Cégep de Sainte-Foy – Automne 2023
Systèmes d'exploitation – 420-W12-SF

Travail Pratique 3

15 janvier 2024

Préparé par
Benjamin Lemelin

1 Résumé

Rédiger divers petits scripts PowerShell et Bash effectuant des tâches plus ou moins complexes. L'objectif de certains de ces scripts est d'administrer le système : l'usage d'un VM est recommandé.

2 Conditions de réalisation

Valeur de la note finale	Contexte	Durée
30%	Individuel	1 semaine

3 Tâches à réaliser

Cette section contient la liste des scripts à rédiger pour ce travail. Il vous est recommandé de le faire dans une machine virtuelle, pour éviter de briser votre machine en cas de mauvaise manipulation.

3.1. Scripts PowerShell

Nom du script : `script1.ps1`

Tâche : Ce script prend en paramètre le chemin vers un dossier (**$path**) et y liste les fichiers plus gros qu'une certaine taille en kilooctets (**$size**). Ces deux paramètres sont facultatifs : si le chemin n'est pas fourni, listez les fichiers dans le dossier actuel. Si la taille n'est pas fournie, listez tous les fichiers sans effectuer de filtrage.

Indices : Windows conserve la taille des fichiers (propriété **Length**) en octets. Il y a 1000 octets dans un kilooctet. Un petit calcul mathématique devrait suffire.

Il est possible d'indiquer à PowerShell le type des paramètres du script. Ajoutez simplement une annotation devant le nom. Par exemple :

```
param([string] $name)
```

Il est aussi possible de spécifier une valeur par défaut :

```
param([int] $age = 18)
```

Exemple :

```
PS C:\Users\Utilisateur> ./script1.ps1
Mode                 LastWriteTime         Length Name
-a----         2023-11-06     08:39                MyFolder
-a----         2023-11-06     10:32            336 Program.cs

PS C:\Users\Utilisateur> ./script1.ps1 ./MyFolder 2
Mode                 LastWriteTime         Length Name
-a----         2023-11-06     08:42           2353 script2.ps1
-a----         2023-11-06     08:51           4406 script4.ps1
```

Nom du script : `script2.ps1`

Tâche : Ce script prend en paramètre le chemin vers un fichier de code *C#* (**$path**) et en compte le nombre de lignes. Il doit ignorer les lignes contenant des commentaires. Pour simplifier ce travail, gérez uniquement les commentaires **//** (pas **/***).

Indices : Utilisez **Get-Content** pour obtenir toutes les lignes d'un fichier.

Pour filtrer un tableau, utilisez **Where-Object**. L'opérateur **-Match** conserve les éléments contenant un bout de texte donné. Par exemple, voici comment filtrer des noms de langage de programmation pour ne conserver ceux contenant la lettre **C** :

```
("C#", "C++", "Java") | Where-Object {$_ -Match "C"}
```

Il existe aussi **-NotMatch**, qui a l'effet inverse. Vous en aurez besoin.

Enfin, sachez qu'il est possible d'obtenir le nombre d'éléments dans un tableau avec la propriété Count. Par exemple :

```
$languages = ("C#", "C++", "Java")
$nbLanguages = $languages.Count
```

Exemple :

```
PS C:\Users\Utilisateur> ./script2.ps1 ./Program.cs
Il y a 32 lignes de code.
```

Nom du script : `script3.ps1`

Tâche : Ce script permet de gérer les services Windows. Il reçoit deux paramètres :

- Le nom d'un service (**$name**). S'il s'agit du seul paramètre reçu, recherchez tous les services ayant ce nom (correspondance partielle) et affichez-les.
- L'état d'un service (**$status**), soit **Running** ou **Stopped**. Si reçu avec le nom d'un service (correspondance exacte), le script change l'état actuel du service pour le nouvel état reçu.

Si le script est utilisé sans aucun paramètre, il affiche simplement la liste de tous les services sur le système.

Indices : Si un paramètre de type **string** n'est pas fourni, sa valeur est égale à **""**.

Il est possible d'indiquer à PowerShell une liste des valeurs possibles pour un paramètre. Utilisez simplement l'annotation **ValidateSet**. Par exemple :

```
param(
    [ValidateSet("Windows", "Linux")][string] $operatingSystem
)
```

Important : Ce script nécessitera des droits administrateurs pour s'exécuter. Or, contrairement aux systèmes Linux, c'est plus compliqué que d'ajouter un simple **sudo** devant une commande.

