Exploitation des systèmes

PowerShell : les premiers scripts

Table des matières

1 Introduction

Ce 2° TP va nous permettre d'introduire les bases du scripting. A l'issue de ce TP, vous saurez :

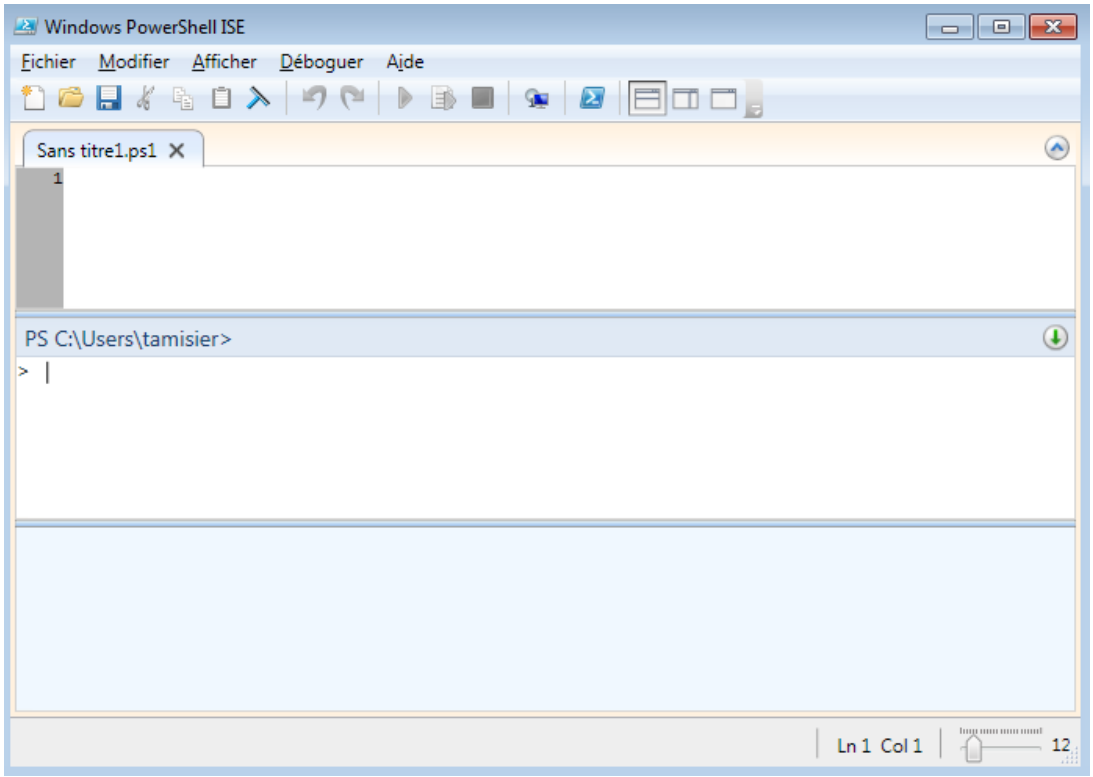
- modifier les polices d'exécution pour exécuter des scripts
- utiliser PowerShell-ise :
  - Charger un script
  - exécuter un script
  - déchiffrer un script
  - Mettre en place des filtres
  - Gérer des dates.

2 Bases du Scripting

2.1 Powershell ise

Aller dans « démarrer > tous les programmes > accessoires > powershell « et sélectionnez powershell ISE.

Vous obtenez l'écran suivant :



1. Partie Haut : Contient l'ensemble des scripts que vous êtes en train de modifier. Il y a un onglet par script.

2. Partie Centrale : C'est la fenêtre dans laquelle vous pouvez saisir des commandes powershell de manière unitaire. Vous pouvez aussi provoquer, dans cette fenêtre, le démarrage du script. Elle se comporte un peu comme la fenêtre de commandes PowerShell que nous avons déjà utilisée.

3. Partie Bas : Contient les résultats des commandes exécutées via la fenêtre 2.

2.2 Modifier la politique d'exécution

Par défaut, l'exécution de scripts est interdite dans PowerShell. Pour vous en convaincre, saissez dans la fenêtre centrale de powershell ISE la commande suivante :

```
get-executionPolicy
```

Donner le résultat que vous obtenez :

Il faut modifier la politique d'exécution. Dans la fenêtre 2 de powershell ise. Saisissez

```
set-executionPolicy  -executionPolicy remoteSigned  -scope localMachine
```

PowerShell va vous demander de valider ce choix.

Décrivez quels sont vos nouveaux droits avec cette politique d'exécution. Quels sont les scripts que vous pouvez utiliser ?

.

