

# Introductie Powershell

Juli 2016

## Objecten

### Introduction Powershell

1. Objecten
2. Werken in de Pipeline
  - Sorteren
  - Selecteren
  - Filteren
  - Vergelijken
3. the Formatting Subsystem
4. De geavanceerde pipeline

Om powershell echt te begrijpen moet je objecten begrijpen. Bijna alles in powershell is een object. Objecten zijn simpel gezegd een digitale presentatie van iets dat bestaat uit onderdelen en details hoe je deze onderdelen kunt gebruiken.

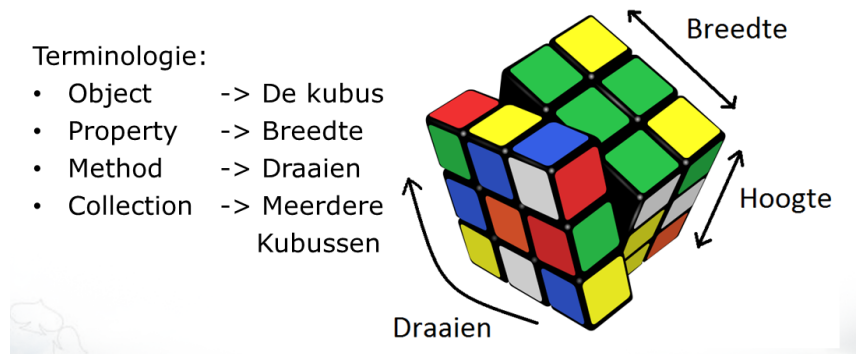
Denk bv aan een fiets. Die heeft onderdelen, wielen, pedalen, stuur, zadel. en hiermee kun je links af trappen, remmen etc.

In Powershell hebben we het over properties (onderdelen) en methods (hoe te gebruiken)

In Powershell bestaat alles uit objecten

Terminologie:

- Object    -> De kubus
- Property -> Breedte
- Method   -> Draaien
- Collection -> Meerdere Kubussen

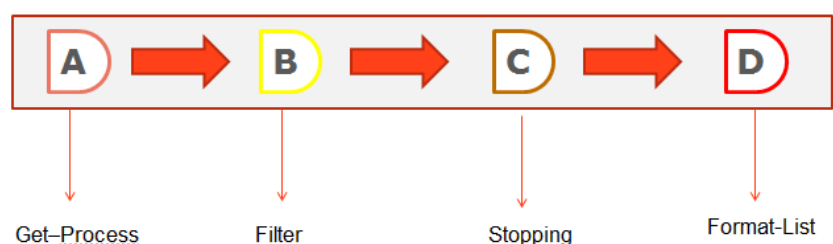


In Powershell is bijna alles een object. Of we nu werken met mailboxen, gebruikers, processen, event logs of netwerkkaarten. Ze worden allemaal gepresenteerd als een object en hebben properties en methods

## Werken in de Pipeline

Powershell is een shell en een scripttaal. Je hoeft geen complex script te schrijven. Soms is een regel in de shell al voldoende.

We gaan nu kijken naar de basis commando's van de pipeline en hoe we deze kunnen filteren, vergelijken, selecteren en sorteren. Vervolgens gaan we kijken hoe we de output kunnen formatteren zodat we krachtige commando's kunnen creëren.



*Een object is een verzameling van onderdelen en hoe je deze kunt gebruiken.*

## Vergelijkings Operators

In powershell worden geen wiskundige operators gebruikt. De rede hiervoor is het > symbool. Deze betekend in powershell namelijk redirect output. Om deze te behouden gebruikt powershell andere vergelijkingen operators.

Vergelijkingen operators kunnen ook gebruikt worden met "regular expressions". Reguliere expressies vallen buitenscope maar zijn erg krachtige manier voor het werken met tekst.

Vergelijkingen operators  
kunnen ook gebruikt  
worden met reguliere  
expressies

Math Symbol	Powershell Operator	Meaning
=	-eq	gelijk
<> or !=	-ne	niet gelijk
>	-gt	groter dan
>=	-ge	groter dan of gelijk aan
<	-lt	kleiner dan
<=	-le	kleiner dan of glijk aan
	-like	Wildcard vergelijking
	-notlike	Wildcard vergelijking
	-match	regular expression vergelijking
	-notmatch	regular expression vergelijking

Powershell, een nieuwe manier van  
beheren.

Meer info :      Get-help about\_comparison\_operators  
Get-help about\_regular\_expressions

## Filteren (Where-Object)

Om uiteindelijk je output te krijgen kun je gebruik maken van filteren.  
We onderscheiden 2 manieren van filteren. Links en rechts filteren.

