TP – Scripting PowerShell

CONSUEGRA Yanis

Table des matières

1. Les variables

1.1. Les variables utilisateurs

Q1 –

La commande Get-Variable en PowerShell permet d'obtenir la liste des variables définies dans la session. Cela inclut les variables système, les variables d'environnement, ainsi que les variables que vous avez définies ou celles fournies par les modules ou les scripts que vous avez chargés. Sans argument, la variable retourne toutes les variables disponibles de la session.

Q2 –

La commande Clear-variable supprime uniquement le contenu de la variable sans supprimer la variable elle-même. Alors que la commande Remove-variable va supprimer le contenu mais également la variable.

Q3 –

```
$test = 12;      #Int32
$test = "Word";  #String

$test = 12, "Word"; #Object[]

$test = Get-ChildItem C:\Windows;  #Object[]
```

Q4 –

[int]$nombre = 8; : $nombre est déclaré comme un entier et initialisé à la valeur 8. En PowerShell, la déclaration indique que la variable doit être de type entier.

$nombre = "12345"; : À ce stade, la valeur de $nombre est changée pour "12345", qui est une chaîne de caractères. PowerShell est un langage faiblement typé, donc il convertit automatiquement la chaîne en entier, car

$nombre a été déclaré comme un entier. La nouvelle valeur de $nombre sera 12345.

$nombre = "Hello"; : De même, cette ligne change la valeur de $nombre pour "Hello", qui n'est pas un entier valide. Cela signifie que PowerShell ne peut pas effectuer une conversion implicite vers un entier, car "Hello" n'est pas une valeur numérique. Par conséquent, $nombre reste une chaîne de caractères après cette ligne.

2. Les opérateurs

2.1. Les opérateurs de comparaisons génériques

Q5 –

```
"Test" -like "t*" ; #True : commence par t suivi de n'importe quelle suite de caractère et PS ne distingue pas la casse
"Test" -like "t?";  # False : La chaîne "Test" ne correspond pas à "t" suivi d'un seul caractère. Il y a plus d'un caractère après "t".
"Test" -like "t???";    # True : La chaîne "Test" commence par "t" et est suivie de trois caractères. PS ne distingue pas la casse.
```

2.2. Les opérateurs de comparaison d'expressions régulières

Q6 –

```
"Powershell" -match "power[sol]hell";   #True : un caractère parmi [sol] entre power et hell
'powershell' -match  'powershel[a-k]';  # False : La chaîne "powershell" contient "powershel" suivi d'un caractère compris entre "a" et "k". Cependant, il n'y a pas de caractère « l » compris en a et k dans l'alphabet
'powershell' -match  'powershel[a-z]';  # False : La chaîne "powershell" contient "powershel" suivi d'un caractère compris entre "a" et "z". L est bien comris entre a et z dans l'alphabet
```

2.3. Les opérateurs logiques

Q7 –

```
(5 -eq 5) -and (8 -eq 9) ;#False, car 5 est bien égal à 5, mais 8 n'est pas égal à 9.
(5 -eq 5) -or (8 -eq 9);#True, un des 2 test fonctionne, le programme retourne donc true
```

```
-not (8 -eq 9); #True, 8 est bien pas égale à 9
!(8 -eq 9); #True, il s'agit du contraire
```

2.4. Les opérateurs de redirections

Q8 –

Testez les exemples suivants et expliquez les résultats en commentaire dans le script

```
Get-Process > .\process.txt #Fichier texte créer dans le répertoire courant du nom de process.txt qui contient le résultat de la commande.

Get-ChildItem .\Unrépertoire  # La commande Get-ChildItem récupère une liste des éléments dans le répertoire spécifié. Cependant, si le répertoire spécifié n'existe pas, elle n'affichera rien à l'écran car aucun élément ne sera trouver.

Get-ChildItem c:\temp\RepInexistant 2> .\err.txt    # La commande Get-ChildItem essaie d'obtenir une liste des éléments dans le répertoire "c:\temp\RepInexistant". Mais ce répertoire n'existe pas, donc une erreur est affiché. L'opérateur "2>" redirige les messages d'erreur vers un fichier nommé "err.txt ». Le fichier "err.txt" contiendra le message d'erreur généré par PowerShell.
```

2.5. Opérateurs de fractionnement et de concaténation

Q9 –

```
$nom = "Rabbit";
$prenom = "Roger";
#Complétez l'instruction suivante pour que la variable $nomPrenom prenne la valeur de la concaténation de [nom] + espace + [prenom]

$nomPrenom = $nom + «  » + $prenom

#En utilisant la méthode split, faites en sorte que la variable $nomPrenom inverse la concaténation (devient [prenom] + espace + [nom])


$nomPrenom = $prenomNom = ($nomPrenom.Split(' '))[1] + ' ' + ($nomPrenom.Split(' '))[0];
```