Pour ouvrir un PowerShell en tant qu'administrateur, exécutez la commande ci-dessous dans un terminal. Cela ouvrira une nouvelle fenêtre.

```
Start-Process powershell -Verb RunAs -ArgumentList ("-NoExit", ("cd {0}" -f (Get-Location).Path))
```

Pour protéger les utilisateurs contre eux-mêmes, Windows empêche l'exécution de scripts PowerShell par défaut. Pour autoriser l'exécution de scripts, exécutez cette commande dans la fenêtre qui vient de s'ouvrir :

```
Set-ExecutionPolicy -Scope CurrentUser RemoteSigned
```

Une fois que c'est fait, vous devriez pourvoir tester votre script normalement.

Exemple :

```
PS C:\Users\Utilisateur> ./script3.ps1
Status   Name               DisplayName
------   ----               -----------
Running  AdobeARMservice    Adobe Acrobat Update Service
Running  AdobeUpdateService AdobeUpdateService
Running  Dhcp               Client DHCP
Running  Dnscache           Client DNS
Stopped  Netlogon           Netlogon
Stopped  Netman             Connexions réseau
Stopped  MozillaMaintenance Mozilla Maintenance Service
Running  mpssvc             Pare-feu Windows Defender
Stopped  pla                Journaux & alertes de performance
Running  PlugPlay           Plug-and-Play
Stopped  W32Time            Temps Windows
Stopped  wuauserv           Windows Update

PS C:\Users\Utilisateur> ./script3.ps1 Adobe
Status   Name               DisplayName
------   ----               -----------
Running  AdobeARMservice    Adobe Acrobat Update Service
Running  AdobeUpdateService AdobeUpdateService

PS C:\Users\Utilisateur> ./script3.ps1 Adobe Stopped
Set-Service : Le service Adobe est introuvable sur l'ordinateur.

PS C:\Users\Utilisateur> ./script3.ps1 AdobeUpdateService Stopped
--Aucune sortie—
```

Nom du script : `script4.ps1`

Tâche : Ce script permet de créer, compiler et d'exécuter des projets C#. Il reçoit en paramètre le nom d'un projet (**$name**) ainsi qu'une commande (**$command**) à exécuter sur ce projet. Il y a trois commandes possibles :

- **Create** : Crée une nouvelle solution Visual Studio avec un projet C#.
- **Build** : Compile le projet C#.
- **Run** : Exécute le projet C#.

Pour faire ces opérations, vous aurez besoin d'un programme nommé **dotnet** qui normalement est installé en même temps que Visual Studio.

Indices : *Créer une solution et un projet C#*

Voici comment créer une nouvelle solution Visual Studio nommée **MySln** :

```
dotnet new sln --name "MySln"
```

Vous devez placer cette solution dans un sous-dossier (voir paramètre **$name**). La solution devrait avoir le même nom que ce sous-dossier. Par exemple, supposons que le nom choisi (**$name**) est **Tp3** :

```
Tp3
└──Tp3.sln
```

La commande précédente crée une nouvelle solution, mais elle est vide. Voici comment créer un nouveau projet C# nommée **MyProj** :

```
dotnet new console --name "MyProj" --use-program-main --no-restore
```

Placez ce nouveau projet dans un sous-dossier à côté de la solution Visual Studio. Le nom du projet devrait être le même que la solution. Par exemple, supposons que le nom choisi (**$name**) est **Tp3** :

```
Tp3
├──Tp3.sln
└──Tp3
    ├──Tp3.csproj
    └──Program.cs
```

Il vous faudra aussi ajouter le projet à la solution Visual Studio. Pour ce faire, utilisez cette commande (en l'adaptant à vos besoins) :

```
dotnet sln "MySln.sln" add "MyProj.csproj"
```

Compiler une solution Visual Studio

Voici comment compiler une solution Visual Studio nommée **MySln** :

```
dotnet build "MySln.sln"
```

Exécuter un projet C#

Voici comment exécuter un projet C# nommé **MyProj** :

```
dotnet run --project "MyProj.csproj"
```

À noter que ce n'est pas la solution Visual Studio qui est exécutée, mais bien le projet C# lui-même (soit le fichier **.csproj**).

