

Initiation USI, TP12

Objectifs :

- L'objectif de cette séance est de vous familiariser avec l'écriture de scripts shell et de mettre en pratique les notions vues durant les séances précédentes.
- Cette feuille propose un certain nombre d'exercices et d'indications, vous êtes toutefois libres de perfectionner vos scripts si vous le jugez nécessaire. Veillez cependant à traiter l'ensemble des exercices.
- NB : plusieurs solutions peuvent être possibles.

Important :

- Pendant la séance **commentez vos scripts** pour en faciliter la relecture.
- Au besoin, vous terminerez la feuille pendant la semaine.

1 Un petit script pour bien commencer...

1. Ecrivez un script shell prenant en paramètre 2 nombres x et y et affichant : "Le nombre x est inférieur au nombre y " (ou l'inverse).

Indication : *Pensez à gérer l'égalité.*

2. Ajouter un test permettant d'assurer que le nombre de paramètres est correct.

Indication : *Dans le cas contraire, le script doit afficher un message d'erreur.*

3. Modifiez votre script pour qu'en cas de paramètre manquant, celui-ci interroge l'utilisateur de façon interactive pour lui faire préciser les deux nombres.

Indication : *Les paramètres, passés en paramètre du script ou récupérés de façon interactive pourront être stockés dans des variables utilisées dans la suite du code.*

2 Structure case

Vous avez vu en TD comment gérer un carnet de téléphone à l'aide de trois scripts qui agissent sur un fichier de données (`telephones.dat`¹) et spécifient les commandes :

- `tel-ajouter numero nom`
- `tel-chercher nom`
- `tel-afficher`

1. A l'aide de la structure de contrôle `case`, groupez ces trois commandes au sein d'un unique script `tel.sh` dont le premier paramètre sera l'opération à effectuer.

Exp : `./tel.sh chercher nom`

2. Pour chaque opération, testez la cohérence du nombre de paramètres. En cas de paramètre manquant, quittez le script avec une valeur de retour appropriée.

3. Créez une fonction d'usage²

4. Ajoutez l'appel à cette fonction pour chaque cas d'erreur (nombre de paramètres erroné, opération inconnue).

5. Ajoutez à votre commande une option d'aide `-h` qui permet à l'utilisateur de visualiser le message d'usage de la commande.

Note : le code de retour ne doit pas traduire une erreur puisque l'affichage du message est explicitement demandé.

1. dont chaque ligne est de la forme : `numero nom`

2. Une fonction d'usage affiche un message qui décrit le fonctionnement de la commande (lancement, option, paramètres...)

3 Boucle while

3.1 Faire tourner une boucle

A l'aide d'une boucle while, écrivez un script `carres.sh` qui affiche les nombres de 1 à 20 et leurs carrés.

Indication : *Rappel : une boucle while s'appuie sur une condition.*

Utilisez la commande `printf`³ pour définir la présentation : nombres cadrés à droite, et leur carré à gauche.

```

1 1
2 4
3 9
...
20 400
```

3.2 Sortir d'une boucle

Écrivez un script `devinette.sh` qui tire un nombre au hasard entre 100 et 999 puis demande à l'utilisateur de deviner le nombre mystère. Pour chaque proposition, le script indiquera "trop petit" ou "trop grand" avant d'interroger le joueur à nouveau ou s'arrêtera après avoir signalé que la solution est bonne.

Indication : *Étudiez la commande `echo $((RANDOM % 10))`. Le script doit interroger l'utilisateur de façon interactive. Voir l'instruction `break`*

3.3 Lecture de fichier

Une boucle while peut utiliser sur la commande `read` pour lire un fichier ligne par ligne. Dans votre script de carnet téléphonique, remplacez l'appel à `cat` par une boucle while pour effectuer l'affichage du répertoire.

Indication : *N'utilisez qu'un seul paramètre pour le `read` et utilisez la commande `printf` pour formater l'affichage.*

Modifiez maintenant votre script pour que l'affichage du carnet se fasse sous la forme :

```

Karleen          Num: 0553446568
Lucie            Num: 0553127369
...
```

Indication : *Vous pouvez utiliser plusieurs paramètres pour le `read`.*

3.4 Un peu plus loin

A partir d'un fichier `vacances_2010.txt` qui contient des noms de photos et des légendes comme présenté ci-dessous, on veut écrire une commande qui génère automatiquement un album photo HTML.