Lecture d’un script :

```
write-Output "*****************************" "* Plus ou moins en PScript *"
"*****************************" "" #Affiche un titre
write-Output "Le nombre Mystère est compris entre 0 et 100" ## Indique la
plage du nombre mystère
$variable=Read-Host "Quel est le nombre mystère ?" #Demande à l'utilisateur de
deviner le nombre
$Random=Get-Random --minimum 0 -maximum 101 #Génère un nombre aléatoire entre
0 et 100
$tentative=<1> #Initialise le compteur de tentatives
While ($variable -ne $random) { #Commentez
if ($variable <> $Random) {Write-Output "C'est plus!"} #Affiche "C'est plus"
si la valeur entrée est supérieure au nombre
if ($variable <> $Random) {Write-Output "C'est moins"} #Affiche "C'est moins"
si la valeur entrée est supérieure au nombre
$variable=Read-Host "Quel est le nombre mystère ?" #Demande à l'utilisateur de
deviner encore
$tentative++    #incrémentation du compteur de tentative
}
Write-OutPut "" #Affiche une ligne vide
Write-Output <"Vous avez gagne, le nombre mystere était bien: $Random. Vous
avez reussi en $tentative tentatives !"> #Remplacez les chevrons de sorte que
la sortie affiche "Vous avez gagné, le nombre mystère était bien: "<Le nombre
mystère>" Vous avez réussi en <X tentatives> coups !"
Read-Host
```

SCRIPTING AVANCE

1. Les tableaux et les pipelines

1.1 Les tableaux

Q1-

```
$tabVide = @()      # Initialise un tableau vide
$tab = 1,5,9,10,6   # Initialise un tableau avec les valeurs 1,5,9,10,6
$tab                # Affiche tous les éléments du tableaux
$tab[0]             # Affiche la valeur du premier index du tableau
$tab[5]             # Affiche la valeur à l'emplacement 5 du tableau (ici
rien)
$tab=1..20          # Remplace ou créer un tableau avec les valeurs entre 1 et
20
$tab+="Coucou"      # Ajoute la chaîne de caractère « Coucou » à la fin du
tableau
$tab[5]=555         # Remplace la valeur à l'emplacement 5 par la valeur 555
$tab[5,3,0,5]       # Affiche les différentes valeurs aux différents
emplacements indiqué dans la variable
$tab[14..20]        # Affiche les valeurs aux emplacement 14 et 20
$tab[-1]            # Affiche la toute dernière valeur du tableau
$tab.count          # Compte et affiche le nombre d'élément du tableau
```

1.2 Les pipelines

Q2-

Get-ChildItem C:\Windows:

Cette commande récupère tous les éléments du répertoire C:\Windows.

ForEach-Object {$ .Get extension().toLower()} :

Pour chaque élément retourné par la commande, cette partie de la pipeline PowerShell utilise la méthode Get_extension() pour récupérer l'extension du fichier, puis applique la toLower() pour convertir l'extension en minuscules.

Sort-Object :

Toutes les extensions de fichiers récupérées sont ensuite triées par ordre alphabétique.

Get-Unique :

Cette commande élimine les doublons, ne laissant que les extensions de fichiers uniques.

Out-File -FilePath extensions.txt :

Cette commande écrit les extensions de fichiers uniques dans un fichier nommé "extensions.txt". Les extensions sont écrites une par ligne dans ce fichier.

La commande va donc prendre les extensions de tous les fichiers présents dans C:\Windows et les convertit en minuscules, les trie par ordre alphabétique, supprime ceux qui sont en double et écrit les extensions uniques dans un fichier nommé "extensions.txt".

1.3 Filtre Where-Object

Q3-

```
Get-Service | Where-Object {$_.Status -eq 'Stopped'}
```

⇨ Cette commande va récupérer tous les services du système et filtrer ceux dont le statut est en « stopped » pour ensuite les afficher.