Links filteren doen we door gebruik te maken van de cmdlet parameters  
Rechts filteren doen we door de output te pipen naar where-object.

Where-Object

Aliassen	Where
	?

Er zijn een paar speciale karakters die in where-object filter gebruikt worden.  
De eerste zijn "Brackets" {}. Wanneer deze gebruikt worden dan geven ze in powershell een scriptblock aan.

```
process | where-object {$_.status -eq "stopped" }
```

In dit voorbeeld bevat {} het scriptblock dat alleen de process objects filtert die gestopt zijn.  
Where-object

- checked ieder object dat in het filter gestopt wordt.
- checked ieder object of deze aan het filter voldoet.
- Meerdere filters kunnen gebruikt worden (-and en -or)
- \$\_ representeerd ieder object.

```
Get-process | where {$_.status -eq "stopped" -and $_.name -like " w*"} 
```

## Select-Object

Select-object kan gebruikt worden om bepaalde properties van een object weer te geven.

Aliassen	select
	?

Key parameters : -property, -first, -last

Get-process | select-object ProcessName,ID,WS

## Sort-Object

Sort-object is een cmdlet dat data kan sorteren.

Aliassen	sort
----------	------

Key parameters : -property, -descending

Get-process | sort-object -property workingset -descending  
Dir | ?{\$\_length -gt} | sort -property length

## The Formatting subsystem

Powershell cmdlets produceren objecten als output. Deze objecten bevatten meestal meerdere properties. Default laat powershell er maar een paar zien. Deze default formatting staat gedefinieerd in het bestand dotnettypes.format.ps1xml

(Folder → C:\Windows\System32\WindowsPowerShell\v1.0)

Ps c:\> cd \$PSHome  
Ps c:\> notepad dotnettypes.format.ps1xml

In powershell zijn er 4 formatting cmdlets. We zullen de 2 belangrijkste bespreken.

### Format-table

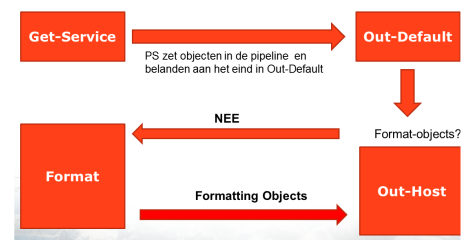
Dit cmdlet zet de output om naar een tabel.

Default Alias : FT  
Key parameters : -property, -autosize

D.m.v. de parameter -property kun je de properties kiezen die je wilt zien. Zonder deze parameter zullen de default properties in tabel format te zien zijn.

Get-service | format -table -property name,dependentservices  
Get-eventlog system -newest 5 | ft eventid, TimeWritten -autosize

Belangrijk om te weten is dat bij formatting cmdlets de structuur verandert van de objecten. Formatting cmdlets veranderen de output van het object waardoor deze alleen nog te gebruiken zijn om te displayen in het scherm. Daarom moeten de formatting cmdlets altijd aan het eind van de pipeline staan.



*Daarom moeten format Cmdlets altijd aan het eind van de pipeline staan*

**Format-list**

Dit cmdlet zet de output om naar een list format.

Cmdlets zijn specifiek en speciaal ontworpen voor powershell.

Default alias : FL  
Key parameter : -property,

D.m.v. de parameter -property kun je de properties kiezen die je wilt zien. Zonder deze parameter zullen de default properties in list format te zien zijn.

## De geavanceerde pipeline

We hebben eerder gezien dat de Pipeline is afgekeken van unix. I.p.v. de output naar tekst te zetten kan Powershell de output gebruiken als input voor het volgende commando.

Nu we dit weten kunnen we verder

Command A | Command B

We maken een tekst file met Computernamen (c:\demo\computers.txt). De computernamen gebruiken als input voor een commando B. Vervolgens vertellen Commando B om deze computers te gebruiken als input.

Wanneer Get-content start, zal powershell de computernamen uit computer.txt halen en deze in de pipeline plaatsen. Powershell moet dan beslissen hoe ze deze objecten als input in de get-service commando moet krijgen.

De truc met Powershell is echter dat cmdlets alleen parameters accepteren.

Dit uitvind proces wordt parameterbinding genoemd. Powershell heeft 2 methodes die het kan gebruiken om de output van get-content te gebruiken als parameter input voor get-service.

