Module 4

Interaction avec les fichiers

Table des matières

Vous apprendrez aujourd'hui comment interagir avec les fichiers: les lire et y récupérer des données, puis y écrire pour produire un fichier de log par exemple.

L'écriture avec Out-File

Écrire dans un fichier n'est pas très compliquée. Il y a en effet un applet de commande prévu justement pour envoyer dans un fichier ce qu'il reçoit en entrée: **out-file**. Par exemple, essayez:

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

Notez qu'il n'est pas obligatoire d'écrire le -filepath, le premier paramètre est toujours considéré comme étant le chemin vers le fichier de destination:

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

Si on veut faire changer le format du contenu du fichier, il faut faire attention. Ceci ne donne pas les résultats escomptés:

```
get-process | out-file processus.txt | format-list
```

En effet, out-file ne produit pas de sortie, il envoie les données dans un fichier. Les données provenant de get-process seront donc envoyées dans processus.txt sous la forme de tableau par défaut et rien n'atteindra format-list. Il faut plutôt faire ainsi:

```
get-process | format-list | out-file processus.txt
```

Si on désire écrire du texte dans un fichier (plutôt que d'envoyer le résultat d'une commande), on ne peut pas faire non plus:

```
write-host Tôt ce matin partons aux framboises | out-file paroles.txt
```

Ceci créera bel et bien un fichier appelé paroles.txt, mais il sera vide! L'explication est simple: write-host envoie les données à la console et non pas dans le pipeline. Encore une fois, rien n'atteindra out-file. Mais pas d'inquiétude! PowerShell est plein de ressources et un applet de

commande a été prévu pour ça! Il suffit d'utiliser write-output plutôt que write-host. write-output n'envoie pas le texte directement à la console, mais il l'envoie plutôt dans le pipeline vers la prochaine commande. (Notez que si write-output est la dernière commande, la sortie sera automatiquement redirigée à la console de toute façon.)

Ajout à un fichier avec l'option "-append"

```
write-output Allons calmer toute notre faim | out-file paroles.txt -append
```

Notez au passage le paramètre -append de out-file qui ajoute à la fin plutôt que d'écraser (le comportement par défaut). Notez également que rien n'atteint la console cette fois-ci!

Si vous regardez le fichier ainsi obtenu, vous constaterez qu'il ressemble à ceci:

```
Allons
calmer
toute
notre
faim
```

write-output envoie un mot à la fois dans le pipeline, ce qui produit un fichier avec un mot par ligne... La solution? Utilisez les guillemets:

```
write-output "Puisqu'est venu mon invité splendide" | out-file paroles.txt -
append
```

Cette fois-ci, tout se retrouve bel et bien sur la même ligne.

Et pour vous simplifier encore plus la vie, l'alias write est créé par défaut et pointe vers write-output!

La lecture dans un fichier

Lire dans un fichier n'est pas beaucoup plus compliqué, en fin de compte. Il existe un applet de commande qui lit un fichier et retourne son contenu: get-content. Son paramètre -path permet de spécifier un fichier à lire. Supposons un fichier a.txt qui contient le texte suivant:

```
allo toi le jeune
comment vas-tu
Un deux un deux patate poil
Il était un petit navire
```

La commande suivante va lire le fichier et écrit son contenu à l'écran:

```
get-content  a.txt
```

C'est bien joli, mais si tout ce qu'on peut faire est d'afficher le fichier à l'écran, on aurait aussi bien pu ouvrir le fichier dans le bloc-notes... Bien entendu, ça ne s'arrête pas là! On peut très bien envoyer le contenu dans une variable avec une simple assignation:

```
$fichier = get-content  a.txt
```

Du coup, on peut regarder le contenu de la variable:

```
$fichier
```

On remarque que c'est le contenu complet du fichier, les quatre lignes l'une en dessous de l'autre. Ça ressemble étrangement à une variable de type "tableau", comme lorsque l'on fait:

```
$variable=1, 2, 3, 4
```

On peut donc tester ceci pour voir:

```
$fichier[1]
```

La deuxième ligne nous apparaît, confirmant qu'effectivement la variable contient un tableau, avec une ligne dans chaque case. Merveilleux! Pourquoi? Parce que qui dit tableau dit foreach! On pourra donc aisément parcourir le contenu du fichier une ligne à la fois:

```
$fichier = get-content  a.txt
$compteurLigne = 1
foreach ($ligne in $fichier)
{
   write-host Ligne $compteurLigne :
   write-host $ligne
   $compteurLigne++
}
```

Décortiquer le fichier

Ce qui serait encore plus intéressant, ça serait de pouvoir accéder au contenu **mot par mot**... Ça permettrait assez facilement de stocker une petite base de données dans un fichier (un enregistrement par ligne, un champ après l'autre, séparés par des espaces). Pour ce faire, rien de plus simple: notre $fichier[0] est en fait un string. Un string dans l'environnement .NET contient tout un tas de méthodes pratiques pour le manipuler. Celle qui nous intéresse ici est split. On lui passe un caractère devant être considéré comme un séparateur et le string se sépare en différents mots, qui nous sont encore une fois retournés sous forme de tableau. On peut donc utiliser un autre foreach (à l'intérieur du premier) pour passer à travers tous les mots de chaque ligne :

```
$fichier = get-content -path a.txt
$compteurLigne = 1
foreach ($ligne in $fichier)
{
   write-host "Ligne $compteurLigne :"
   $mots = $ligne.split(" ")
   $compteurMot = 1
   foreach ($mot in $mots)
   {
      write-host "  Mot $compteurMot : $mot"
      $compteurMot++
   }
   $compteurLigne++
}
```

