TP Shell

Frédéric Fauberteau

Durée : 3 heures

1 Introduction

Ce TP se compose de 4 exercices, à réaliser seul ou en binôme, et valant chacun 5 points de note de suivi. Voici quelques recommandations

- tout ce que j'écris entre chevrons :

```
<like_that>
```

  est à remplacer par ce que vous avez envie de mettre.
- dans ce TP, vous aurez à utiliser diverses commandes. Pour obtenir une documentation exhaustive et officielle d'une commande, il suffit de taper :

```
man <command>
```

  Celui qui m'appelle pour me dire qui obtient « `bash: erreur de syntaxe près du symbole inattendu « newline »` » parce qu'il a tapé `man <command>` **tel quel** se verra gratifié par 2 points en moins sur sa note de suivi. Si la page de **man**`ual` est trop longue, vous pouvez faire une recherche avec `/`. Rien ne vous empêche de chercher de l'aide ailleurs.
- un script shell est un simple fichier texte contenant une suite de commande à exécuter. Vous pouvez créer ce fichier avec la commande :

```
touch <file.sh>
```

  L'extension `.sh` n'est pas obligatoire, mais elle vous permet de vous rappeler qu'il s'agit d'un script shell (et non d'un exécutable binaire).
- vous devrez rendre votre script exécutable. Cela se fait simplement avec la commande :

```
chmod +x <file.sh>
```

  Non les personnes qui ont écrit ces commandes ne sont pas des fous : **ch**`ange` **mod**`e + (ajouter) e`**x**`ecutable`. Ils voulaient juste optimiser leur temps.
- À partir de ce moment là, vous allez devoir éditer votre script avec un **éditeur de texte**. Un éditeur de texte est un programme qui écrit dans un fichier uniquement les octets correspondants au texte que vous saisissez. Non, LibreOffice ou Word ne sont pas des éditeurs de texte.
- Vous devez ajouter en première ligne de votre script le *shebang* qui indique au shell qu'il s'agit d'un script et non d'un binaire. La première ligne de votre script sera donc :

```
#!/usr/bin/env bash
```

- Lorsque vous voudrez exécuter votre script, n'oubliez pas que le répertoire dans lequel vous êtes n'est sûrement pas inscrit dans la variable d'environnement `PATH`. Vous devrez donc le lancer ainsi :

```
./<file.sh>
```

- Vous devrez utiliser l'opérateur *tube* (*pipe*) : `|`. Il permet d'envoyer la sortir d'un programme vers l'entrée d'un autre. Par exemple :

```
env | grep HOME | cut -f2 -d'='
```

  affiche le nom de votre répertoire personnel. Pour vraiment comprendre, essayez :

```
env
env | grep HOME
env | grep HOME | cut -f2 -d'='
```

1.1 Zenity

Vous trouvez que le terminal est déprimant, et que taper des lignes de commande sans cesse est une attitude d'un autre âge ? Vous avez tort ... Lorsque l'on sait taper au clavier, on est beaucoup plus efficace qu'à cliquer dans tous les sens.

Mais cela demande, comme toute chose, beaucoup d'expérience et nous n'avons pas le temps ! Alors vous allez voir que vous pouvez coder en quelques mots des interfaces graphiques.

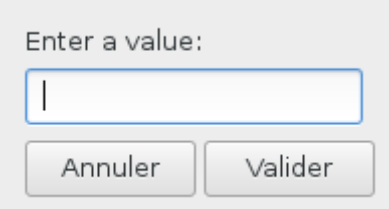

FIGURE 1 – Exemple d'une zone de saisie avec Zenity.

Pour obtenir la fenêtre de la Figure 1, il vous suffit d'écrire dans votre script :

```
zenity --entry --text "Enter␣a␣value:"
```

Maintenant, il serait intéressant de pouvoir récupérer la valeur pour pouvoir l'utiliser dans votre script. Zenity écrit par défaut sur la sortie standard (le terminal). Il faut donc récupérer cette sortie comme ceci :

```
value=`zenity --entry --text "Enter␣a␣value:"`
echo ${value}
```

L'apostrophe inversée s'obtient par `AltGr + 7`. De manière générale, vous pouvez récupérer la sortie de n'importe quelle commande avec les apostrophes inversées. La commande `echo` permet d'afficher soit une chaîne de caractères :

```
echo "Hello␣World!"
```

soit comment précédemment le contenu d'une variable shell, soit encore un mix des deux :

```
echo "I␣am␣${USER}"
```

sachant que `USER` est une variable d'environnement et que vous pouvez obtenir la liste de ces variables en tapant :

```
env
```

2 Exercices

2.1 Gérer votre dépassement de quota

Vous avez un quota sur votre compte personnel assez restrictif. Si vous le dépassez, vous risquez de ne plus pouvoir vous connecter. Vous devez écrire un script qui cherche dans votre répertoire personnel tous les fichiers dépassant une certaine taille. Le résultat sera redirigé vers un fichier dont vous aurez défini le nom. Cela signifie que vous devez trouver tous les fichiers :

- qui sont dans votre répertoire personnel et ;
- dont la **size** est **+** que `n` mégaoctets (**M**).

