

S3T4 - POWERSHELL 2

Vous rédigerez un tableau récapitulatif des VM installées, avec leurs logiciels

S3T4 votre nom.docx

Tout fichier ne correspondant pas à cette demande sera ignoré !

1 Introduction

Au cours de ce TP, nous allons tester les premiers scripts Powershell avec l'interface de développement ISE

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.

Ce TP contient des questions, **vous écrirez vos réponses dans les cadres jaunes prévus à cet effet.** Prêt ?

Oh OUI !

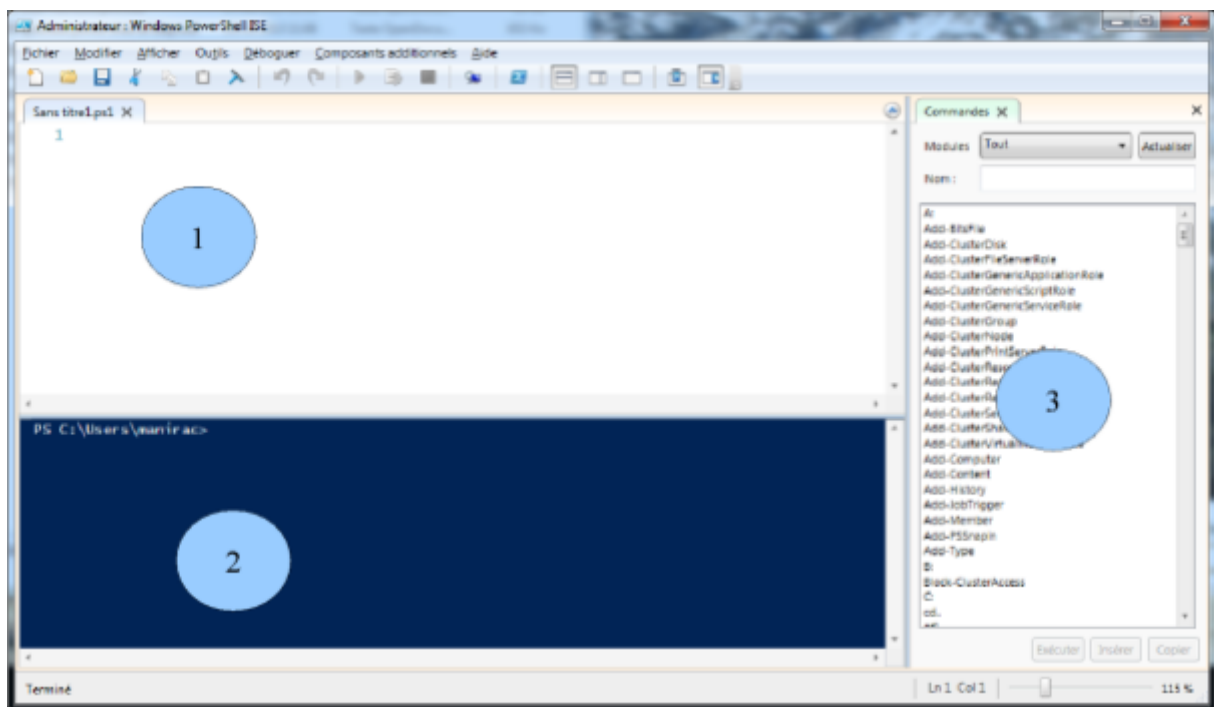
Alors c'est parti ...

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

(2) C'est la fenêtre dans laquelle vous pouvez frapper 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) Contient la liste des commandes et une aide pour chacune d'entre elles. Vous pouvez sélectionner le « module » puis cliquer sur une commande pour obtenir des explications sur les paramètres.

2.2 Modifier la politique d'exécution

Par défaut, l'exécution de scripts est interdite dans PowerShell. Pour vous en convaincre, frappez dans la fenêtre 2 de powershell ISE la commande suivante :

```
get-executionPolicy
```

Donner le résultat que vous obtenez :

```
Restricted
```

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

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

PowerShell va vous demander de valider ce choix.