Q4-

```
Get-ChildItem -Path C:\Windows -File | Where-Object {$_.Length -gt 500}
```

```
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        08/07/2023     17:59         102400 bfsvc.exe
-a--s-        31/01/2024     14:37          67584 bootstat.dat
-a----        07/05/2022     07:21          24895 CloudEdition.xml
-a----        31/01/2024     13:42           2327 DtcInstall.log
-a----        07/05/2022     07:21          24895 Education.xml
-a----        07/05/2022     07:21          24935 Enterprise.xml
-a----        08/07/2023     17:59        5071400 explorer.exe
-a----        08/07/2023     18:01        1089536 HelpPane.exe
-a----        07/05/2022     07:20          36864 hh.exe
-a----        07/05/2022     07:21          24895 IoTEnterprise.xml
```

2. Les boucles

2.1 While et Do-While

Q5-

La Différence entre une boucle While et Do While est que pour entrer dans la boucle de l'instruction While il faut que la condition soit initialement vrai si le programme veut continuer (c'est pour cela que rien n'est affiché dans l'exemple) alors que pour la commande Do While le code est testé au minimum 1 fois avant que la condition soit évalué (d'où un affichage lors de l'exécution du code).

Q6-

Ce script crée une boucle infinie qui alterne entre deux caractères, il affiche un message entre eux à chaque itération. Il utilise une variable $tour pour déterminer quel caractère utiliser à chaque itération en changeant entre deux options stockées dans les variables $charGauche et $charDroite. Le message "Je n'arrive pas à m'arrêter" est affiché entre les caractères spéciaux, et la boucle continue à s'exécuter à l'infini.

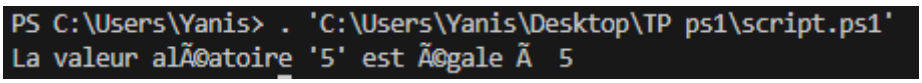
Q7-

La condition ne la boucle n'est jamais rencontré, par conséquent le programme tourne en boucle.

3. Les structures conditionnelles : If, Else, ElseIf, Switch

3.1 If, Else, ElseIf

Q8-



Q9-

```
$lettre1 = Read-Host "Entrez la première lettre"
$lettre2 = Read-Host "Entrez la deuxième lettre"

if ([char]::ToUpper($lettre1) -lt [char]::ToUpper($lettre2)) {
    Write-Output "$lettre1 est avant $lettre2 dans l'alphabet"
}
elseif ([char]::ToUpper($lettre1) -eq [char]::ToUpper($lettre2)) {
    Write-Output "$lettre1 est égale à $lettre2 dans l'alphabet"
}
else {
    Write-Output "$lettre1 est après $lettre2 dans l'alphabet"
}
```

Q10- On peut utiliser la commande « default » afin d'afficher un message et d'ajouter l'instruction « break » afin de sortir du programme.

On peut améliorer ce code en mettant une boucle et une condition afin de faire en sorte que si l'utilisateur

Q11-

```
if ($null -eq (Get-ChildItem "whatever.txt")) {
    New-Item "whatever.txt"
    "Whatever you do `rWhatever you say `rYeah I know it's alright" | Out-
File -FilePath "whatever.txt" -append
}

$lines = Get-Content "whatever.txt"
for ($i = 0; $i -lt $lines.Count; $i++) {
    Write-Host ("({0}) {1}" -f ($i + 1), $lines[$i].ToUpper())
}
```

Powershell et l'AD

En ligne de commande

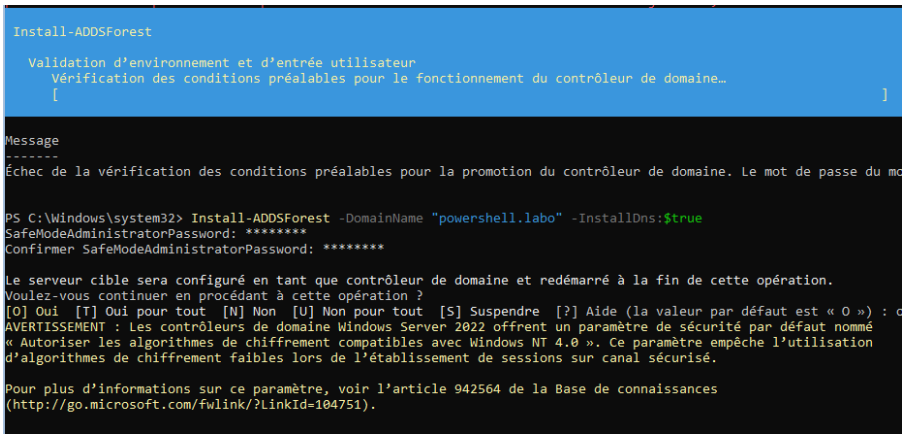
Q2.

```
PS C:\Windows\system32> New-NetIPAddress -InterfaceIndex 6 -IPAddress 192.168.130.20 -PrefixLength 24 -DefaultGatew
2.168.130.254


IPAddress         : 192.168.130.20
InterfaceIndex    : 6
InterfaceAlias    : Ethernet
AddressFamily     : IPv4
Type              : Unicast
PrefixLength      : 24
PrefixOrigin      : Manual
SuffixOrigin      : Manual
AddressState      : Tentative
ValidLifetime     : Infinite ([TimeSpan]::MaxValue)
PreferredLifetime : Infinite ([TimeSpan]::MaxValue)
SkipAsSource      : False
PolicyStore       : ActiveStore
```

