Powershell en pratique

Sommaire

Table des matières

1)Prérequis

Avant de commencer je vous invite à relire vos cours d'algorithme

Pour installer et utiliser PowerShell dans les meilleurs conditions, je vous invite à faire quelques modifications.

En fonction de votre système d'exploitation, vous devez installer :

-Windows Management Framework 4.0

-Windows Management Framework 5.1

-Microsoft .Net Framework4.5

-NotePad++

- Et pour faire le lien avec Active directory, vous pouvez aussi installer RSAT

-vous pouvez aussi essayer Powershell Plus qui est un environnement de Dev sympa.

N'oubliez pas de lancer **« PowerShell ISE »** qui permet de créer des scripts et dispose d'une Interface plus sympa que la ligne de commande

#Et lancez PowerShell ISE en Administrateur !!!

Une fois Lancé, tapez cette commande :

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope LocalMachine
 # Par sécurité, et par défaut il est impossible de lancer des scripts
```

Créez vous un dossier de travail à la racine de votre pc, du genre c:\TEMP ou c:\PS

Soyez sûr d'avoir lancé votre IDE avec des droits d'administrateurs et c'est parti !

2)Variable

A)Théorie

##########----------Déclaration----##########

New-Variable -name MaVariable -Value 42 # declaration de variable sans utiliser de raccourci

$MaVariable1 = 12.34 # Déclaration de variable (nombre)

$MaVariable2 = "Montexte" # Déclaration de variable (texte)

$MaVariable3 = $true # Déclaration de variable (bool)

##########

Write-Host `n # Saute une ligne à l'affichage

Write-Warning "Pour la Variable 1 " # Affiche une message de type warning

write-host "la Variable 1 Vaut $MaVariable1 et elle est de type $($MaVariable1.GetType().Name) " # Affiche une message

Write-Host `n # Saute une ligne à l'affichage

##########

Write-Warning "Pour la Variable 2 " # Affiche une message de type warning

write-host "la Variable 2 Vaut $MaVariable2 et elle est de type $($MaVariable2.GetType().Name) " # Affiche une message

Write-Host `n # Saute une ligne à l'affichage

##########

Donnez le type pour les variables 3 et 4

##########

##########

On peut définir à la création le type de variable

[string]$var54 = "RE"

En voici d'autres :

```
[string] caractère       [int] nombre
[bool] Booléen           [DateTime]  Date
[array]  Liste
```

##########

Cela marche aussi avec des « listes »

La variable est alors une liste d'élément , on peut déjà introduire le terme Array

$MesFichiers = get-childitem

$MesServices = get-service

##########------Les Prédéfinies-----##########

#

$PSVersionTable # donne la version de powershell

$null # Vide

$true + $false # Vrai ou faux

$_ # objet en cours

$Error # liste des erreurs

$$ # dernier mot en cache

et beaucoup d'autre

Passons aux exercices

B)Pratiques

1) Créer une variable Var1 qui contient la valeur "yopmail"

2) Créer une variable Var2 qui contient la valeur "centre de formation"

3) Créer une variable Var3 de Type Bool et lui affecter la valeur "Faux"

4) Donner Votre version de PowerShell

5) Créer une variable Var4 qui contient la valeur "42,5476"

6) Créer une variable Var5 qui contient « False »

7) Créer une variable Var6 qui contient la valeur "1011010"

8) Demander à l'utilisateur de saisir son nom , puis son prénom.

9) Stocker la liste des processus systeme dans une variables MesProcess

3)Opérateurs

A)Théorie

##########----------Opérateurs arithmétiques----##########

Write-Warning "Opérateurs arithmétiques"

34+12 # Le plus

34-12 # Le moins

*34*12* # Le multiplié

34/12 # Le divisé

34%12 # Le modulo

#

Et ça marche aussi avec les variables :

Write-Warning "Opérateurs arithmétique avec variables"

Write-Warning "avec un type String"

$abcef = "je suis un "

$ijklm = "morceau de phrase"

$abcef + $ijklm

Write-Warning "avec un type Int"

$12 = 12

$34 = 34

$12+$34

Write-Host `n

#

##########----------Opérateurs Comparaison----##########

Write-Warning "Opérateurs Comparaison"

$true -eq $false # Egale (Equal)

$true -ne $false # N'est pas Égale (Not Equal)

34 -gt 12 # ou -ge pour le supérieur OU égale (Greater Than)

34 -lt 12 # ou -le pour le inférieur OU égale (Leather Than)

#

Attention à LIKE ET MATCH ----------> trouver la différence