Observons maintenant le résultat pour la ligne 2:

```
Mot 1: comment
Mot 2: vas-tu
```

Évidemment, comme le split sépare les mots à partir des espaces, "vas-tu" est considéré comme un seul mot. Toutefois, il est possible de passer à split un tableau de séparateurs qui seront tous utilisés. On pourrait modifier notre script afin de séparer aussi avec des "-":

```
$fichier = get-content -path a.txt
$separateurs = " ", "-"
$compteurLigne = 1
foreach ($ligne in $fichier)
{
   write-host Ligne $compteurLigne :
   $mots = $ligne.split($separateurs)
   $compteurMot = 1
   foreach ($mot in $mots)
   {
      write-host "  Mot $compteurMot : $mot"
      $compteurMot++
   }
   $compteurLigne++
}
```

Il est possible de mettre autant de séparateurs que désiré dans une variable.

Gestion des fichiers en format csv sous powershell

Lorsque vous travaillez avec PowerShell, nous pouvons utiliser des fichiers CSV pour importer des données dans des systèmes ou pour les utiliser comme liste de référence, d'utilisateurs, par exemple, pour mettre à jour ou obtenir des paramètres.

Pour ce faire, nous utilisons la fonction Import-CSV dans PowerShell.
La fonction Import-CSV convertit les données CSV en un objet personnalisé dans PowerShell. De cette façon, nous pouvons facilement parcourir chaque ligne du fichier CSV et utiliser les données dans nos scripts.

Importer des données depuis un fichier csv

Import-csv

Pour écrire : export-csv

L'applet de commande Import-CSV est assez simple et n'a que quelques propriétés utiles :

- **path :** (Obligatoire) Emplacement du fichier CSV
- **delimiter :** Le caractère qui permet de servir de délimiteur entre les colonnes. Virgule par défaut, mais cela vous permet de le changer
- **header :** Vous permet de définir des en-têtes personnalisés pour les colonnes. Utilisés comme noms de propriété
- **UseCulture :** Utilisez le délimiteur par défaut de votre système
- **Encoding :**Spécifiez l'encodage du fichier CSV importé

Nous allons commencer par une simple liste d'utilisateurs que nous pouvons importer dans PowerShell.

J'ai créé le fichier CSV suivant que j'utiliserai dans les exemples ci-dessous :

courrielUtilisateur,nom,titre,salaire,anciennete
jtremblay@compagnie.com,Justin Tremblay,Developpement,41911.87,2
vduhaime@compagnie.com,Vincent Duhaime,Recrutement,57918.00,4
lzee@compagnie.com,Lee Zee,marketing,78320,7
gtior@compagnie.com,Gnam Tior,ITReseau,68900,6

Ce fichier CSV avait déjà des en-têtes pour chaque colonne, nous n'avons donc pas à nous en soucier pour l'instant. Pour importer ce fichier CSV dans PowerShell, nous pouvons simplement utiliser la commande suivante :

```
import-csv  test.csv
```

ou importer les données dans une variable:

```
$usager = import-csv  test.csv
```

Il devient facile ensuite d'accéder aux données par l'intermédiaire de notre variable en utilisant les noms des colonnes qui ont été mentionnées dans le fichier.

courrielUtilisateur,nom,titre,salaire,anciennete

Cette première ligne du fichier constitue le nom des colonnes et ainsi le noms des propriétés que l'on peut utiliser pour accéder aux données de nos utilisateurs.

Exemple:
Je peux alors accéder au nom de l'utilisateur et à son courriel en faisant:

$usager.nom
$usager.courrielUtilisateur

Spécification du délimiteur

Un problème courant lorsqu'il s'agit d'importer un fichier CSV avec PowerShell est que le délimiteur n'est pas reconnu. Par défaut, l'applet de commande Import-CSV utilise la virdule comme délimiteur par défaut. Il est cependant possible de le spécifier avec l'option "-delimiter".

```
$users = Import-Csv -Path  test.csv -Delimiter ;
```

ou autre délimiteur que vous voudrez.

Traitement des données

La plupart du temps, lorsque vous importez un fichier CSV dans PowerShell, vous souhaitez parcourir chaque ligne du fichier CSV.

Pour ce faire, nous pouvons combiner l'applet de commande Import CSV avec ForEach. Dans le bloc ForEach , nous pouvons référencer chaque colonne du fichier CSV en tant que propriété de l'objet utilisateur.

En prenant l'exemple de fichier CSV ci-dessus, nous pouvons obtenir la boîte aux lettres de chaque utilisateur en utilisant le courrielUtilisateur comme ceci :

Exemple 1:

```
import-csv  test.csv  | ft courrielUtilisateur
```

Exemple 2:
Sortir uniquement le nom et le salaire des employés qui ont 5 ans ou plus d'ancienneté

```
$usagers = import-csv test.csv

foreach($usager in $usagers)
{
   if ($usager.anciennete -ge 5)
   {
      write-host "Nom: $($usager.nom)    Salaire: $($usager.salaire)"
   }
}
```

Exemple 3 (avec un where-object ou ? qui est la même chose):

```
import-csv  test.csv  |  ? {$_.anciennete -ge 5} | ft nom, salaire
```

Exemple 4 (avec le .foreach):

```
(import-csv  test.csv).foreach( { if ($_.anciennete -ge 5) { write-host "Nom: $($_.nom)  Salaire: $($_.salaire)"} } )
```

Notez bien la syntaxe.