Ce dont vous avez besoin

- la commande `zenity` permet de créer une fenêtre de **scale** ;
- la commande `find` permet de trouver des fichiers en fonction de différents critères ;
- les opérateurs `>` et `2>` permettent de rediriger les sorties (*cf* le cours ou Google)

2.2 Trouver les failles de sécurité

Vous devez écrire un script qui cherche dans un répertoire donné tous les fichiers et sous-répertoires ayant des permissions qui vous permettent d'écrire dedans. Le résultat sera redirigé vers un fichier dont vous aurez défini le nom (peu importe). Vous devrez également rediriger la sortie d'erreur (pour éviter les messages *Permission denied*) dans le fichier `/dev/null` (le néant). Cela signifie que vous devez trouver tous les fichiers ou répertoires dont :

- soit le **group** est le même que l'un des vôtres **et** les **perm** en écriture pour ce groupe sont positionnés (**g+w**) **ou** ;
- soit les **perm** en écritures pour les autres sont positionnés (**o+w**).

Ce dont vous avez besoin

- la commande `zenity` permet de créer une fenêtre de **file-selection** dans laquelle vous pouvez sélectionner des **directory** ;
- la commande `find` permet de trouver des fichiers en fonction de différents critères ;
- la commande `groups` écrit sur la sortie standard les groupes auxquels vous appartenez ;
- l'opérateur `for` **name** `in` **words** ; `do` **list** ; `done` vous permet d'appliquer la **list** de traitement sur tous les éléments de **words** en utilisant la variable **name**
- les opérateurs >, » et 2> permettent de rediriger les sorties (*cf* le cours ou Google)

Un exemple de boucle `for` :

```
for user in `cat /etc/passwd | cut -f1 -d':'` ; do
  echo "${user} is in the `hostname` place!"
done
```

à tester !

2.3 Informations systèmes

Vous devez écrire un script qui affiche les informations du système. Cela signifie que vous devez afficher une fenêtre contenant les informations suivantes :

- la version du noyau ;
- le nombre de processeur ;
- le modèle du processeur (tous les cœurs sont du même modèle) ;
- la quantité de mémoire.

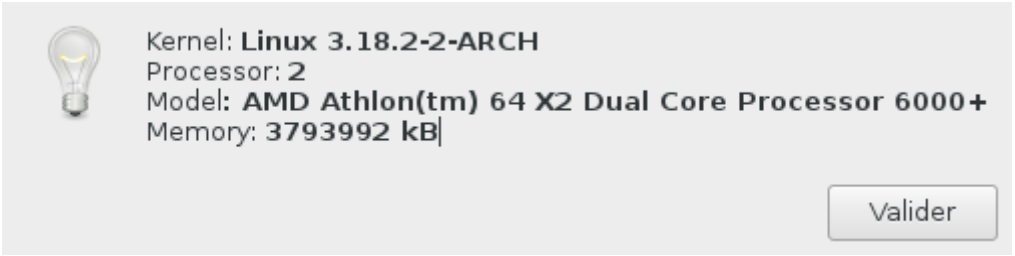

FIGURE 2 – Informations système à obtenir.

Ce dont vous avez besoin

- la commande `zenity` permet de créer une fenêtre d'**info** dans laquelle vous pouvez afficher du **text** ;
- la commande `uname` donne des informations sur la version du noyau ;
- le fichier `/proc/cpuinfo` contient des informations sur les processeurs qui sont indexés de 0 à n-1 ;
- le fichier `/proc/meminfo` contient des information sur la mémoire ;
- la commande `cat` permet d'afficher le contenu d'un fichier ;
- la commande `grep` permet d'afficher les lignes contenant une expression donnée ;
- la commande `tail` permet d'afficher les **n** dernières lignes ;
- la commande `cut` permet de découper une ligne

2.4 Archiveur de répertoire

Vous devez écrire un script qui archive (et compresse) un répertoire que vous aurez sélectionné dans un format que vous aurez également sélectionné. Cela signifie que vous devez :

- permettre la sélection d'un répertoire (comme dans l'exercice 2.2) ;
- permettre la sélection d'un format de compression ;
- compresser le répertoire au format choisi.

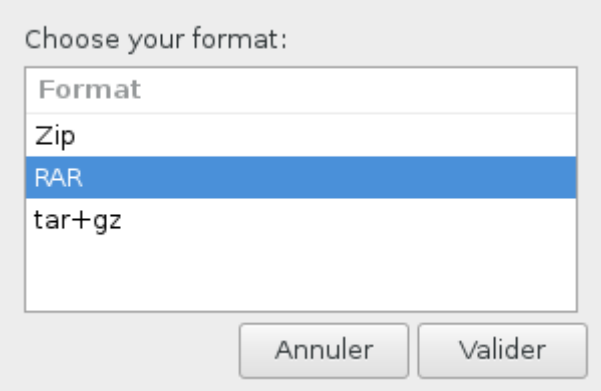

FIGURE 3 – Fenêtre de sélection du format de compression.

La fenêtre de sélection des formats de compression devrait ressembler à celle donnée en Figure 3. Mais libre à vous de rajouter des formats supplémentaires si vous le souhaitez (7z, tar+bz2, tar+xz, ...)

Ce dont vous avez besoin

- la commande `zenity` permet de créer une fenêtre de **file-selection** dans laquelle vous pouvez sélectionner des **directory** et une fenêtre de **list** dans laquelle vous pouvez spécifier des éléments dans des **column** ;
- l'opérateur `case` permet d'implémenter un `switch/case` ;
- la commande `zip` permet d'archiver et de compresser au format Zip ;
- la command `rar` permet d'archiver et de compresser au format RAR ;
- la command `tar` permet d'archiver au format tar et de compresser au format voulu (gz, bz2, xz, ...)

Un exemple de boucle `case` :

```
case ${value} in
  "yes")
    do_something
    ;;
  "no")
    do_nothing
    ;;
  *)
    error
esac
```

à tester !