```
"formation" -like "for"
"formation" -like "for*" # * tous les caractères
"formation" -like "for?" # ? N'importe quel caractère mais un seul
"formation" -like "?or?a*"
Write-Host `n
#
"formation" -match "for"
"formation" -match "for*" # * tous les caractères
"formation" -match "for?" # ? N'importe quel caractère mais un seul
"formation" -match "form[abc]"
"formation" -match "form[b-z]"
Write-Host `n
#
#comparaison de type
$Monnombre = 27
$Monnombre -is [bool]
$Monnombre -isnot [String]
Write-Host `n
#
##########################----------Opérateurs Logique----##########################
Write-Warning "Operateurs Logique"
$true -and $false # ET logique
$true -or $false # OU logique
$true -not $false # NON logique
$true -xor $false # OU exclusif
Write-Host `n
#
##########################----------Opérateurs redirection----##########################
#Write-Warning "Opérateurs redirection"
Get-ChildItem > c:\temp\file1.txt # remplace le contenu
Get-ChildItem >> c:\temp\file1.txt # ajoute à la fin
```

##########----------Opérateurs de remplacement ---##########

'Formation' -replace 'tion', 'teur'

$varmplace = 'formation'

$varmplace.replace('tion', 'teur')

Pour jouer avec les caractères vous avez aussi la commande Substring

("$varmplace".substring(0,1)) ou {$champ1.Substring(4,2)+'.'+$Champs2.Substring(6,9)}

Passons aux exercices

B)Pratiques

1) Demander à l'utilisateur de saisir un nombre1 puis un Nombre2 et lui afficher le produit des deux nombres.

2) Demander à l'utilisateur de saisir le prix d'un sandwich , puis de saisir le prix d'une bouteille d'eau. Lui demander combien de fois par mois il achète ce repas. Afficher le total dépensé par an.

3) Demander à L'utilisateur le prix d'un repas à la cantine. Afficher si il est vrai que prix d'un repas à la cantine est moins chère que ce qu'il dépense par jour à la question 2

4)Demander à l' utilisateur son nom et son prénom, lui construire un login de type :

Première lettre du prénom, point, le nom complet , puis @jaimepowershell.yopmail

Ex pour Mr Luc Dupont --------> L.Dupont@jaimepowershell.yopmail

5)Demander à l' utilisateur son nom et changer toutes les lettre "e" par des "a"

4)Tableaux

A)Théorie

##########----------Tableaux à 1 dimension----##########

##########----------Déclaration et affectation----##########

```
$Tableau1 = @("a","b","c","d")
$Tableau2 = @("a",2,[bool]$True,"d")
[string[]]$tab3 = "a","b","c"
$tab1[0]
$tab1[1]
```

Un tableau commence toujours à 0 !!!

```
$tab2 = 1..24
$tab2[3,7]
$tab2[$tab2.Length-1]
$tab2[-1]
```

Déclaration d'un tableau qui contient les chiffre de 1 à 10

```
$Montableau=@(1..10)
```

##########----------Modification/ajout----##########

#Modif

```
$MonTab = 'abc','def','ijk','lmn','opq','rst','uvw' # déclaration d'un tableau
$MonTab[0] = 'ABC' # Méthode de modification 1
$MonTab.SetValue('DEF',1)  # méthode de modification 2
```

#

#Suppression

il est impossible de supprimer un élément d'un tableau, on peux juste recréer un nouveau tableau sur la base de l'ancien