Exemple :

```
PS C:\Users\Utilisateur> ./script4.ps1 Sample Create
Le modèle « Fichier solution » a bien été créé.
Le modèle « Application console » a bien été créé.
Projet 'Sample\Sample.csproj' ajouté à la solution.

PS C:\Users\Utilisateur> ./script4.ps1 Sample Build
Version MSBuild 17.7.0+5785ed5c2 pour .NET
  Identification des projets à restaurer...
  Restauration effectuée de Sample\Sample\Sample.csproj (en 2 ms).
  Sample -> Sample\Sample\bin\Debug\net8.0\Greet.dll
La génération a réussi.
    0 Avertissement(s)
    0 Erreur(s)
Temps écoulé 00:00:00.09

PS C:\Users\Utilisateur> ./script4.ps1 Sample Build
Hello World!
```

3.2. Scripts Bash

Nom du script : `script5.sh`

Tâche : Ce script prend en paramètre le chemin vers un fichier de code *C#* (**path**) et en compte le nombre de lignes. Il doit ignorer les lignes contenant des commentaires. Pour simplifier ce travail, gérez uniquement les commentaires **//** (pas **/***).

Indices : Il est possible d'inverser le résultat de **grep** avec le paramètre **--invert-match**.

Exemple :

```
[user@fedora ~]$ ./script1.sh ./Program.cs
Il y a 32 lignes de code.
```

Nom du script : `script6.sh`

Tâche : Ce script ne prend aucun paramètre en entrée et affiche le nom d'utilisateur, le nom de l'ordinateur, le numéro de release du kernel ainsi que le modèle du CPU. Cela devrait ressembler à ceci :

```
Informations sur le système :
 - Nom d'utilisateur : felix
 - Nom de l'ordinateur : ordi-felix
 - Version du kernel : 5.13.0-20-generic
 - Model du CPU : Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz
```

Indices : Les variables d'environnement **$USER** et **$HOSTNAME** vous permettrons d'obtenir le nom d'utilisateur et le nom de l'ordinateur respectivement. Utilisez le programme **uname** pour obtenir le numéro de release du kernel. Enfin, lisez le contenu du fichier spécial **/proc/cpuinfo** pour obtenir une multitude d'informations sur le CPU.

Consultez le manuel de **uname** pour savoir quel paramètre utiliser (avec **man**). À noter que nous voulons le **kernel-release**, pas le **kernel-version**.

Pour limiter à 1 le nombre de résultats de **grep**, ajoutez le paramètre **-m1**. Envoyez ce résultat à la commande « **cut -d : -f 2** » pour obtenir uniquement le nom de modèle du CPU.

Exemple :

```
[user@fedora ~]$ ./script2.sh
Informations sur le système :
 - Nom d'utilisateur : felix
 - Nom de l'ordinateur : ordi-felix
 - Version du kernel : 5.13.0-20-generic
 - Model du CPU : Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz
```

Nom du script : `script7.sh`

Tâche : Ce script prend en paramètre un nom d'utilisateur (**name**) et un groupe (**group**). Il effectue ensuite plusieurs opérations.

Pour commencer, le script crée l'utilisateur, et s'il existe déjà, s'arrête avec un message d'erreur. Le script crée ensuite le groupe, mais s'il existe déjà, le script poursuit son exécution avec un simple message informatif.

Il ajoute ensuite le nom de l'utilisateur dans un fichier portant le nom du groupe dans le dossier **/home**. Par exemple, si le nom du groupe est **student**, le chemin vers ce fichier est **/home/student**. Ce fichier doit appartenir à l'utilisateur **root** et au groupe auquel il correspond. Tout le monde peut lire le fichier, mais seul **root** peut y écrire. Personne ne peut exécuter ce fichier.