3 Déchiffrer le premier script

3.1 Texte du script

Voici le premier script que nous allons étudier. Il permet de chercher un fichier dans un dossier donné sur le disque dur. Le fichier et le dossier seront désignés en tant que paramètres du script.

```
1  <#--------------------------------------------------------------
2  Apprentissage PowerShell - Script n° 1
3  Fonction : Ce script cherche un fichier dans un dossier donné
4  Auteur – 10/03/2015
5  ------------------------------------------------------------------#>
6  $cherche = $args[0]
7  $dossier = $args[1]
8  echo "Recherche du fichier $cherche dans le dossier $dossier"
9  Get-ChildItem -Path $dossier -Recurse |
10 ? {$_.Name -eq $cherche}
```

3.2 Exécuter ce script

Copier / Coller ce script dans la fenêtre 1 de powershell ISE.

Enregistrez le sur votre disque dur sous le nom chercheFichier.ps1 Notez bien le dossier dans lequel vous l'enregistrez (mettez-le dans un dossier spécial qui ne vous servira qu'à ça).

Dans la fenêtre n° 2, au moyen de la commande CD (ou get-location en PWS) positionnez-vous sur le dossier où vous avez stocké le script.

Dans la fenêtre n° 2, saisissez

```
./chercheFichier.ps1 hosts c:\windows
```

Expliquez ce que fait la commande que vous avez tapée.

3.3 Analyser ce script

Comment procède ce script ?

Le cœur de la recherche se trouve dans les lignes 9 et 10 de ce script.

L'instruction 9 liste l'ensemble des fichiers contenus dans le répertoire sélectionné et tous ces sous répertoires.

L'instruction 10 récupère les données précédentes et applique un filtre pour ne conserver que les fichiers dont le nom ($_.Name) correspond à celui passé en paramètre.

Qu'est ce qui nous permet d'affirmer que l'instruction 10 récupère les données fournies par l'instruction 9 ?

Passer des arguments au script

Considérez les lignes 6 et 7 de ce script.

La ligne 6 charge la variable $cherche avec le contenu de la variable $arg[0], la ligne 7 met dans $dossier le contenu de $arg[1]

$arg est une variable prédéfinie dans powerShell. Lorsqu'on lance l'exécution du script, on saisit :

`./chercheFichier.ps1`	`Hosts`	`C:\windows`
Nom du script	1° Paramètres du script.	2° paramètre du script
	`$arg[0]`	`$arg[1]`

3.4 Améliorer le script

Ce script fonctionne, mais le résultat obtenu est parfaitement hideux. Pour améliorer cela, nous allons procéder à deux manipulations : la première va nous permettre d'éviter l'affichage des messages d'erreur signalant que des dossiers sont inaccessibles, la deuxième va nous permettre de n'afficher que le nom du dossier qui contient le fichier que nous cherchons.

Éliminer l'affichage des erreurs

Pour ne plus avoir de messages d'erreur, il suffit de rajouter à l'instruction n° 9 le paramètre :

```
-ErrorAction SilentlyContinue
```

Réalisez cette modification, sauvegardez le script et réexécutez le. Donnez ici le résultat que vous obtenez :

Afficher uniquement le nom du dossier

Pour cela, nous allons utiliser le bloc « forEach ». Cette instruction est utilisable, comme where-object, uniquement à la suite d'un tube. Elle permet de réaliser un traitement sur chaque ligne sortie par l'instruction à laquelle elle est liée par le tube.

Nous allons donc rajouter un « pipe » à la fin de l'instruction 9 pour signaler que le résultat doit être envoyé à une autre instruction. Nous allons rajouter l'instruction suivante :

```
forEach-Object {
    echo ("le fichier $cherche est dans "+$_.DirectoryName)
}
```

Réalisez cette modification, sauvegardez le script et réexécutez le. Donnez ici le résultat que vous obtenez :

3.5 Résumé

Nous avons vu une technique de base nous permettant de faire très simplement des actions sur un grand nombre d'objets :

- where-object permet de filtrer uniquement les objets sur lesquels nous souhaitons avoir une action.
- ForEach permet de mettre en place une action spécifique sur chaque objet.

4 Exercice

Rajoutez les instructions permettant de compter le nombre de dossiers qui contient ce fichier.

Vous devez obtenir, après cette modification, le résultat suivant :

```
PS C:\Users\manirac\tp_pws> .\chercheFichier.ps1 hosts 'C:\Windows'
on cherche si hosts existe dans C:\Windows
le fichier hosts est dans C:\Windows\System32\drivers\etc
le fichier hosts est dans C:\Windows\winsxs\amd64_microsoft-windows-w..nfrastructure-other_31b
f3856ad364e35_6.1.7600.16385_none_6079f415110c0210
le fichier hosts est présent dans 2 dossiers
```

Donnez le script final.