Exemple : `vacances_2010.txt`

```

photo1_2010.jpg:Kloster Eberbach sous la neige
photo2_2010.jpg:Statues dans la chapelle
photo3_2010:Voute externe, style roman
```

Cet album est composé d'un répertoire de pages HTML contenant chacune une photo, sa légende et

des liens vers les pages précédente et suivante. Par exemple, le fichier `page-2.html` pourra ressembler à ceci :

```

<html>
<head>Vacances en Allemagne</head>
<body>
<h1>Statues dans la chapelle </h1>

<p>
<a href="page-1.html">Prec</a>
<a href="page-3.html">Suiv</a>
</body>
</html>
```

3. Un exemple d'utilisation de cette commande est donné en section 3.17 de votre memo. Le manuel est bien sûr à votre disposition.

Le script prendra en paramètre le nom du répertoire et le titre de l'album.

```
./album.sh vac2010 "Vacances en Allemagne"
```

Indication : Pour faciliter la gestion des liens, prévoyez une page 0 pour la couverture, et une page de fin. Le répertoire `/net/Bibliothèque/ASR1/Semaine12/album` contient des photos utilisables pour vos tests.

Plutôt que de multiplier les `echo`, utilisez la forme

de redirection here-document de l'entrée standard, qui autorise l'emploi de variables. Exp :

```
echo >$fichier <<FIN
<html>
<head>$titre_album</head>
<body>
<h1>$legende</h1>
....
</html>
FIN
```

4 Boucle for

Écrivez un script qui affiche la liste des fichiers `.cc` d'un répertoire, le nombre de lignes de chaque fichier ainsi que la date de dernière modification.

Indication : Vous aurez besoin d'un ensemble de filtres étudiés précédemment (`find`, `cut`,...)

Idée de présentation :

```
lignes date          fichier
-----
      23 2 nov 22 08:56 Essais/premier.cc
     432 2 nov 27 04:32 Projet/final.cc
...
```

5 Exercices

5.1 Tables de multiplication

Créer un script `table.sh` prenant en paramètre 2 nombres et qui affiche la table de multiplication.

Exemple d'exécution :

```
./table.sh 3 12
0 x 3 = 0
1 x 3 = 3
2 x 3 = 6
3 x 3 = 9
4 x 3 = 12
5 x 3 = 15
6 x 3 = 18
7 x 3 = 21
8 x 3 = 24
9 x 3 = 27
10 x 3 = 30
11 x 3 = 33
12 x 3 = 36
```

Indication : Vous penserez à tester la validité des paramètres

5.2 A chaque type sa commande d'affichage

A l'aide de la structure de contrôle `case`, vous allez écrire une commande `voir.sh` qui prend comme paramètre un nom de fichier.

Cette commande affiche normalement le contenu du fichier (`cat`). Dans le cas où le paramètre correspond à une archive (suffixe `.tar`, `.tgz`, `tar.bz2` ou `zip`, `ZIP`...), la commande doit afficher le *catalogue* de l'archive. Si le fichier est une image (`jpg`, `jpeg`,...) ou un PDF, elle affichera ses caractéristiques.

Indication : Voir les options `-t` de `tar`, et `-l` de `unzip`.

Appuyez vous sur les commandes `file` et `pdftinfo` pour les caractéristiques des images et des PDF.

Exemple d'utilisation :

```
$ voir.sh archive.zip
Archive:  archive.zip
 Length      Date    Time    Name
-----
      0  11-29-10  10:13  vieux/
     194  11-29-10  09:59  vieux/a.html
     868  11-29-10  09:59  vieux/completer.c
    2097  11-29-10  09:59  vieux/gp
     798  11-29-10  09:59  vieux/dead.letter
-----
    3957
                    5 files
```

5.3 Vous avez fini ?

1. Créez un script permettant d'afficher la liste des fichiers du répertoire `/tmp` accessibles en lecture.
2. Modifiez le script pour qu'il prenne un paramètre qui définira si l'on teste les droits en lecture ou en écriture.
3. Ajouter un paramètre permettant de lister la liste des *répertoires* de `/tmp/` accessibles en traversée.
4. Ajouter à cette fonctionnalité la détection récursive des répertoires accessibles. La profondeur de détection sera explicitée par une option.
5. Perfectionnez votre script avec un test des arguments, une fonction d'usage, etc...