PowerShell

Introduction

Powershell est un interpréteur de commandes et un langage script orienté objet. Il s'appuie sur le framework .Net Core

Powershell permet :

- L'administration du système
- L'automatisation des tâches (création de comptes dans l'AD par exemple)
- Développement d'outils pour la simplification de l'administration
- Powershell est multiplateforme et Opensource

Toutes les technologies Microsoft sont basées sur Powershell, mais on le retrouve aussi chez d'autres constructeurs : Vmware, Citrix, …

Windows Powershell a vu le jour en 2006 et fût à l'origine uniquement développé sous Windows. Microsoft a pris conscience que l'administration ne pouvait se faire tout en graphique. La version PowerShell Core Opensource fut présentée en 2018. Il peut être utilisé sous Windows, Linux, et MacOS.

Depuis cette année (2020), Powershell Core devient PowerShell et remplace Windows Powershell.

Notion d'objet

Powershell est donc un langage orienté objet, cela signifie que tout ce que nous manipulerons sera considéré comme un objet : texte, fichiers, tableaux, … Un objet informatique est une information structurée. Chaque objet possède des propriétés et des méthodes. Une propriété est une information sur l'objet, et une méthode est l'action que l'objet pourra effectuer.

Prenons l'exemple d'un objet de type fichier :

Ses propriétés :

- Sa taille : 1024 ko
- Date de création : 15/02/2019
- Date de modification : 20/03/2019

Ses méthodes :

- Suppression du fichier
- Renommer le fichier

Exécution de PowerShell

On peut lancer la console via le menu, ou encore en exécutant powershell.exe (éventuellement suivi d'une commande : powershell.exe –noexit –command Get-Date)

La commande powershell.exe contient quelques options :

- -Version : affiche le numéro de la version PS
- -WindowStyle (Maximized, Minimized ou Hidden) : définit la taille de la fenêtre
- -WorkingDirectory : indique de se positionner dans un chemin particulier (ex : C:\Windows)

Toutes ces options, et les autres, se retrouvent en tapant powershell.exe / ?

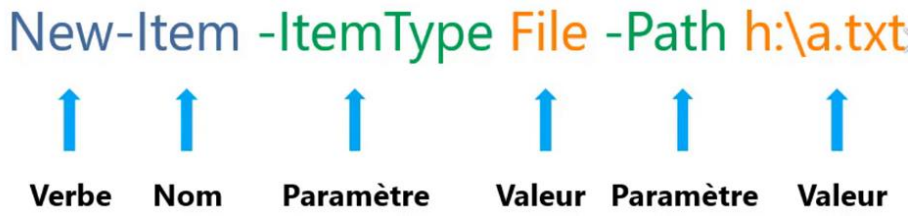
A partir de la console, il est possible d'exécuter des commandes Windows (CMD), mais aussi de lancer des applications.

Syntaxe des commandes Powershell

Le format de la syntaxe d'une commande Powershell est du type suivant :

Verbe-Nom –Paramètre valeur

Une commande peut avoir plusieurs paramètres.



Une commande PowerShell se nomme **cmdlet** (on prononce commandlet). Il est possible d'enchainer plusieurs cmdlet en les séparant par des ;

L'ensemble des cmdlet de la machine s'obtient en tapant **Get-Command**. Chaque ligne correspond à un objet. Il est possible de filtrer avec le paramètre –verb, –noun, -name ou -commandtype. Par exemple :

Get-command –verb get -> affiche toutes les cmdlet commençant par le verbe get (récupérer quelque chose)

Get-command –noun alias -> affiche toutes les cmdlet qui contiennent le mot alias

Get-command –name get-date -> affiche les infos sur la cmdlet get-date

Get-command –commandtype cmdlet -> affiche toutes les commandes de type cmdlet

On peut retrouver tous les paramètres pouvant être passé à une commande en tapant :

(get-command get-command).parameters

Comme d'habitude, il est possible d'utiliser les caractères génériques :

Get-command *date -> affiche les cmdlet se terminant par date

Powershell est en fait constitué de cmdlet issues de modules. Il peut être important d'importer ces modules pour disposer des commandes précises (gestion AD, hyperV, …)

Afin de visualiser toutes les commandes issues d'un module particulier, nous utiliserons la cmdlet suivante :

