SISR 4 : Exploitation des systèmes

PowerShell : les bases

Table des matières

1 Qu'est ce que powershell ?

1.1 Introduction

Powershell est un langage de script en mode commande **destiné aux systèmes Windows** et plus particulièrement aux systèmes serveurs.

C'est en 2004-2005 que Microsoft a pris conscience des faiblesses que représentait l'interface graphique pour administrer des systèmes serveurs. En effet, jusqu'alors, les administrateurs système n'avaient d'autre choix que de réaliser des scripts batch à l'aide de commandes MS-DOS. Si ceux-ci permettent de faire facilement des tâches assez faciles (monter un lecteur réseau), ils s'avèrent notoirement insuffisant lorsqu'il s'agit de **réaliser des analyses** de fichiers logs ou de faire de simples boucles. Par la suite, est apparu le langage VBScript avec l'outil d'administration WSH. Cependant, ce système est rempli de failles de sécurité. Malgré cela, c'est resté pendant de longues années le seul système permettant de faire du script sous Windows.

PowerShell est actuellement en version 3.0. Cette version est livrée nativement avec Windows 2012 et Windows 8. Il faut la télécharger pour les systèmes plus anciens. L'objectif de Microsoft est de placer PowerShell au coeur du système.

On trouve actuellement dans PowerShell de nombreuses commandes permettant de gérer Active Directory, les stratégies de groupe, la gestion des rôles et des fonctionnalités, etc...

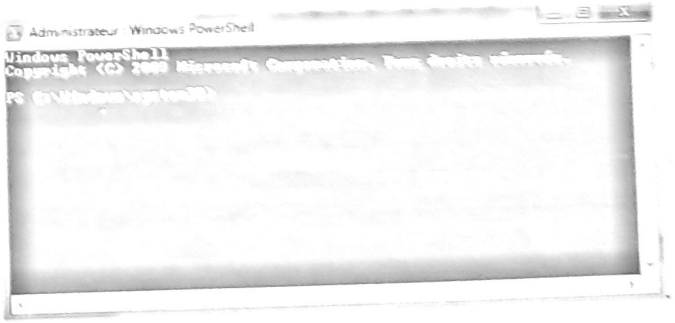
PowerShell est à la fois un interpréteur de commandes et un puissant **langage de scripts. Il s'appuie sur le framework .NET.** Celui-ci est une énorme bibliothèques de classes (un peu comme la machine virtuelle Java). A l'aide de powerShell, nous ferons naitre des objets qui nous permettrons d'agir sur l'ensemble du système d'exploitation.

Ce TP est basé sur la version 2 de PowerShell. Cependant, il fonctionne aussi très bien avec la version 3.

1.2 Utiliser Powershell

1. Lancer l'interpréteur de commandes PowerShell.

   Menu démarrer > Accessoires > Windows PowerShell® > Lancez le en mode Administrateur.



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 paramètres.

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`**

Dans PWS, la frappe n'est pas sensible à la casse, ce qui veut dire qu'on peut frapper aussi bien des majuscules que des minuscules

2. Entrer les commandes et renseigner la partie droite du tableau.

Commande	*Que fait-elle ?*
`Get-command -commandtype cmdlet`	
`Get-command -Verb write`	
`Get-command write-*`	
`Get-command -noun object`	
`Get-command *-Object`	

1.3 Commandes et alias

Pour faciliter la transition des anciens langage de script vers PowerShell, il existe les **alias**. Celui-ci permet que la même commande puisse être frappée avec différents « verbes » de commandes.

3. Entrer la commande et indiquer ce qu'elle fait.

`Get-childitem c:\users`	

4. Entrer les commandes ci-dessous.

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

Vous remarquez 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.

5. Examiner l'aide (en frappant get-help) afin d'expliquer le résultat de chacune des commandes. Compléter le tableau ci-dessous.

Commande	*Résultat*	*Commande*

		sous forme verbe-nom
`cd c:/windows`		
`ls -R`		
`ls -force c:/`		
`Clear`		
`Pwd`		

Donnez l'alias de get-command	

6. Expliquer la commande.

`Get-command -commandtype alias`	

2 Quelques aspects fondamentaux

2.1 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).

7. Commenter les instructions.

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

2.2 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

8. Saisir et commenter les instructions.

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

2.3 Faciliter la saisie des commandes.

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

Historique des commandes

Cette liste est accessible en tapant **F7**. On peut aussi parcourir les précédentes lignes de commandes avec **les flèches** et les éditer.

Le clic-droit

Le clic-droit recopie (**coller**) le texte sur la ligne de commande.

L'opérateur tilde

Le caractère tilde ~ (alt 126) **seul** renvoie au **répertoire personnel** de l'utilisateur actuel. Il s'obtient en frappant en même temps sur les touches altgr et 2, puis sur la barre d'espace.

9. Saisir et commenter la commande.

`cd ~`	

Compléter une commande

Lorsqu'on tape ligne de une commande incomplète, puis sur la touche **TAB**, l'interpréteur cherche à compléter, nom du fichier ou nom de commande, suivant le contexte.

10. Compléter les commandes.

`get-ch<TAB>`	
`ls /wind` **`<TAB>`**	

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.

11. Saisir la commande.

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

Que fournit-elle ?

Que signifie le paramètre -newest 25 ?

Exercices

12. Saisir les commandes et commenter le résultat.

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]`		
`$Ligne[2][5]`		
`$Ligne[1]`		

```
Write-host $ligne
```

13. Expliquer 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.

3 Filtres et tubes

3.1 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 :

```
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).

14. Que fait cette commande ?

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

3.2 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 « **?** »

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

Cette instruction permet de dresser la liste des services arrêtés. 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.

15. Donner les différentes colonnes obtenues par la commande.

Vous pouvez faire un filtre sur chacune de ces colonnes.

16. Saisir la commande qui permet de filtrer tous les services démarrés.