TAF : 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 calculer le volume occupé par un dossier sur le disque dur.

```
<#-----
Apprentissage PowerShell - Script n° 1
Fonction : Ce script cherche un fichier dans un dossier donné
Auteur BD - 17/12/2013
-----#>

$cherche = $args[0]
$dossier = $args[1]
echo "Recherche du fichier $cherche dans le dossier $dossier"
Get-ChildItem -Path $dossier -Recurse |
? {$_.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, frappez (remplacez {votre nom d'utilisateur} par votre nom sans mettre les accolades)

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

Expliquez ce que fait la commande que vous avez frappée

```
.
```

Mettez ici une copie de l'écran de résultat que vous obtenez (dans la fenêtre n° 2)

```
.
```

3.3 Analyser ce script

3.3.1 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 ?

3.3.2 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 frappe :

./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.

3.4.1 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 :

3.4.2 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 ajouter un « pipe » à la fin de l'instruction 9 pour signaler que le résultat doit être envoyé à une autre instruction. Ajouter l'instruction suivante après le Pipe :

```
foreachObject {  
    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 contiennent 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 ici le script final :

5 Script n° 2

5.1 Analyse du script

```
<#-----
Apprentissage PowerShell - Script n° 2
Auteur BD - 17/12/2013
-----#>

$dossier = $args[0]
echo "calcul en cours sur $dossier"
Get-ChildItem -Path $dossier -Recurse -Force | ? {$_.PsisContainer -eq 0}
Measure-Object -property Length -Sum | `
ForEach-Object {
    $total = $_.sum / 1MB
    write-host -foregroundColor yellow ("le dossier "+$dossier+" contient {0:#,##0.0}
    MB" -f $total)
}
```

Il manquait un point derrière le \$_ dans la ligne get-childitem ...

NB : 1 : ligne 7, le "?" est le raccourci de la commande **where-object**.

2 : **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 « Length » qui contient le volume des données de chaque fichier en octets.

Indiquez ce que fait ce script. Donnez un aperçu du résultat qu'il fournit.

Expliquez l'instruction « where-object {\$_.PsisContainer -eq 0} »

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 ?

A l'aide d'une recherche Internet sur Google, expliquez l'expression « {0:#,##0.0} MB » dans l'instruction write-host »

<http://technet.microsoft.com/en-us/library/ee692795.aspx>

5.2 Modifications du script

5.2.1 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.

5.2.2 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 :

```
<#-----  
Apprentissage PowerShell - Script n° 3  
Auteur BD - 18/12/2013  
-----#>  
  
$invite = "saisissez une couleur :"  
$couleur = Read-Host $invite  
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 frappe une mauvaise couleur.

Tout d'abord, nous allons mettre la liste des couleurs en place. Après la ligne 4 du script, nous allons frapper 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 et la variable est un tableau avec un indice numérique.

Ensuite, nous allons mettre en place un système afin de d'assurer que la couleur frappée est bien dans la liste. Nous allons utiliser un filtre, que nous allons placer après la dernière ligne :

```
$z = $listeCouleurs | where-object {$_ -match $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 frapper les instructions suivantes, en remplacement de l'instruction n° 7 :

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

Expliquer 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 frappée, 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 n° 5 (celle qui déclare la liste des couleurs) l'instruction suivante :

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

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

```
}
```

Expliquer, en langage naturel, ce que vous avez fait

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

Enregistrez le script et contrôlez qu'il fonctionne bien.

7 Lecture et exploitation d'un fichier csv

7.1 Introduction

On a récupéré la liste des employés de la Sodécaf dans un fichier CSV. Un fichier CSV se présente comme une liste d'éléments séparés par un caractère qui peut être un point virgule (;), une virgule (,) ou tout autre caractère. Le plus utilisé est le point-virgule. Par exemple :

```
1 id;username;email;firstname;lastname;Phone1;phone2;Function;Agency;country
2 5;romain.cassignol;romain.cassignol@sodecaf.local;Romain;CASSIGNOL;05.63.68.99.11;;COMPTABLE;MONTAUBAN;FR
3 4;flore.diaz;flore.diaz@sodecaf.local;flore;DIAZ;05.63.68.91.89;;ACCUEIL;MONTAUBAN;FR
4 5;clément.ferrier;clément.ferrier@sodecaf.local;Clément;FERRIER;05.63.68.91.24;;COMPTABLE;MONTAUBAN;FR
5 6;samy.parisse;samy.parisse@sodecaf.local;Samy;PARISSE;05.63.68.95.24;;INFORMATIQUE;MONTAUBAN;FR
6 7;thibaud.pellissier;thibaud.pellissier@sodecaf.local;Thibaud;PELLISSIER;05.63.68.92.71;;COMPTABLE;MONTAUBAN;FR
```