Indices :

Vous pouvez écrire la sortie d'un programme dans un fichier avec l'opérateur **>>**. Cet opérateur ajoute le contenu à la fin du fichier. Par exemple :

```
echo "Hello World" >> myFile.txt
```

Vous pouvez vérifier qu'un programme s'est exécuté avec succès en utilisant une simple condition. Par exemple :

```
if sudo dnf install git -y; then
    echo "Git a été installé"
fi
```

Vous pouvez aussi vérifier si un programme a échoué en inversant la condition. Il suffit d'utiliser l'opérateur **!**. Par exemple :

```
if ! sudo dnf install git -y; then
    echo "Git est déjà installé"
fi
```

Pour arrêter prématurément un script, utilisez la commande **exit**. Vous pouvez aussi indiquer que le script s'est arrêté à cause d'une erreur, comme ceci :

```
exit 1
```

Enfin, pour ignorer la sortie d'un programme, envoyez-la dans le fichier **/dev/null**. C'est un fichier spécial ignorant tout ce qu'il reçoit. Voici un exemple :

```
apt install git -y &> /dev/null
```

Exemple :

```
[user@fedora ~]$ ./script3.sh felix students
Utilisateur "felix" créé et ajouté au groupe "students".

[user@fedora ~]$ ./script3.sh alexandre teachers
Utilisateur "alexandre" créé et ajouté au groupe "teachers".

[user@fedora ~]$ ./script3.sh laurie students
Info : Groupe "students" existe déjà.
Utilisateur "laurie" créé et ajouté au groupe "students".

[user@fedora ~]$ ./script3.sh felix students
Utilisateur "felix" existe déjà.
```

À la fin de l'exécution de cet exemple, le fichier **/home/students** contient :

```
felix
laurie
```

Le fichier **/home/teachers** contient seulement le nom de **alexandre**.

4 Collaboration et plagiat

Ce travail ne peut être le produit d'une collaboration entre individus. Si de tels comportements sont observés, le professeur attribuera automatiquement la note de 0 %. Aussi, tout étudiant peut être convoqué à une rencontre afin de s'assurer de sa compréhension du travail remis.

> Tout acte de plagiat, de tricherie et de fraude sera sanctionné. Constitue notamment un plagiat, une tricherie ou une fraude tout acte de copier, de fournir ou de recevoir volontairement de l'information lors d'un examen, de reproduire en tout ou en partie le travail d'une autre personne, qu'il s'agisse d'un document imprimé, multimédia ou électronique, sans y faire expressément référence, de remplacer un étudiant ou de se faire remplacer lors d'un examen, de remettre un travail réalisé par une autre personne, d'obtenir, posséder ou utiliser frauduleusement des questions ou réponses d'examen, d'utiliser du matériel, des applications , des sites Web ou des ressources non autorisés, de se faire aider par une autre personne lorsqu'il est demandé de réaliser l'évaluation seul, de falsifier les résultats de travaux ou d'examens.
>
> *Source : Politique d'évaluation des apprentissages du collège*

5 Modalités de remise

Remettez sur LÉA, dans la section travaux, tous les scripts complétés dans une archive *Zip*. Ne remettez rien d'autre : l'archive ne doit contenir que ces scripts, et rien d'autre.

Une pénalité de 15 % est appliqué en cas de retard de moins d'une journée. Au-delà de ce délai, le travail est refusé et la note de 0 % est automatiquement attribuée.

6 Évaluation

Livrable
Script 1 : • Affiche les fichiers plus gros qu'une certaine taille. • Paramètres facultatifs.
Script 2 : • Compte le nombre de lignes de code dans un fichier. • Ignore les commentaires.
Script 3 : • Liste les services (aucun paramètre). • Filtre les services (paramètre avec le nom). • Change l'état d'un service (nom et statu).
Script 4 : • Création de projet C# (solution, projet, dans un sous-dossier). • Compilation de la solution Visual Studio. • Exécution du projet C#.
Script 5 : • Compte le nombre de lignes de code dans un fichier. • Ignore les commentaires.
Script 6 : • Affiche le nom de l'utilisateur. • Affiche le nom de l'ordinateur. • Affiche la version du kernel (release). • Affiche le modèle du processeur (usage de `cut`).
Script 7 : • Création de l'utilisateur (erreur si déjà existant). • Création du groupe (message si déjà existant). • Ajout de l'utilisateur au groupe. • Écriture dans un fichier. • Modification des propriétaires du fichier. • Modification des droits d'accès du fichier.
PowerShell : • Syntaxe générale. • Usage de la section `param`. • Usage de `Where-Object`.
Bash : • Syntaxe générale. • Usage de *pipes*.

Pénalités – Rigueur
Remise du travail (incluant, mais sans s'y limiter) : • Fautes de français. • Non-respect des consignes de remise. • Travail remis en retard.