5 Script n° 2

5.1 Analyse du script

1	<#----------
2	Apprentissage PowerShell - Script n° 2
3	Auteur – 10/03/2015
4	----------#>
5	$dossier = $args[0]
6	echo "calcul en cours sur $dossier"
7	Get-ChildItem -Path $dossier -Recurse -Force |
8	Measure-Object -property Length -Sum | `
9	ForEach-Object {
10	$total = $_.sum / 1MB
11	write-host -foregroundColor yellow ("le dossier "+$dossier+" contient {0:#,##0.0} MB" -f $total)
12	}

NB : la commande measure-object permet de calculer le total (ou la somme) d'une des propriétés d'un objet. Ici, elle permet de calculer le total de la propriété « Lentgh » qui contient le volume des données de chaque fichier en octets.

Indiquez ce que fait ce script.

Donnez le numéro des instructions qui sont exécutées dans ForEach ? Qu'est-ce qui vous amène à cette

conclusion ?

Dans quelle unité est donné le résultat ? Quelle est l'instruction qui permet de le convertir dans cette unité ?

A quoi sert le paramètre foregroundColor dans l'instruction write-host ?

5.2 Modifications du script

Eviter l'affichage des messages d'erreur

Modifier le script pour que les messages d'erreur ne s'affichent plus.

Donner le nouveau script en indiquant la modification que vous avez effectuée.

Calculer le volume des données des fichiers de plus de 1MO

On se propose de modifier ce script pour ne traiter que les fichiers qui ont un volume > 1MO (1 048 576 octets). Le volume d'un fichier est donné par son attribut length. *Apportez la modification nécessaire, testez et recopiez cette modification ci-dessous*

6 Script n° 3

6.1 Conditions et boucles

Avec powerShell, on peut aussi faire des tests et des boucles. Les conditions permettent d'exécuter une partie des instructions d'un script ou une autre partie en fonction d'une condition donnée. Une boucle permet de répéter plusieurs fois une suite d'instructions.

Copier/coller le script suivant :

1	<#----------
2	Apprentissage PowerShell - Script n° 3
3	Auteur – 10/03/2015
4	----------#>
5	$invite = "saisissez une couleur :"

6	`$couleur = Read-Host $invite`
7	`Write-Host -ForegroundColor $couleur ("vous avez demandé à écrire en "+$couleur)`

Enregistrez le sous le nom listeCouleurs.ps1

Testez le avec la couleur « yellow ». Quel est le résultat ?

Testez le avec la couleur « jaune ». Quel est le résultat ?

6.2 Condition

Ce script ne fonctionne qu'avec une liste bien précise de couleurs. Pour améliorer le fonctionnement de ce script, nous allons le modifier pour éviter ce message d'erreur à chaque fois que l'utilisateur saisit une mauvaise couleur.

Tout d'abord, nous allons mettre la liste des couleurs en place. Après la ligne 4 du script, nous allons saisir l'instruction suivante

```
$listeCouleurs =
@("Black","DarkBlue","DarkGreen","DarkCyan","DarkRed","DarkMagenta","DarkYellow"
,"Gray","DarkGray","Blue","Green","Cyan","Red","Magenta","Yellow","White")
```

Le caractère « @ » spécifie que ce qui suit est une liste d'éléments tous semblables.

Ensuite, nous allons mettre en place un système afin de d'assurer que la couleur saisie est bien dans la liste. Nous allons utiliser un filtre, que nous allons placer après la ligne contenant l'instruction ***$couleur = Read-Host $invite*** :

```
$z = $listeCouleurs | where-object {$_ -eq $couleur}
```

A l'issue de cette instruction, $z contiendra le nom de la couleur si le filtre a réussi ou rien si le filtre a échoué, c'est à dire si la couleur choisie n'existe pas dans la liste. Notez l'opérateur de comparaison « **`-match`** » qui permet d'éviter les majuscules et les minuscules.

Nous allons maintenant introduire une condition et saisir les instructions suivantes, en remplacement de la dernière instruction, qui contient **white-host** :

```
if ($z -ne $null)  {
    Write-Host -ForegroundColor $couleur ("vous avez demandé à écrire en
"+$couleur)
}
else {
    write-host ("la couleur "+$couleur+" n existe pas.")
}
```

Expliquez en langage naturel ce que fait cette suite d'instructions

Il ne vous reste plus qu'à enregistrer le script modifié et à vérifier son bon fonctionnement en lançant son exécution.

6.3 Boucle

Le problème, dans ce script, est que sitôt qu'une couleur a été saisie, il faut le relancer. Pour éviter cela, nous allons mettre en place un système pour qu'il se répète jusqu'à ce que l'opérateur décide d'arrêter. Pour cela nous allons utiliser une boucle « while ».

Introduisez, après la ligne qui déclare la liste des couleurs, l'instruction suivante :

```
while ($couleur -ne 'stop') {
```

puis saisissez, tout à fait à la fin du script, l'instruction suivante :

```
}
```

Expliquez, en langage naturel, ce que vous avez fait

Que doit saisir l'opérateur pour arrêter le déroulement du script ?

Quelle est la valeur initiale de la variable $couleur ?

Enregistrez le script et contrôlez qu'il fonctionne bien.