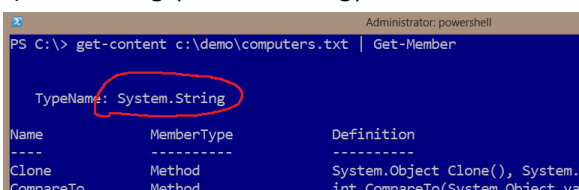
De eerste methode is *ByValue*. Als dat niet werkt dan probeert Powershell de *ByPropertyValue* methode

**ByValue**

Via deze methode kijkt Powershell naar het object Type geproduceerd door command A

get-content c:\demo\computers.txt | Get-Member

Je ziet dat get-content objecten produceert van het Type System.String (oftewel string)



```
Administrator: powershell
PS C:\> get-content c:\demo\computers.txt | Get-Member

TypeName: System.String

Name      MemberType Definition
-----
Clone     Method      System.Object Clone(), System.
CompareTo Method      int CompareTo(System.Object va
```

We gaan nu checken welke parameters van get-service een object accepteert van het type string via de ByValue methode.

```

-Name <String[]>
    Specifies the service names of services to be retrieved. Wildcards are permitted. By default, Get-Service gets all of the services on the computer.

Required?                false
Position?                1
Default value             All services
Accept pipeline input?    true (ByPropertyName, ByValue)
Accept wildcard characters? true
  
```

Je ziet dat get-service een parameter heeft die een string accepteert als input via ByValue. Helaas accepteert deze parameter alleen servicenamen en gaat dus niet werken.

```

PS C:\> get-content C:\demo\computers.txt | get-service
get-service : Cannot find any service with service name 'localhost'.
At line:1 char:37
+ get-content C:\demo\computers.txt | get-service
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (localhost:String) [Get-Service], ServiceCommandException
+ FullyQualifiedErrorId : NoServiceFoundForGivenName,Microsoft.PowerShell.Commands.GetServiceCommand
  
```

Dit werkt wel:

Open calc

ps c:\>get-process -name calc\*

ps c:\> get-process -name calc\* | Stop

### BypropertyValue

Als Byvalue niet werkt zal Powershell het gaan proberen via de BypropertyValue methode. Met deze methode, is het mogelijk dat meerdere parameters van command B betrokken raken.

Ps c:\> Get-service -name sec\* | stop-service

Even opnieuw, pipe de output van Commando A naar get-member en kijk dan naar de syntax van Commando B. De output van commando A heeft een property van wie de naam correspondeert met een paramater van commando B.

Get-service -name sec\* | get-member → (zie linker scherm)

get-help stop-service → (zie rechther scherm)

Property naam → Match → paramater naam

```

PS C:\> get-service -name s* | gm
TypeName: System.ServiceProcess.Service
Name      MemberType De
----      -
Name      AliasProperty Na
RequiredServices AliasProperty Re
Disposed   Event
Close      Method
Continue   Method
CreatedObjRef Method
Dispose    Method
Equals     Method
ExecuteCommand Method
GetHashCode Method
GetLifetimeService Method
GetType    Method
InitializeLifetimeService Method
Pause      Method

PS C:\> get-help Stop-Process
NAME
    Stop-Process
SYNOPSIS
    Stops one or more running processes.
SYNTAX
    Stop-Process [-Id] <Int32[]> [-Force] [<SwitchParameter>] [-PassThru [<SwitchParameter>]] [-Confirm [<SwitchParameter>]] [-WhatIf [<SwitchParameter>]] [<CommonParameters>]
    Stop-Process [-InputObject] <Process[]> [-Force] [<SwitchParameter>] [-PassThru [<SwitchParameter>]] [-Confirm [<SwitchParameter>]] [-WhatIf [<SwitchParameter>]] [<CommonParameters>]
    Stop-Process [-Force] [<SwitchParameter>] [-PassThru [<SwitchParameter>]] [-Name <String[]>] [-Confirm [<SwitchParameter>]] [-WhatIf [<SwitchParameter>]] [<CommonParameters>]
DESCRIPTION
  
```

Dit zal werken!

```

PS C:\> Get-service -name sec* | stop-service -whatif
What if: Performing operation "Stop-Service" on Target "Secondary Logon (seclogon)".
PS C:\>
  
```

## Providers

Een provider is een gemeenschappelijke interface naar verschillende datastores. Denk bij een datastore aan het "file system".

Een PowerShell provider is een adapter, deze zorgt er voor dat een bepaald data storage medium te zien is als een diskdrive

Voor meer informatie : `get-help about_provider`

---

*Denk bij een  
datastore aan het file  
system*

---