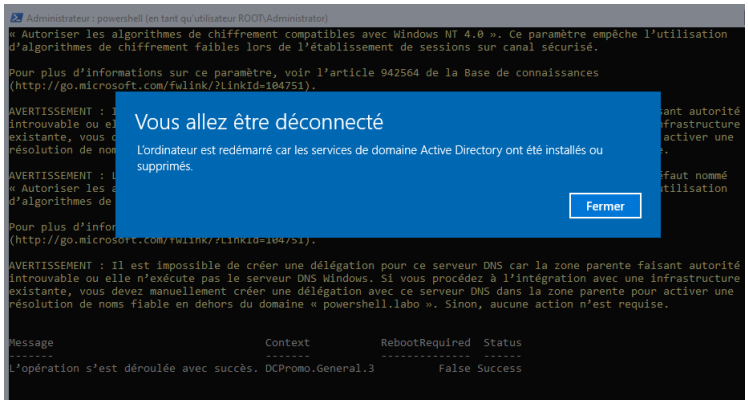
Ajouter le rôle ADDS

```
PS C:\Windows\system32> Install-WindowsFeature AD-Domain-services -IncludeManagementTools

Success Restart Needed Exit Code      Feature Result
------- -------------- ---------      --------------
True    No             Success        {Services AD DS, Gestion de stratégie de g...
```

Création forêt avec DNS, notez la commande



⇨ Après la commande, le service va redémarrer :



Modification de la VM Windows 11

Définition de l'adresse IP

New-NetIPAddress -InterfaceAlias "Ethernet" -IPAddress 192.168.130.10 -PrefixLength 24 -DefaultGateway 192.168.56.254

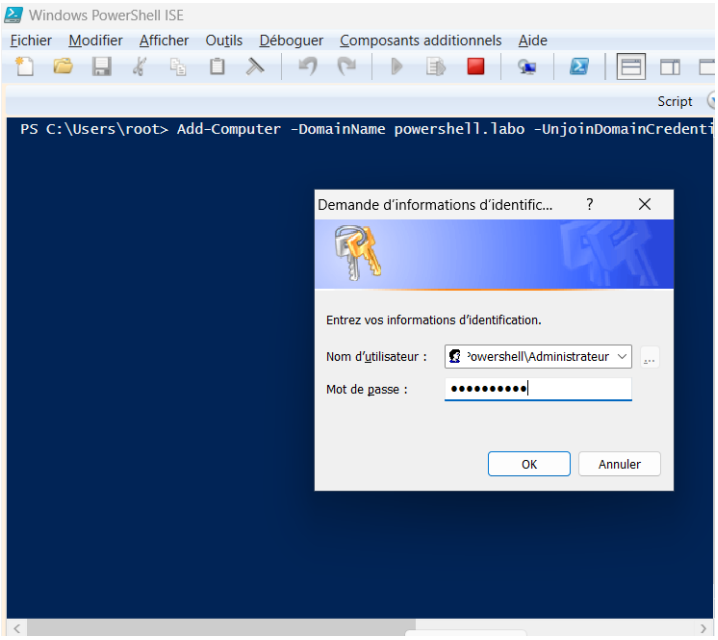
Définition du serveur DNS

Set-DnsClientServerAddress -InterfaceAlias "Ethernet" -ServerAddresses 192.168.56.20

Test de ping avec le nom de la machine serveur

```
C:\Users\root>ping YCO-PWS-DC

Envoi d'une requête 'ping' sur YCO-PWS-DC.local [fe80::7054:57a3:974e:62e9%5] avec
 32 octets de données :
Réponse de fe80::7054:57a3:974e:62e9%5 : temps<1ms
Réponse de fe80::7054:57a3:974e:62e9%5 : temps<1ms
Réponse de fe80::7054:57a3:974e:62e9%5 : temps<1ms
Réponse de fe80::7054:57a3:974e:62e9%5 : temps<1ms

Statistiques Ping pour fe80::7054:57a3:974e:62e9%5:
    Paquets : envoyés = 4, reçus = 4, perdus = 0 (perte 0%),
Durée approximative des boucles en millisecondes :
    Minimum = 0ms, Maximum = 0ms, Moyenne = 0ms

C:\Users\root>
```

Joindre le domaine powershell.labo



Après avoir ajouter le poste au domaine un redémarrage de la session est nécessaire