4.4 Connexion de commandes : le pipe

Comme les autres shells, PS permet de connecter deux commandes à l'aide du pipe « | ». C'est simplement un moyen de transférer les informations résultantes de la première commande à la commande suivante. Il y a toutefois des différences fondamentales.

Exercice 16 Saisissez la suite d'instructions suivantes :

```
Get-Process | Export-Csv procs1.csv
Notepad procs1.csv
```

Réponse :

Grâce au pipe, vous venez d'exporter les processus courants dans un fichier nommé `procs1.csv` au format CSV (Comma Separated Values) et de lancer la visualisation de ce fichier dans l'éditeur de texte. Comme vous venez de le voir, PS utilise le même principe de pipe que celui que vous pouvez connaître mais l'étend pour obtenir des effets plus importants. Il est plus riche et plus moderne car il travail directement sur des objets comme nous le verrons plus tard. Par ailleurs, il existe d'autre format d'export que CSV comme CliXML (`Export-Clixml`). L'intérêt d'exporter réside dans la possibilité de comparer ensuite les fichiers ou de les publier. Il est possible de traduire du contenu en CSV ou autre format comme de l'HTML sans forcément les sauvegarder sur disque en utilisant les commandes `ConvertTo-*`.

Exercice 17 Grâce à la cmdlet `Send-MailMessage`, envoyez votre fichier texte par mail à la personne qui est sur la machine d'à coté. Ensuite, identifiez les processus qui sont différents d'une machine à l'autre grâce à la commande `Compare-Object`. Indice : limitez vous à considérer toute la ligne pour l'instant.

Réponse :

Exercice 18 Afin de ne récupérer que le nom des processus qui sont différents sans s'intéresser à d'autres informations comme le % CPU ou autres, répondez à la question précédente en intégrant la commande import-csv ou import-CliXML en fonction de votre précédent export. On s'intéressera uniquement à la colonne `Name`.

Réponse :

Exercice 19 Par défaut, PS vous affiche les résultats de votre commande sous format texte. Il est possible de le récupérer sous un autre format. Testez la commande suivante : `Get-Process | Out-GridView`.

Réponse :

Exercice 20 Merci de réfléchir et de noter ce que pourrait faire cette commande SANS L'EXECUTER :

```
Get-Process | Stop-Process
```

Réponse :

Exercice 21 Vous souhaitez réellement savoir ce qui se passerait si vous exécutiez réellement cette commande mais vous avez peur de ce que le prof vous réserve ?! Dans ce cas, ayez confiance en lui en ajoutant l'option –`WhatIf` à la fin de la commande de l'exercice précédent. Alors, cela vous donne quoi ?

Réponse :

Exercice 22 Afin de bien comprendre la notion d'objets dans les scripts, saisissez l'instruction suivante :

```
Get-ChildItem C:\ | Format-Table –property Name,LastWriteTime
```

Réponse :

Cette commande récupère la liste des fichiers ou répertoires du chemin précisé et l'affiche en utilisant un format bien précis grâce au pipe. Néanmoins, PS diffère des autres shells dans la mesure où les informations qui transitent dans le pipe ne sont pas textuelles mais sont bien orientée objets. Pour bien comprendre cette différence, nous allons essayer d'ajouter une chaine de caractères devant chaque ligne récupérées précédemment. Pour cela, l'instruction qui vous viendrait à l'idée serait :

```
"un peu de texte avant: " + (Get-ChildItem C:\ | Format-Table –property Name,LastWriteTime)
```

Vous obtenez un résultat complètement différent de celui que vous espériez. Ceci s'explique par le fait que vous essayez de combiner du texte avec une *collection d'objets* formatés ; ainsi la chaine spécifiée s'affiche grâce à la méthode `ToString` qui est appelée pour chaque objet de la collection. Or, la méthode `ToString` appliquée à un objet renvoyé par `Format-Table` affiche tout simplement `Microsoft.PowerShell.Commands.Internal.Format.FormatEntryData`. Ainsi, PS affiche autant de fois cette chaine qu'il y a d'objets dans la table concaténée avec « un peu de texte avant ».