```
$MonTabFinal = $MonTab[0..3+5]
```

Ici on copie tout le tableau Montab sauf la 5eme ligne (le tableau commence à zéro donc la ligne numéro 4 est la 5eme)

#

ajout d'un élément à la fin du tableau

$tab3=@("a","b","c","d","e","f","g")

$tab3+= "h"

#

concaténation

$chaine1= @('P','o','w','E','r')

$chaine2= @('S','h','e','l','l')

$both = $chaine1+$chaine2

$both

$both[3] = 'e'

$both

#

#

##########----------Tableaux à 2 dimensions----##########

$MonTableau=@(0,0),(0,1),(0,2),(1,0),(1,1),(1,2),(2,0),(2,1),(2,2)

#Sinon il existe aussi des tableaux dit de hash ou tableaux associatifs

$Mesnotes = @{math = 12 ; francais = 11 ; Latin = 07 ; informatique = 18 ; anglais = 13 }

$Mesnotes | ft

$Mesnotes | fl

$Mesnotes['Latin']

$Mesnotes.francais

Passons aux exercices

B)Pratiques

1) Créer un tableau à 1 dimension de longueur 464 qui contient les nombre 100 à 563

2) Afficher le contenu de la 7eme case du tableau

3) Modifier le contenu de la 7eme case remplacer sa valeur par "AZERTY"

4) Recréer un tableau de longueur 463 qui contient la même chose que le tableau précèdent sauf le contenu de la 7eme case

5) Faire un tableau à 2 dimensions qui contient les noms de 5 jeux vidéos et de leur note attribuée par jeuxvideo.com

6) donner la note du deuxième jeu de la liste

7) faire un tableau avec le nom des couleurs primaires et leur code couleur html associé

5)PipeLine

A)Théorie

##########----------PipeLine ------##########

Le pipeline : "|" est un outil très important et nous l'avons déjà utilisé avant.

voici une très brève définition suivi d'exemple

la sortie d'une commande devient l'entrée d'un autre, ex :

$MesServices = Get-Service # j'ai une liste de tous les services en cours sur la machine.

#Si j'appel $mesServices , PowerShell m'affiche tout plein de chose sous forme de liste

#Essayons maintenant ceci :

$MesServices | Format-Table # rien de bien diffèrent car il affiche déjà sous forme de Table essayons autre chose

$MesServices | Format-List # l'affichage est maintenant sous forme de List

$MesServices | gm # cela nous donne l'ossature de MesServices (get-member)

j'attire votre attention ici le résultat de cette commande sera décortiqué plus tard mais retenez bien ceci pour le moment :

Dans la colonne MemberType nous allons nous focaliser sur les résultats don la valeur est Property

Nous allons jouer avec les propriétés par la suite donc mettez ca dans un coin de votre mémoire.

echo $MesServices | Out-GridView # echo $MesServices va afficher le contenu de la variable $MesServices et tout ce qui doit être affiché sera redirigé dans le Out-GridView

suite au" MesServices | gm" je me suis aperçu que un service à des propriétés et des méthodes

il est possible de ne sélectionner que ce qui nous voulons. Pour l 'exemple je ne veux que le nom et le statut

$MesServices | Select-Object name, status # ne m'affiche que le nom et status

$MesServices | Select-Object name, status | out-gridView # m'affiche dans un PopUp le nom et le statut.

$MesServices | Select-Object name, status | sort-object status | out-gridView # la même chose mais trié par statut.

même si nous allons en parler plus tard dans le cours on peut déjà utiliser export-csv

très très utile pour un administrateur : le fichier csv

$MesServices | Select-Object name, status | sort-object status | export-csv –path "c:\temp\ListeServices.csv" -Delimiter ";" -NoTypeInformation

Alors ça, c'est à retenir car le CSV on va en reparler

j'ai maintenant un fichier qui contient le contenu de ma Variables MesServices

#Un équivalent est le out-file

$mesServices | out-file –path "c:\temp\ListeServices2.txt"

$maliste | ForEach-Object {Write-Host $_.name }

ou aussi

$maliste | % name Get-ChildItem | Select-Object name | Sort-Object | Out-GridView

#Après ce bref aperçu de ce que l'on peut faire avec le "PipeLine" passons au filtre avec le "Where-Object".

Et là, ça commence à piquer.

########################## Rappel Objet ###################################

Petit Rappel ICI

Sans parler tout de suite d' OBJET, On va essayer de l'introduire par l'exemple :

Exemple le fichier

Get-childitem # Nous donnes les fichiers/dossiers du répertoire courant

$Mesfichiers = get-childitem # j'ai une liste des fichiers dans cette variable

cette variable est une liste (Array) et chaque ligne est un "Objet" (ici un objet fichier)

Comme dans la réalité , un objet possède des propriétés , ici par exemple un fichier possède :

un nom, un type, un chemin, une date de création ,etc...

La liste de ces attributs est consultable grâce à Get-childitem | gm

je regarde la colonne "MemberType " et je sélectionne ceux donc la valeur de "memberType" vaut "Property"

########################Where-object ##################################

pour le moment voici la syntaxe et les exemples, pratiquez, on en reparle après.