Get-command –module microsoft.powershell.archive

Trouver de l'aide

Afin de rendre disponible l'aide en ligne, il faut d'abord télécharger cette dernière :

Update-help (en administrateur)

Une fois installé, nous utiliserons la cmdlet suivante :

Get-help cmdlet

Par exemple : get-help get-command –examples (pour obtenir des exemples d'utilisation)

Notions de variables

Comme sous le bash, il est possible sous PS de déclarer des variables. Une variable commence toujours par un $ (non sensible à la casse) :

$var = "www.google.com"

Pour afficher le contenu d'une variable on tape simplement son nom : ici $var

Pour vider une variable : $var= $null

Pour PS www.google.com est un objet référencé par la variable $var. On peut donc accéder à ses propriétés et méthodes.

Comme nous l'avons vu au dessus, nous accèderons aux propriétés et méthodes de l'objet en tapant

$var | get-member

Pour accéder aux propriétés, il suffit de taper la commande suivi d'un point, suivi du nom de la propriété. Par exemple $var.length pour afficher sa longueur en caractère.

Même opération pour utiliser une méthode, on indique la variable, suivi d'un point, suivi d'une méthode. Une méthode utilise toujours des parenthèses. Par exemple pour modifier la variable en remplaçant le .com par un .fr, nous taperons la commande suivante :

$var.replace("com", "fr")

A noter que PS n'a pas modifié mon objet, mais en a créé un nouveau. Si nous avions voulu remplacer .com par .fr dans la variable $var, il aurait fallu effectuer :

$var = $var.replace("com", "fr")

Les variables sous PS sont typées (entier, chaine, booléen, …). Afin d'afficher le type de variable que PS a affecté, on utilisera une méthode gettype() :

$var.gettype() (ici une chaine de caractère)

$age = 10 -> $age.gettype() nous renvoie un entier

Attention $age = "10" est pris comme un entier (à cause des guillements)

$age = $age + 2 permet d'effectuer des opérations sur les variables.

Il est aussi possible d'affecter un chemin à une variable. Par exemple :

$chemin = "C:\windows\write.exe"

Dans ce cas, pour lancer l'application il suffira de taper la commande suivante :

& $chemin

Enfin il est possible de stocker le résultat d'une commande dans une variable (comme en bash)

$date = get-date

Les opérateurs

Opérateurs arithmétiques

Ils permettent de faire des calculs arithmétiques : + - * / et % (modulo)

On peut stocker les résultats dans des variables : $result = 2 * 7

Il est aussi possible de multiplier les caractères : "a" * 5 donnera aaaaa

Opérateurs d'affectation

Ils permettent d'affecter des valeurs à des variables. Nous avons déjà vu le =

Il existe un opérateur d'affectation qui permet d'incrémenter une variable : +=

Par exemple $var = 10

$var += 5

$var -> 15

Mais on peut aussi effectuer cette opération sur des chaines de caractères. Par exemple :

$test = "Greta"

$test += "TSSR"

$test -> Greta TSSR

Opérateurs de comparaison

Comme sous linux, nous utiliserons –lt, -gt, -le, -ge, -eq, et –ne

Le résultat renverra un booléen true ou false. Cela est valable pour des valeurs numériques, mais aussi des chaines de caractère.

En outre il est possible d'utiliser les caractères génériques pour comparer 2 valeur en introduisant ces valeurs par –like ou –notlike. Par exemple :

PowerShell –like Power* -> true

PowerShell –like Power?hell -> true

Exemple concret

$ip = "192.168.1.2é

$ip –like "192.168.1.*"

- ➔ True. Nous pourrons donc agir en conséquence si l'ip fait partie du réseau

Opérateurs logiques

ET logique : -and (true si les 2 valeurs sont vraies)

OU logique : -or (true s'il au moins l'une des valeurs est vraie)

OU Exclusif : -xor (true si 1 seule valeur est vraie)

Non logique : -not (inverse)

Opérateurs de fractionnement et de concaténation

Il est possible de fractionner des valeurs en fonction d'un caractère introduit par l'option –split :

"PowerShell,Python,Bash" –split (",") nous donnera :

PowerShell

Python

Bash

Il est aussi possible de concaténer des valeurs entre elles avec l'opérateur –join :

"PowerShell","Bash","Python" –join "-" nous donnera :

PowerShell-Bash-Python

Utilisation des Pipe

Comme sous linux, il est possible d'envoyer le résultat d'une commande vers une autre :