Exercice 23 Proposez l'instruction qui permet de faire ce que nous voulions au départ. Pour cela, transformez la

collection d'objets formatés en chaîne de caractères à l'aide de la cmdlet `Out-String`.

Réponse :

Encore une autre instruction qui vous prouvera que PS est orienté objets, saisissez la commande suivante :

```
"un peu de texte" | Get-Member
```

Vous obtiendrez les informations sur l'objet précédé du pipe, c'est-à-dire de l'objet `System.String`.

Exercice 24 En utilisant l'aide de PS, renseignez vous sur cette nouvelle commande `Get-Member` et sur une de ses options `-force`. Appliquer cette option sur la commande précédente et comparez. Identifiez les lignes supplémentaires. Cherchez à quoi elles correspondent grâce à l'aide ?

Réponse :

4.5 Les caractéristiques du langage

4.5.1 Les variables

Comme vous aurez pu le comprendre avec les instructions précédentes et/ou l'aide que vous avez consulté, les variables en PS sont définies à l'aide du caractère `$`. Voici un exemple :

```
$a= 1
$b= "1"
```

Exercice 25 Appliquez la méthode vue dans l'Exercice 24 sur chacune des variables `a` et `b` et comparez. Que donne les instructions suivantes : $a*= 2.5 et $b*=2.5 ? Affichez le contenu de `a` et `b` et regardez leur type.

Réponse :

Pour avoir plus d'informations sur les variables, lisez l'aide fournie par PS en utilisant l'instruction `help variable`. Remarquez que PS crée par défaut un disque contenant toutes les variables (`variable:`)

4.5.2 Les variables prédéfinies

En plus des variables définies par l'utilisateur, PS supporte des *variables automatiques* (constantes utilisées pour stocker des valeurs de l'état courant ; utilisez l'instruction `help about_Automatic`), des *variables d'environnement* (stocke des valeurs concernant l'environnement de travail de l'utilisateur courant et du système utilisé ; utilisez l'instruction `help about_Environment`), de *préférence* (stocke des valeurs modifiables de

configuration de PS courant ; utilisez l'instruction `help about_Preference`). Vous utiliserez très souvent les variables `$_`, `$Args`, `$Error`, `$Input`, and `$MyInvocation` mais voici les principales variables.

Exécuter l'instruction `Get-ChildItem variable:` pour avoir la liste des variables prédéfinies et leurs contenus.

Variables Automatiques	Description
$$	Dernier élément de la commande reçue par PS
$^	Premier élément de la commande reçue par PS
$?	Stocke le résultat d'exécution de la dernière commande (True ou False en fonction de la réussite de la commande)
$_	Stocke l'objet courant dans l'ensemble des objets d'un pipe. Utiliser cette variable dans les commandes qui effectuent une action sur tous les objets d'un pipe.
$Args	Représente un tableau de paramètres
$ConsoleFileName	Stocke le plus récent chemin de fichier de console utilisé (.psc1)
$Error	Stocke un tableau des dernières erreurs survenues dans la session courante. `$Error[0]` représente l'erreur la plus récente
$ExecutionContext	Stocke un objet EngineIntrinsics qui représente le contexte d'exécution de l'hôte Windows PS
$False / $True	Stocke les valeurs booléennes False et True
$ForEach	Stocke l'énumérateur d'une boucle ForEach-Object
$Home	Chemin complet du répertoire utilisateur
$Host	Objet représentant l'application courante hôte
$Input	Stocke les entrées qui sont passées à une fonction
$LastExitCode	Code de sortie du dernier programme Windows qui a été exécuté
$Matches	Stocke les chaines de caractères résultats lors de l'utilisation de l'opérateur de comparaison `-match`
$MyInvocation	Stocke un objet représentant des informations de la commande courante
$NestedPromptLevel	Stocke le niveau d'imbrication de la ligne de commande. La valeur 0 correspond au niveau original
$NULL	Stocke la valeur NULL ; peut être utilisée à la place de la chaîne « NULL »
$PID	Numéro identifiant du processus représentant PS pour l'utilisateur courant
$Profile	Chemin complet de PS pour l'utilisateur courant
$PSBoundParameters	Table de hachage contenant les valeurs des paramètres actifs
$PsCulture	Paramètres de culture du système courant (fr-FR)
$PsDebugContext	Contient des informations sur l'environnement de débogage en cours (cf. §Le débogage (Dummies page 240))
$PsHome	Chemin complet du répertoire d'installation de PS
$PsUICulture	Informations de culture sur PS-ISE
$PSVersionTable	Informations sur la version de PS
$Pwd	Objet représentant le répertoire courant
$ShellID	Chaine représentant le nom du shell courant
$This	Définit une propriété ou une méthode de script à l'intérieur d'un script

