|  |  |
| --- | --- |
| **BTS SIO bloc 2** | **B2-Act4-TP2 – Powershell : les bases (2h)** |

**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 5.

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...

**Utiliser Porwershell**

Pour utiliser PowerShell, il faut lancer l'interpréteur de commandes. Vous pouvez le trouver dans le menu démarrer, dans les accessoires. Sélectionner Windows PowerShelll et lancez-le en mode Administrateur (clic 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 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

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 write-\* | Renvoie toutes les commandes dont le nom commence par le verbe write- |
| Get-command \*-Object | Renvoie toutes les commandes dont le nom finit par le nom -Object |

## Commandes et alias

Pour faciliter la transition des anciens langage 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 | Liste le contenu du répertoire 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 | Change le répertoire courant pour le C:\Windows | Set-Location |
| ls -R | Lister récursivement le contenu du répertoire courant | Get-ChildItem -Recurse |
| ls -force c:/ | Lister les fichiers cachés à la racine du disque | Get-ChildItem -Force C: |
| Clear | Réinitialise l’affichage de la console. | Clear-Host |
| Pwd | Renvoie le répertoire courant | Get-Location |
| Cat win.ini | Affiche le contenu du fichier win.ini | Get-Content |

|  |  |
| --- | --- |
| Donnez l'alias de get-command | gcm |

Expliquez la commande suivante :

|  |  |
| --- | --- |
| Get-command -commandtype alias | Renvoie les alias de toutes les cmdlet powershell |

## 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 est une chaîne de caractère égale à « Bonjour le monde » |
| $maVariable | get-Member | Renvoie le type de la variable |
| $maVariable.toUpper() | Convertit les caractères de la chaîne en capitales |
| $maVariable.length | Renvoie la longueur de la chaîne |

## 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 | Liste tous les fichiers dans le répertoire windows dont le nom finit par .ini |
| ls /windows/\*p\*.\* | Liste tous les fichiers dans le répertoire windows dont le nom comporte p, quelle que soit leur extension. |

## 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. 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.

### *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 ~ | Revient à cd C:\Users\%USERNAME% |

### *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.

|  |  |
| --- | --- |
| get-ch<TAB> | Complète Get-ChildItem |
| ls /wind **<TAB>** | Complète \Windows |

### *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 ?

Renvoie les vingt-cinq dernières erreurs et avertissements du système par ordre déchronologique.

Que signifie le paramètre -newest 25 ?

Les vingt-cinq derniers.

## Exercice

Écrivez et commentez le résultat des commandes suivantes :

|  |  |  |
| --- | --- | --- |
| Commande | Résultat | Commande sous forme verbe-nom |
| cd ~ | Définit le dossier utilisateur comme répertoire courant | Set-Location ~ |
| Md tp\_pws | Crée un dossier tp\_pws | New-Item tp\_pws\ |
| Cd tp\_pws | Définit le répertoire courant à tp\_pws | Set-Location tp\_pws |
| Get-date > essai.txt | | |
| 'une première ligne de renseignements' >> essai.txt | | |
| 'suivi par une seconde ligne' >> essai.txt | | |
| cat essai.txt | mardi 24 septembre 2024 11:06:13  une première ligne de renseignements  suivi par une seconde ligne | Get-Content essai.txt |
| $ligne = cat essai.txt | | |
| Write-host $ligne | mardi 24 septembre 2024 11:06:13 une première ligne de renseignements suivi par une seconde 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.

# Filtres et tubes

## 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](file:///c:/windows" \o "file:///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 ?

|  |
| --- |
| [g](file:///c:/windows" \o "file:///c:/windows)et-process | out-file -filepath process.txt |

Liste les processus en cours et écrit la sortie dans process.txt

## 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. Écrivez la commande qui permet de filtrer tous les services démarrés :

|  |
| --- |
| Get-Service | where {$\_.status -eq ‘running’} |

## Exercice

Commentez les instructions suivantes :

|  |  |
| --- | --- |
| $invite = "donnez votre nom :" | Définit une variable égale à « donnez votre nom » |
| $user = read-host $invite | Définit une variable égale à ce qu’entrera l’utilisateur après l’affichage de la variable $invite |
| write-output("Bonjour : "+$user) | Renvoie « bonjour : » concaténé à la variable $user |
| $date = get-date -f 'dd-MM-yyyy' | Définit une variable date égale la date actuelle au format jour-mois-année |
| write-output ("la date du jour est :"+$date) | Renvoie la chaîne « la date du jour est : » concaténée à la variable $date |
| write-output ("le mois est :"+$date.substring(3,2)) | Renvoie la chaîne « le mois est : » concaténée aux deux caractères de la variable $date après le troisième |
| write-output ("le mois est :"+(get-date).Month) | Renvoie « le mois est: » concaténé à l’attribut Month de la sortie de la commande get-date |
| $lesMois = "janvier","février","mars","avril","mai","juin","juillet","aout","septembre","octobre","novembre","décembre" | Initialise la variable lesMois contenant plusieurs chaînes de caractères |
| write-output ("le mois en toutes lettres est :"+$lesMois[$date.substring(3,2)]) | Renvoie la chaîne « le mois en toutes lettres est : » concaténée à la chaîne correspondant à l’indice au numéro du mois actuel. |
| $dossierCookies = [system.environment]::GetFolderPath('Cookies') | Définit la variable $dossierCookies égale au chemin du dossier des cookies d’Internet Explorer |
| write-output $dossierCookies | Renvoie la valeur de la variable $dossierCookies |
| dir $dossierCookies | Liste le contenu du dossier des cookies |
| dir $dossierCookies | ? {$\_.name -like '\*google\*'} | Liste les fichiers dont le nom comporte Google dans le dossier des cookies |
| Cd ~ | Définit le répertoire courant comme le répertoire de l’utilisateur qui lance la commande. |
| Dir '.\Documents' | ? {$\_.LastWriteTime.addMonths(2) -lt (get-date)} | Renvoie le dernier élément qui a été écrit il y a deux mois dans le répertoire Documents |

**ANNEXE**

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](http://technet.microsoft.com/en-us/library/hh849887.aspx" \o "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)

Liste les dernières erreurs systèmes depuis les deux derniers jours.