Get-command | out-file c:\fic.txt

Get-process | Select-object Processname

Get-content –Path c:\nontri.txt | sort-object | out-file c:\tri.txt

Afin de visualiser les méthodes et propriétés d'un objet, nous utiliserons | get-member

Filtrage en sortie

Il existe 4 propriétés de filtrage en sortie de commande :

- Where Object

Ex : Get-Service | Where-Object –property status –eq "Running" pour visualiser tous les services démarrés

Get-Childitem c:\windows | where-object –property lenght –lt 1MB

- Foreach-Object

Get-Childitem c:\windows –file | foreach-object {"Taille du fichier $($_.name) : " + $_.length }

Afficher tous les fichiers de c:\windows et leur taille

$_ correspond à l'objet reçu par le pipe

1..5 | foreach-Object {new-item –path c:\ -name "Fichier-$_.txt"

- Group-object

Get-service | group-object –property status

Indique le nombre d'objet en fonction du statut

- Sort-object

Get-childitem c:\windows | sort-object –property length

Affiche le dossier windows trié par taille de fichiers

Sécurité dans l'éxécution des scripts

Par défaut, PowerShell interdit l'exécution de scripts non signés

Get-ExecutionPolicy

Afin de pouvoir importer des scripts existants il faut forcer l'exécution :

Set-ExecutionPolicy bypass

Modules Powershell

Nous avons vu que PS fonctionne par module. Un module est un package contenant des scripts, des variables, des fonctions, des alias, …

Il est possible d'importer des modules via internet ou de développer soi-même ses propres modules. Cela permettra d'étendre les possibilités de PS. Il existe 3 types de modules :

- Script

Il s'agit d'un script PS qui porte l'extension .psm1 et qui contient du code PS

- Binaire

Il s'agit d'un module compilé au format dll. C'est le type le plus courant fourni entre autre par MS, Citrix, VMware, …

- Manifeste

C'est un fichier qui porte l'extension .psd1 et décrira le module (version, auteur, …)

Pour lister tous les modules de notre machine : get-module –ListAvailable

Quelques commandes à trouver …

```
Get-ChildItem -Path h:\ressources –Recurse

Get-ChildItem -Path h:\ressources -Recurse -Exclude *.txt

Set-ItemProperty -Path h:\test.txt -Name IsReadOnly -Value $true

Copy-Item -Path H:\ressources\ -Recurse -Destination h:\res

Rename-Item -Path H:\ressources\fichier1.txt -NewName test.txt

Restart-Service -Name Audiosrv

Get-Service | Where-Object -Property StartType -eq "Disabled"

Start-process notepad ; Wait-Process notepad ; Write-Host "Notepad vient d'être
fermé. CMD va s'ouvrir" ; Start-Process cmd

$MotDePasse = ConvertTo-SecureString "Azerty02" -AsPlainText -Force
New-LocalUser -Name User3 -Description "Utilisateur test" -FullName "Jean Dupont" -
Password $MotDePasse -AccountExpires "17/08/2020"

Get-LocalGroupMember -Name administrateurs

Remove-LocalGroupMember -Group administrateurs -Member user1

get-disk | format-list

get-partition -DiskNumber 1

Set-Partition -DriveLetter P -NewDriveLetter I

Get-ChildItem -Path h:\test | Where-Object -Property Length -LE 11KB | ForEach-
Object {Remove-Item -Path $_.FullName}

Get-NetAdapter

(Get-NetAdapter -InterfaceIndex 5).MacAddress

Get-NetIPAddress -AddressFamily IPv4

New-NetIPAddress -InterfaceIndex 10 -IPAddress 192.168.0.1 -PrefixLength 24 -
DefaultGateway 192.168.0.5
```

```powershell
Set-NetIPInterface -InterfaceIndex 10 -Dhcp Enabled


Set-DnsClientServerAddress -InterfaceIndex 10 -ServerAddresses
"10.10.10.10","20.20.20.20"

Get-NetTCPConnection


Get-NetFirewallRule


New-NetFirewallRule -DisplayName "Bloquer le ports 80" -Description "Cette règle
permet de bloquer le trafic vers le port 80" -Direction Outbound -RemotePort 80 -
Protocol TCP -Action Block


(Get-CimInstance Win32_Processor).Name
```