```
$Mesfichiers = get-childitem
$Mesfichiers | where-object {$_.name }
$Mesfichiers | where-object {$_.name -like "a*" }
```

chercher les fichiers dont le nom commence par un "a"

```
$Mesfichiers | where-object {$_.name -like "*e" }
```

chercher les fichiers dont le nom termine par un "e"

Avec les services c'est le même principe

```
$MesServices = get-service
$mesServices | where-object {$_.status -match "running"}
```

cherche tous les services dont le statut est "en cours"

Maintenant utilisons ce que nous avons déjà vu avant

```
$MesServices | Where-Object { ($_.status -match "running") -and ($_.name -like "w*") }
```

#ici nous cherchons les services qui ont un statut en cours mais aussi dont le nom commence par « w ».

Passons aux exercices

B)Pratiques

1) Mettre dans une variable $Maliste1 le nom et le statut de tous les services.

2) Mettre dans une variable $Maliste2 le nom et le statut de tous les services en cours.

3) Mettre dans une variable $Maliste3 le nom et le statut de tous les services en cours et dont le nom commence par un "w"

5) Mettre dans une variable $Maliste4 le nom et le statut de tous les services en cours et dont le nom commence soit par un "w" ou par un "s"

6) Mettre dans une variable $MesCommandes la liste des commandes PowerShell disponible (Get-command) et donner leur nombre.

7) Regarder le nom des attributs(propriétés, méthodes, etc..) possible. Et Afficher uniquement le nom des Propriétés .

8) Afficher les commandes qui sont en version 3 ou 3.x et dont le nom commence par "wr" (possible que en PowerShell V5)

6)Boucles

A)Théorie

##########----------LE WHILE----------##########

une boucle permet de répéter, autant de fois que l'on désire, une action ou un bloc d'action.

il en existe plusieurs types, commençons avec le While (tant que)

voici un exemple simple d'utilisation du while

While (CONDITION) {ACTION}

Attention aux parenthèses et au crochets , ne pas les confondre

```
[int]$Var1 = "0"
while ( $Var1 -ne "10" )
{
echo $Var1
$Var1 = $Var1+1
}
```

Traduction En Français : Tant que VAR1 est diffèrent de 10, j 'affiche le contenu de Var1 et après je lui ajoute 1. (Attention) soyez sur que VAR1, au départ, est bien un entier en dessous de 10 sinon le programme part dans les nuages.

##########----------LE DO WHILE----------##########

même principe que le While sauf que l'action est effectué avant la condition

```
[int]$Var1 = "0"
DO {
echo $Var1
$Var1 = $Var1+1
}
while ( $Var1 -ne "10" )
```

##########----------LE FOR----------##########

Toujours dans le même principe la boucle FOR

Voici comment elle fonctionne :

For (initial; condition ; incrément) {ACTION}

```
For ($Var1 = "0" ; $Var1 -ne "10" ; $Var1 = $Var1+1  ) { echo $Var1 }
```

##########----------LE FOREACH----------##########

Pour Chaque lignes dans la liste faire cette action

Foreach ($ligne in $liste) { ACTION }

Dans l 'exemple du dessous pour chaque services dans la liste , PowerShell affiche le nom du service

```
$MesServices = get-service
Foreach ($_ in $MesServices ) { echo $_.name }
```

un autre exemple

```
Get-Service | foreach {$_.status} | sort –Unique
```

ici je prend tous les services, je liste tous les statuts et je n'affiche pas les doublons

J'attire votre attention sur la Variable prédéfinis $_ en PowerShell

elle veut dire la "ligne en cours/ l’Objet en cours"

si on veut la valeur de l'attribut" nom" d'une ligne/Objet dans une liste/array , on écrira "$_.nom"

Mais pour le moment, passons aux exercices.

B)Pratiques

1) Écrire un script qui demande à l'utilisateur de taper un nombre entre 1 et 10 , tant qu'il ne tape pas le chiffre 7 le programme continu de lui demander de rentrer un nombre.

2) Écrire un script qui demande à l'utilisateur de rentrer toutes les notes d'un élève dans un tableau et que tant que l'utilisateur ne saisi pas le mot clef "FIN" le script continu de lui demander et d'ajouter des notes au tableau.

3) Écrire un script qui demande à l'utilisateur de saisir 2 nombre, puis affiche la sommes des deux nombre. Le programme doit pouvoir être arrêté par utilisateur , je vous laisse décider comment.

4) Lister tous les attributs de fichiers que l'on trouve dans le dossier "c:\Windows" et sans doublon.

7)Conditions

A)Théorie

##########----------Conditions----##########

##########----------Le IF , ELSE ----##########

if (condition) {action si condition = oui} else { Action si condition = non}

Si la condition est ok, alors je fais ceci , sinon je fais cela > voila comment résumer le si/sinon.

Un petit exemple