Chaque ligne contient donc un ensemble d'éléments qui ont la même signification. C'est souvent sur la première ligne du fichier (mais ce n'est pas toujours le cas) qu'on trouve la signification de chacun des éléments constituant une ligne. Par exemple :

Les éléments constituant chaque ligne sont souvent appelés des « champs » ou encore des « zones ».

Vous pouvez facilement visualiser un fichier .csv avec un utilitaire de type notePad ou notePad++.

Nous allons créer un script qui va nous permettre de faire plusieurs traitements sur ce fichier. Nous allons successivement :

Faire une liste colorisée du fichier suivant la fonction de la personne.

Créer un dossier par agence, puis un sous-dossier par personne. Le nom du sous dossier contiendra le préfixe fic_ suivi du trigramme de la personne (1^{er} lettre du prénom, 1^{er} lettre du nom et dernière lettre du nom).

Pour commencer, créez un dossier tp_csv. Copiez le fichier « S3T4 utilisateurs sodecaf.csv » dedans. Ouvrez powershell.ise et positionnez vous dans ce dossier.

7.2 Lecture simple du fichier CSV

Pour ouvrir le fichier, nous allons utiliser la commande import-csv ou ipcsv en abrégé :

```
ipcsv ".\S3T4 utilisateurs sodecaf.csv" -Delimiter ";"
```

Comment apparaît le contenu du fichier CSV ?

Frappez maintenant :

```
ipcsv ".\S3T4 utilisateurs sodecaf.csv" -Delimiter ";" | format-table
```

Quelle est la différence avec l'instruction précédente ?

Nous allons maintenant placer un filtre sur le fichier .csv. Frappez l'instruction suivante

```
ipcsv ".\S3T4 utilisateurs sodecaf.csv" -Delimiter ";" | where {$_.function -match "informatique"} | format-table
```

Que contient maintenant la liste qui vous apparaît à l'écran ?

Nous voulons maintenant uniquement faire apparaître le nom, le prénom, le trigramme et le n° de téléphone des employés. Frappez l'instruction suivante

```
ipcsv ".\S3T4 utilisateurs sodecaf.csv" -Delimiter ";" | foreach {
    $triGramme=$_.firstname.substring(0,1)+$.lastname.substring(0,1)
    $triGramme = $triGramme + $.lastname.substring($_.lastname.length-1,1)
    $triGramme = $triGramme.ToUpper()
    Write-Host ($_.firstname+" "+$.lastname+" ("+$triGramme+") "+$.phone1)
}
```

Expliquez comment est constitué le trigramme de chaque personne

Vous savez maintenant :

- lire le contenu du fichier csv.
- Filtrer des lignes
- faire un traitement sur chacune des lignes du fichier.

7.3 Exercice 1

Faites un script qui lit ce fichier et affiche le prénom, le nom, le trigramme et le téléphone avec une couleur différente selon le service (champ « fonction »).

- Cyan pour l'informatique
- rouge pour la direction (conseil)
- jaune pour les comptables
- bleu pour le social
- blanc pour le juridique.

Mettez ici le contenu du script que vous avez réalisé. Montrez les résultats obtenus.

7.4 Exercice 2

Faites un script qui lit le fichier des utilisateurs de la SODECAF et qui crée un répertoire pour chaque agence, puis un dossier pour chaque personne à l'intérieur du répertoire de l'agence. Le répertoire de l'agence aura pour nom, le nom de l'agence. Le répertoire de chaque personne sera appelé « **fic_** » suivi du trigramme de la personne. Par exemple, pour Bernard Ducasse : **fic_bde**.

Pour créer un dossier, vous pouvez utiliser soit MKDIR soit NEW-ITEM.

Mettez ici le contenu du script que vous avez réalisé. Montrez les résultats obtenus.