PS supporte les concepts de stockage comme le kilo-octet (`KB`), le méga-octet(`MB`), le giga-octet (`GB`), le téra-octet(`TB`), le peta-octet(`PB`). Il existe par ailleurs des instructions spécifiques aux variables comme `Get-Variable`, `Set-Variable`, `Remove-Variable` et `Clear-Variable`.

Exercice 26 Saisissez le code ci-dessous. Qu'affiche t-il ? Inversez l'ordre des variables `MyString` et `MyDouble` lors de la définition de la variable `Outstring`. Que se passe t il ? Pour y remédier, essayez d'introduire [Double] devant la variable `MyDouble`. Afin de mieux écrire votre script, récrivez cette dernière version avec l'opérateur –as.

```
$MyString = " Windows PowerShell "
$MyDouble = "2.0"
$outstring = $MyString + $MyDouble
```

```
write-output $outstring
```

Réponse :

4.5.3 Les opérateurs

Les opérateurs possibles sont :

- Les opérateurs *arithmétiques* : +, –, *, /, % (modulo) ;
- Les opérateurs *d'affectations* : =, +=, – =,*=, /=, %=, ++, – – ;
- L'opérateur $ qui désigne une variable ;
- L'opérateur $_ représente l'instance de l'objet courant renvoyé par le pipe ;
- L'opérateur & pour invoquer un bloc d'instructions comme le montre l'exemple ci-dessous.

```
PS C:\Users\denis> $a={Get-Process -id 0}
PS C:\Users\denis> &$a

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
      0       0        0         24     0                0 Idle
```

- Les opérateurs de *regroupement* : `()` ou `{}` qui regroupe un ensemble d'instructions pouvant être appelées dans une instruction, `@()` qui exécute plusieurs instructions et stocke chacun des résultats dans un tableau. Par exemple, l'instrcution `@($env:COMPUTERNAME; Get-Date;$env:LOGONSERVER )[1]` renverra la date courante.
- Les opérateurs de *comparaisons* : `-eq` (=), `-ne`(≠), `-lt`(<), `-gt`(>), `-ge`(≥), `-le`(≤), `-contains`, `-notcontains`, `-like`, `-notlike`, `-match`, `-notmatch`
- L'opérateur de conversion `-as` qui convertit une entrée en un type spécifique à .NET (`"3/31/10" -as [datetime]` donnera "Wednesday, March 31,2010 12:00:00 AM".
- L'opérateur de construction de tableaux : , (`$a = 1,2,4,6,4,2`)
- L'opérateur de construction d'intervalle : .. (`$a = 2..24`)

Exercice 27 Trouver l'instruction qui modifie le PATH courant en y ajoutant un nouveau répertoire comme c:\toto. Modifiez vos profiles (PS & PS-ISE) pour ajouter le chemin qui permet d'accéder à vos scripts.

Réponse :

Exercice 28 Trouvez l'instruction qui permet de définir la constante PI à 3.1415926. Puis supprimer la variable. Que constatez-vous ? Essayez avec l'option `ReadOnly`.

Réponse :