```
$MaVariable = Read-Host "Entrez un nombre entre 1 et 100 "
If ($MaVariable –eq 23 ) {echo "Bravo la bonne réponse est 23"}
else { echo "Dommage  $MaVariable n'est pas la bonne réponse "}
```

un autre exemple mais on va combiner avec un foreach

```
$MaListe = Get-Service
foreach ($_ in $MaListe)
{
    If ($_.status -eq "Running")  { echo "$($_.Name) est en statut en cours." }
    Else { echo "$($_.Name) n'est pas en fonction sur l'ordinateur." }
}
```

##########----------Le IF , ELSE ELSEIF -----##########

Une variante est le ELSEIF, voici un exemple. Il sera plus parlant que des explications

```
$saisie = Read-Host "entrez un nombre entre 1 et 100"
If ($saisie –eq 50 ){ echo "la valeur est 50"}
ELSEIF ($saisie –eq 0) { echo "la valeur est 0 tricheur ! on avait dit entre 1 et 100"}
ELSEIF ($saisie –eq 30) { echo "la valeur est  30"}
ELSE {echo " la valeur saisie n'est ni 0, ni 30, ni 50 !" }
```

##########----------Le Switch -----##########

Petit cousin du ELSEIF le SWITCH fonctionne sur le même principe

un petit exemple vaut bien un grand discours

```
$saisie = Read-Host "entrez un nombre entre 1 et 100"
Switch ($saisie )
{
 0 { echo "la valeur est 0 tricheur ! on avait dit entre 1 et 100"}
 30 { echo "la valeur est 30 "}
 50 { echo "la valeur est 50"}
DEFAULT {echo " la valeur saisie n'est ni 0, ni 30, ni 50 !" }
}
```

Passons aux exercices

B)Pratiques

1) Créez un script qui n'affiche que les services en cours dont le nom commence par un "s"

2) Créez un script qui liste tous ce que l'on trouve dans le dossier "c:\Windows" mais vous ne devez afficher que les fichiers log

3) Créez un script qui liste tous ce que l'on trouve dans le dossier "c:\Windows" mais vous ne devez afficher que les fichiers log qui commence par la lettre "w"

4) Créez un script qui liste tous ce que l'on trouve dans le dossier "c:\Windows" mais vous ne devez afficher que les fichiers log qui commence par la lettre "w" et dont l'attribut du fichier est "A"

5) Créez un script qui liste tous ce que l'on trouve dans le dossier "c:\Windows" mais vous ne devez afficher les fichiers ou dossier qui commence par la lettre "w" et dont l'attribut du fichier est "A" ou "D"

6) Créez un script qui demande à un utilisateur de saisir un nombre, puis un autre . Afficher ensuite la somme des deux nombres . Puis redemandez encore la saisie de deux nombres et affichez encore la somme. Tant que l'utilisateur ne saisi pas le mot Clef FIN le programme continu.

8)Fonctions

A)Théorie

##########---------Les Fonctions----##########

Pour cette dernière partie parlons Fonctions

une fonction est une suite d'instructions qui va être répété (souvent)

on doit appeler cette fonction grâce à un mot clef (son nom) et on peut lui passer des paramètres

En principe une fonction retourne une valeur

Créons notre première toute petite fonction.

Function MonNomDeFonction ([Type] $MonParametre ; [Type]$MonParametre) { Action }

```
function Fsalut ([string]$Prenom)
{
echo "Bonjour $Prenom "
# ou aussi
#Return  ,$Prenom
}
```

ma fonction est prête , je peux maintenant l'utiliser/l'appeler :

