POWERSHELL 1

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

S3T3 votre nom.docx

Tout fichier ne correspondant pas à cette demande sera ignoré !

1 Introduction

Au cours de ce TP, nous allons tester les premières commandes Powershell

Les commandes de PWS sont appelées des **cmdlets** (pour command-applets). Elles sont pour la plupart d'entre elles constituées de la manière suivante : un verbe, un tiret, un nom : **verbe-nom**. Par exemple

```
get-command
```

Le verbe indique l'action que l'on va appliquer sur le nom. Il y a toute une série de verbes génériques : Get,Add, Remove, Set, etc... Les noms constituant les commandes sont toujours au singulier. C'est aussi vrai pour les options des commandes (on parle de paramètres en PWS et pas d'options).

On peut obtenir de l'aide en permanence sur un cmdlet. Il suffit de frapper :

```
get-help get-command -detailed | more
ou encore  help get-command
```

2 Tester les commandes proposées et répondre aux questions

Pour utiliser PowerShell, il faut lancer l'interpréteur de commandes. Vous pouvez le trouver dans le menu démarrer, dans les accessoires.

Sélectionner Windows PowerShell et lancez l'interpréteur.

*[OPTIONNEL] Vous ne pourrez peut-être pas faire toutes les commandes sur la machine physique sans être en mode administrateur. Il vaudrait mieux le faire dans une VM Windows 7 ou plus et lancer powershell en **mode Administrateur**. (click avec le bouton droit et sélectionner « exécuter en tant qu'administrateur ».*

2.1 *Utiliser Powershell*

Commande	***Que fait-elle ?***
`Get-command -commandtype cmdlet`	affiche la liste des commandes internes
`Get-command -commandtype cmdlet *write*`	
`Get-command write-*`	
`Get-command -commandtype cmdlet -noun object`	

Get-commande *-Object	

2.2 *Commandes et alias*

Pour faciliter la transition des anciens langages de script vers PowerShell, on a imaginé mettre en place un système d'alias. Celui-ci permet que la même commande puisse être frappée avec différents « verbes » de commandes.

Par exemple, frappez la commande ci-dessous et indiquez ce qu'elle fait dans la partie droite :

`Get-childitem c:\users`	

Maintenant frappez les commandes suivantes :

`Ls c:\users`
`Dir c:\users`

Vous observez que ces commandes font la même chose que la première. Ce sont des ALIAS. La première ressemble à la commande linux, la deuxième à la commande MS-DOS

Examinez l'aide (en frappant get-help) afin d'expliquer le résultat de chacune des commandes suivantes. Complétez le tableau ci-dessous :

Commande	***Résultat***	***Commande sous forme verbe-nom***
`cd c:/windows`		
`ls -R`		
`ls -force c:/`		
`Clear`		
`Pwd`		
`Cat win.ini`		

Donnez l'alias de get-command	

Expliquez la commande suivante :

`Get-command -commandtype alias`	

3 Quelques aspects fondamentaux

3.1 Les variables

Les variables commencent par le caractère $. Il existe des variables systèmes et des variables que vous pouvez vous même définir. Ces dernières restent actives pendant le temps de la session. Les variables sont typées automatiquement lors de leur initialisation (ce qui veut dire qu'elles sont reconnues comme des variables entières, flottantes, alphanumériques, dates, etc.... selon la valeur qu'on leur donne la première fois). Commentez les instructions ci-après.

`$maVariable= 'Bonjour le monde'`	
`$maVariable | get-Member`	
`$maVariable.toUpper()`	
`$maVariable.length`	

3.2 Les *caractères génériques*

Ce sont des caractères qui prennent un sens particulier dans un nom de fichier ou de répertoire :

? remplace un seul caractère

* un nombre quelconque de caractères

Frappez et commentez les instructions ci-après :

`ls /windows/*.ini`	
`ls /windows/*p*.*`	

3.3 *Faciliter la saisie des commandes.*

Pour faciliter la saisie des commandes powerShell, diverses facilités ont été mises à disposition :

L' Historique des commandes est accessible en tapant F7. Sélectionner la commande dans la liste.

On peut aussi parcourir les précédentes lignes de commandes avec les flèches (comme dans les commandes MS-DOS de Windows) et les éditer. Ceci permet très facilement de reprendre une précédente commande pour l'éditer et la modifier.

La flèche vers le haut sur une ligne de commande vide permet de rappeler les commandes précédentes

La touche tab permet aussi de faire de l'autocomplétion des commandes : essayer avec **ls c:\w** et la touche **tab.**

3.3.1 Le backtick

Le backtick permet de continuer une commande sur plusieurs lignes. Il s'obtient en frappant en même temps sur la touche altgr et 7, puis sur la barre d'espace.

