Exploitation des systèmes

PowerShell : les bases

Table des matières

1 - Qu'est-ce que PowerShell ? 2
1.1 - Introduction 2
1.2 - Utiliser PowerShell 2
1.3 - Commandes et alias 4
2 - Quelques aspects fondamentaux 5
2.1 - Les variables 5
2.2 - Les caractères génériques 5
2.3 - Faciliter la saisie des commandes. 5
Historique des commandes 5
Le clic-droit : 6
L'opérateur tilde 6
Compléter une commande 6
Le backtick 6
Exercices 7
3 - Filtres et tubes 8
3.1 - Les tubes 8
3.2 - Les filtres 8
3.3 - Exercez-vous 9

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 script 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 cœur du système. Ainsi dans Exchange 2010 les actions graphiques exécutent en réalité des commandes PowerShell.

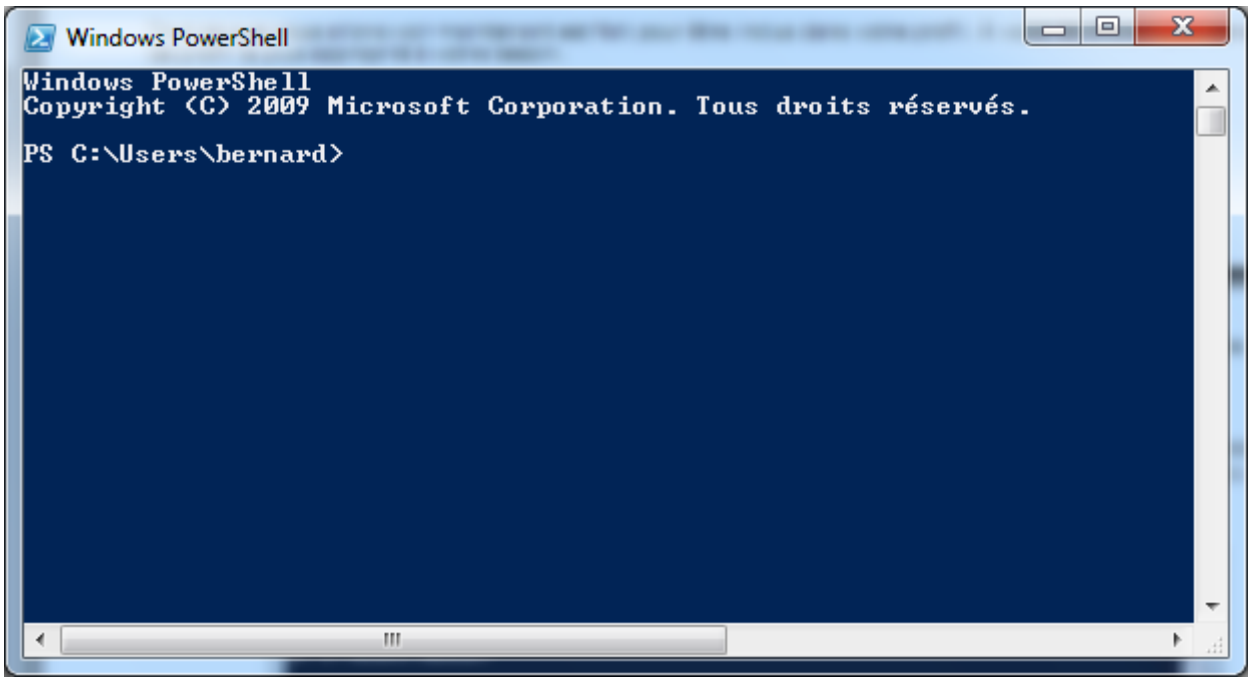
On trouve actuellement dans PowerShell de nombreuses commandes permettant de gérer Active Directory, les stratégies de groupe, la gestion des rôles et des fonctionnalités, etc...

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 bibliothèque 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.

Le TP que vous avez sous les yeux est basé sur la version 2 de PowerShell. Cependant, il fonctionne aussi très bien avec la version 3. Vous pouvez donc le faire fonctionner aussi bien sous Windows 7 que sous Windows 8.

1.2 Utiliser PowerShell

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-le en mode Administrateur. (click avec le bouton droit et sélectionner « **exécuter en tant qu'administrateu**r ». Vous obtenez alors la fenêtre suivante :



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 saisir :

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

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

Frappez les commandes suivantes et indiquez quelle est leur action dans 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-commande *-Object`	

1.3 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`	

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). Commentez les instructions ci-après.

`$maVariable= 'Bonjour le monde'`	
`$maVariable`	
`$maVariable | get-Member`	
`$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

Frappez et commentez les instructions ci-après :

`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

On peut 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.

Le clic-droit :

Le clic-droit recopie ce texte sur la ligne de commande.

L'opérateur tilde

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

Frappez et commentez la commande suivante :

`cd ~`	

Compléter une commande

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

`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.

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 ?

Exercices

Saisissez 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`		
`$Ligne[2]`		
`$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.

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 :

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
```

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 « ? »

Par exemple :

```
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.

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 :

```

```

3.3 Exercez vous

Commentez les instructions suivantes :

`$invite = "donnez votre nom :"` `$invite`	

`$user = read-host $invite`	
`"Bonjour : "+$user`	
`$date = get-date -f 'dd-MM-yyyy'`	
`"la date du jour est :"+$date`	
`"le mois est :"+$date.substring(3,2)`	
`"le mois est :"+(get-date).Month`	
`$lesMois = "janvier","février","mars","avril","mai","juin","juillet","aout","septembre","octobre","novembre","décembre"`	
`"le mois en toutes lettres est :"+$lesMois[$date.substring(3,2)-1]`	Pourquoi faire **-1** ? Essayez sans **-1**.
`$dossierCookies = [system.environment]::GetFolderPath('Cookies')`	
`$dossierCookies`	
`dir $dossierCookies`	
`dir $dossierCookies | where-object { $_.name -like '*google*' }`	
`Cd ~`	
`Dir '.\Documents' | where-object { $_.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(-10)
```