of getallen.

```
# Deze ketting geeft als resultaat een lijst van file en directory namen
$lijst = Get-ChildItem | Select-Object Name

# Op deze lijst kunnen we dan itereren
foreach ($naam in $lijst) {
    $naam
}

# Als we willen tellen van 1-10 kunnen we gebruiken maken van `n..m`
foreach ($teller in 1..10) {
    $teller
}

# 1 en 10 kan je aanpassen naar andere getallen of vervangen met variabelen
$begin = 1
$eind = 5
foreach ($teller in $begin..$eind) {
    $teller
}
```

1. Maak een script dat voor een gegeven argument `getal` teruggeeft of het een priem getal is of niet. De eenvoudigste manier om dit op te lossen is om een loop te schrijven van 2 tot het getal en kijken of deze deelbaar is. Gebruik hiervoor de operator met rest waarde.

```
4 => False
13 => True
```

2. Maak een script dat voor een gegeven argument `getal` alle priemgetallen teruggeeft kleiner dan het gegeven getal.

```
# getal = 20
2
3
...
19
```

2. Active Directory

Om met Powershell te werken in Active Directory gaan we verder moeten werken op de Windows Server VM. Gelukkig kunnen we Powershell scriptjes schrijven zonder Visual Studio Code.

In de volgende stappen gaan we elementen van vorige labs terugzien maar dan in commando vorm. We gaan hier niet al te diep op in gaan aangezien je dit allemaal al gezien hebt door gebruik te maken van Windows' user interface. Volg mee met de lector hoe je deze scripts kan maken op Windows Server en wat je hiermee kan doen.

Dit sectie is louter ter voorbeeld van wat je zou kunnen doen met Powershell en is geen leerstof voor het examen.

Op je Windows Server VM, maak een folder `PowerShellScripts` op desktop aan, maak een nieuwe file `Script.ps1` aan en dubbelklik hierop.

2.1. User & Computer Management

Er zijn heel wat cmdlets die we kunnen gebruiken om Active Directory te beheren met Powershell. Als je deze eens nodig hebt kan je die best opzoeken met behulp van de [Microsoft documentatie](#).

1. Hier een scriptje waarmee je één user kan aanmaken, waaraan je de opties voornaam en naam meegeeft.

```
$voornaam = $args[0]
$achternaam = $args[1]
New-ADUser -Name "$voornaam $achternaam"
```

2. Als je een heleboel users wil aanmaken kan je dit doen met een CSV-bestand. Hieruit kan je alle users lezen en één voor één aanmaken.

```
foreach ($user in Import-Csv .\users.csv -Header Voornaam, Achternaam) {
    New-ADUser -Name ($user.Voornaam + " " + $user.Achternaam)
}
```

Kijk eens na of deze nieuwe Users er tussen staan, het kan zijn dat je moet refreshen.

2.2. DNS Management

1. Om een DNS A record toe te voegen kan je een script maken als.

```
Add-DnsServerResourceRecordA -Name "newhost" -ZoneName "cosci.be" -  
AllowUpdateAny -IPv4Address "10.10.11.10"
```

2. Om DNS Zone te pauzeren en hervatten kan je ook telkens een script maken.

```
Suspend-DnsServerZone -Name "cosci.be" -Verbose -Force
```

```
Resume-DnsServerZone -Name "cosci.be" -Verbose -Force
```

2.3. DHCP Management

1. Hier een script die alle DHCP leases oplijst van een scope

```
$scope = $args[0]  
Get-DhcpServerv4Lease -ScopeId $scope -AllLeases
```

2. Dan een script die een DHCP scope aanmaakt.

```
Add-DhcpServerv4Scope -Name "NewScope" -StartRange 10.10.11.1 -EndRange  
10.10.11.30 -SubnetMask 255.255.255.0
```

3. Tenslotte een script die een DHCP scope verwijderd.

```
Remove-DhcpServerv4Scope -ScopeId 10.10.11.0 -Force
```


3. Wat moet je na dit labo kennen/kunnen

- Je moet pipelines kunnen maken met gegeven cmdlets.
- Je moet eenvoudige scripts kunnen schrijven met variabelen.
- Je moet eenvoudige scripts kunnen schrijven met if-else statements.
- Je moet eenvoudige scripts kunnen schrijven met een foreach loop.
- Je moet niet alle cmdlets kennen, deze worden gegeven.
- Je moet geen functies leren om Active Directory te beheren.