Par exemple, essayez de frapper la commande ci dessous :

```
get-eventlog -logName system `
        -newest 25 `
        -entryType Error,Warning
```

Que vous donne-t-elle ?

Que signifie le paramètre -newest 25 ?

3.3.2 Exercices

Frappez et commentez le résultat des commandes suivantes :

Commande	***Résultat***	***Commande sous forme verbe-nom***
`cd ~`		
`Md tp_pws`		
`Cd tp_pws`		
`Get-date > essai.txt`		
`'une première ligne de renseignements' >> essai.txt`		
`'suivi par une seconde ligne' >> essai.txt`		
`cat essai.txt`		
`$ligne = cat essai.txt`		
`$Ligne[2]`		
`$Ligne[2][5]`		
`Write-host $ligne`		

Expliquez en quelques lignes ce que vous avez fait dans cette suite d'instructions PWS. Pour vous aider, vous pouvez utiliser l'explorateur de fichiers Windows.

4 Filtres et tubes

4.1 Les tubes

Les tubes sont des dispositifs qui établissent des liens entre des commandes. Le tube est matérialisé par le caractère « | » (qui s'obtient en frappant en même temps sur les touches altGr et 6) et qui est appelé « pipe ». Le résultat de la commande situé à gauche du pipe est utilisé comme entrée de la commande située à droite :

par exemple :

```
Get-ChildItem c:\windows | set-content monWindows.txt
```

La première commande liste le contenu du dossier windows et envoie cette liste à la deuxième qui l'enregistre dans le fichier monWindows.txt. Pour vous en convaincre, exécutez cette commande et vérifiez le contenu du fichier monWindows.txt (en frappant `cat monWindows.txt` par exemple)/

Que fait cette commande ?

```
get-process | out-file -filepath process.txt
```

4.2 Les filtres

Grâce aux pipelines, il est aisé de filtrer le résultat de certaines commandes. Un filtre se pose grâce à la commande « where-object », en abrégé : »where » ou encore plus abrégé « ? »

Par exemple :

```
get-service | where  {$_.status -eq 'stopped'}
```

Cette instruction permet de dresser la liste des services arrétés. Elle est composée de 2 commandes.

. get-service : fait la liste des services

. where : récupère cette liste et applique le filtre indiqué (ici, il faut que le service soit arrêté).

La variable $_ est une variable système qui désigne un objet du résultat de la commande de gauche (ici, un service).

La question qui se pose, c'est comment on sait que $_.status désigne l'état du service ?

Faites un simple get-service. Regardez l'entête des colonnes de la liste que vous obtenez.

Donnez les différentes colonnes que vous obtenez :

Vous pouvez faire un filtre sur chacune de ces colonnes.

Frappez la commande qui permet de filtrer tous les services démarrés :

4.3 Exercez vous

Commentez les instructions suivantes :

Commande	Commentaire
`$invite = "donnez votre nom :"`	
`$user = read-host $invite`	
`write-output("Bonjour : "+$user)`	
`$date = get-date -f 'dd-MM-yyyy'`	
`write-output ("la date du jour est :"+$date)`	
`write-output ("le mois est :"+$date.substring(3,2))`	
`write-output ("le mois est :"+(get-date).Month)`	
`$lesMois = "janvier","février","mars","avril","mai","juin","juillet","aout","septembre","octobre","novembre","décembre"`	
`write-output ("le mois en toutes lettres est : " + $lesMois[$date.substring(3,2)])`	
`$dossierCookies = [system.environment]::GetFolderPath('Cookies')`	
`write-output $dossierCookies`	
`dir $dossierCookies`	
`dir $dossierCookies \| ? {$_.name like '*google*'}`	
`Cd ~` *Note : le tild (~) est obtenu avec :* ***alGr+2, espace***	
`Dir '.\Documentsee' ? {$_.LastWriteTime.addMonths(2) -lt (get-date)}`	

Avec les dates, on peut utiliser les propriétés suivantes :

Day	Le jour dans le mois
DayOfWeek	Le jour dans la semaine
DayOfYear	Le jour dans l'année
Hour	L'heure (selon format local)
Minute	La minute
Month	Le mois
Second	La seconde
MilliSecond	La milliseconde
Year	L'année
AddDays(nb)	Ajoute un nombre de jours à une date
AddMonths(nb)	Ajoute un nombre de mois
AddHours(nb)	Ajoute un nombre d'heures
AddMinutes(nb)	Ajoute un nombre de minutes
AddSeconds(nb)	Ajoute un nombre de secondes
ToString(format)	Transforme la date en chaine de caractère selon l'expression format : on peut trouver les expressions relatives à « format » dans l'article : http://technet.microsoft.com/en-us/library/hh849887.aspx

Par exemple, expliquez ce que fait l'instruction suivante :

```
Get-EventLog -LogName System -EntryType Error -After (Get-Date).AddDays(-2)
```