```
Fsalut -prenom "Jean"
```

Un autre exemple plus proche de ce qu'on trouve en pratique

Voici la création d'une boite à lettres, suivi d'une modification et pour finir on ajoutera la boite au lettre à un groupe de distribution

en pratique on a un fichier (csv) avec une liste d'utilisateurs qui contient , non, prénom, mot de passe , groupe, etc...

et on lui dit que pour chaque ligne du fichier csv on va appeler la fonction et créer la boite à lettres

voici à quoi pourrait ressembler cette fonction

```
function CreaMAilBox ([string] $UserNom ,[string] $UserPrenom ,[string]  $UserEmail , [string]  $UserMotDePasse, [string]  $UserPromotion,[string] $UserAlias) # Déclaration
```

```
du nomn et des paramètres possible pour la fonction

       {
         $MdpCrypte = ConvertTo-SecureString $UserMotDePasse -AsPlainText –force # 
commande pour crypter le mot de passe
         New-Mailbox -MicrosoftOnlineServicesID $UserEmail -Alias $UserNom -FirstName 
$UserNom  -LastName $UserPrenom -DisplayName $UserAlias -Password $MdpCrypte -Name 
$UserAlias # création de la Boite à lettres
         Set-Mailbox $UserEmail -CustomAttribute15 "2020-2021"   -CustomAttribute1 
$UserPromotion # modification de la Boite à lettres
         Add-DistributionGroupMember –identity $UserPromotion –member $UserEmail # Ajout 
de la BAL à un groupe
       }
```

puis voici son utilisation pour Mr Dupont Luc dans la promotion BTS SIO

```
  CreaMAilBox -UserNom Dupont -UserPrenom Luc -UserEmail L.dupont@yopmail.com -
UserMotDePasse AZERTY -UserPromotion BTS_SIO -UserAlias LDUPONT
```

on peut aussi appeler une fonction depuis une autre fonction

B)Pratiques

1) Créez une fonction avec comme paramètres possible le nom et le prénom. Lui construire un login de type : Première lettre du prénom, point, le nom complet , puis @jaimepowershell.yopmail . Affichez le résultat

Ex pour Mr Luc Dupont --------> L.Dupont@jaimepowershell.yopmail (CF : Question B4)

2) Créez une fonction avec comme paramètres possible le nom et le prénom. Lui construire un login de type : Première lettre du prénom, point, le nom complet , puis @jaimepowershell.yopmail

Ex pour Mr Luc Dupont --------> L.Dupont@jaimepowershell.yopmail (CF : Question B4)

Si le prénom de cette personne commence par un "L" ou un "E" alors vous devez mettre son login sous la forme Luc.Dupont@jaimepowershell.Com. Affichez le résultat

2) Créez une fonction avec comme paramètres possible le nom et le prénom. Lui construire un login de type : Première lettre du prénom, point, le nom complet , puis @jaimepowershell.yopmail

Ex pour Mr Luc Dupont --------> L.Dupont@jaimepowershell.yopmail (CF : Question B4)

Si le prénom de cette personne commence par un "L" ou un "E" alors vous devez mettre son login sous la forme Luc.Dupont@jaimepowershell.Com . Sauf si son prénom est "Seb" si c'est la cas alors vous devez juste afficher " Seb c'est bien".

3) Créez une fonction avec comme paramètres possible le nom et le prénom. Lui construire un login de type : Première lettre du prénom, point, le nom complet , puis @jaimepowershell.yopmail

Ex pour Mr Luc Dupont --------> L.Dupont@jaimepowershell.yopmail (CF : Question B4)

Si le prénom de cette personne commence par un "L" ou un "E" alors vous devez mettre son login sous la forme Luc.Dupont@jaimepowershell.Com . Sauf si son prénom est "Seb" si c'est la cas alors vous devez demander à l'utilisateur son année de naissance. Si son année est inférieure à 1980 alors afficher " Seb c'est bien" sinon afficher un login comme demandé question G1.

9) Notes et références

Windows-PowerShell-v1-et-2-Guide-de-reference-pour-l-administration-système de Robin LEMESLE, Arnaud PETITJEAN

http://fr.scribd.com/doc/51376567/Windows-PowerShell-v1-et-2-Guide-de-reference-pour-l-administration-systeme#scribd

https://technet.microsoft.com/fr-fr/library/hh847769.aspx