TD nº 4 - Scripts shell

Le shell est interactif mais on peut lui fournir des commandes à exécuter dans un fichier : on appelle celà un *script*. Dans ce TD, on étudie l'écriture des scripts.

Lorsqu'un script prend des arguments, veillez à vérifier que ceux-ci sont correctement passés, et si ce n'est pas le cas, affichez un message indiquant la façon dont doit être appelé le script.

Exercice 1. *Auto-reproduction*

1 Écrivez un script qui copie son propre contenu dans un fichier portant le même nom auquel on ajoute `_bis` (si le script s'appelle `copie` il va créer un script `copie_bis` ayant le même contenu).
Proposez plusieurs solutions, dont au moins une utilisant la commande `cat`.

Exercice 2. *Timestamp*

1 Écrivez un script shell nommé `change.sh` qui prend en argument un nom de fichier et affiche sa date de dernière modification puis la modifie en l'heure actuelle.
En exécutant la commande `change fichier`, vous devriez obtenir quelque chose comme

```
avant : -r--r--r-- 1 user group 40 Fev 3 2001  fichier
apr\'es : -r--r--r-- 1 user group 40 Oct 7 15:12 fichier
```

Indices :
— Cherchez dans la documentation, comment obtenir la date de dernière modification d'un fichier.
— Comment passer des arguments à un script ?
— N'oubliez pas que la commande `touch` modifie la date de dernière modification.

Exercice 3. *Sélection des fichiers*

1 Écrivez un script qui affiche la liste des répertoires dans le répertoire donné en argument.

2 Écrivez un script qui prend en argument un répertoire et affiche la liste des fichiers de ce répertoire accessibles en lecture pour au moins un utilisateur.

Indices :
— Étudiez les filtres `grep, egrep` pour filtrer les lignes qui contiennent les indications demandées.
— Que signifient `$#` et `$1` dans un script ?

Exercice 4. *Multiplication*

1 Écrivez un script `table.sh` qui prend en argument deux entiers et affiche la table de multiplication du premier entier, de 0 jusqu'au second entier. Par exemple, "`table 5 8`" affichera

```
0 x 5 = 0
1 x 5 = 5
2 x 5 = 10
3 x 5 = 15
4 x 5 = 20
5 x 5 = 25
6 x 5 = 30
7 x 5 = 35
8 x 5 = 40
```

Indices :
— Comment gérer les variables numériques ?
— Comment organiser une boucle `for` ?

Exercice 5. *Jours*

Dans cet exercice, on peut se baser sur la commande `cal`. Testez la.

1. Écrivez un script `nombreJours.sh` qui affiche le nombre de jours du mois courant.

2. Écrivez un second script `nbJours.sh` qui prend en argument un entier entre 1 et 12 représentant un mois et un entier représentant une année et affiche le nombre de jours du mois ainsi désigné. Par exemple la commande `nbJours.sh 2 2007` affichera

```
28 jours en fevrier 2007
```

Exercice 6. *Recherche d'un mot*

La commande `grep toto fich1` retourne les lignes de `fich1` qui contiennent le mot `toto`.

1. Écrivez un script `trouver.sh` qui attend deux arguments : le premier est un mot et le deuxième est un répertoire. Le script doit retourner la liste des fichiers de ce répertoire qui contiennent le mot donne en argument.

Exercice 7. *Decimus*

1. Écrivez un script shell qui crée un répertoire `Exercice6` contenant 10 fichiers nommés `un`, `deux`, etc. Chaque fichier contiendra une unique ligne : `première ligne` dans le fichier `un`, `deuxième ligne` dans le fichier `deux`, etc.

2. Modifiez le script pour qu'il vérifie avant de le créer que le répertoire `Exercice6` n'existe pas. S'il existe, il ne faut pas essayer de le créer, mais il faut quand même créer les fichiers.

Exercice 8. *Stat*

1. Que fait la commande `stat` ?

2. Essayez cette commande sur un fichier.

3. Modifiez l'affichage pour ne voir que la taille du fichier.
Indication : L'option `-c` permet de changer l'affichage.

4. Écrire un script qui affiche tous les fichiers dont la taille est supérieure à 100 ko le répertoire passé en argument.

Exercice 9. *Archive*

1. Écrivez un script qui liste récursivement les fichiers se trouvant dans le répertoire passée en argument et sauvegarde cette liste dans un fichier nommé `archive.txt`

2. Modifiez le script pour qu'il compresse la liste de fichiers générée si l'option `-z` est passée en paramètre (le fichier produit s'appelle alors `archive.zip`).

Exercice 10. *évaluation*

Exécutez et comprenez le script suivant

```
#!/bin/bash
cpt1="1"
cpt2="1"
#affectation simple
cpt1="$cpt1 + 1"
#evaluation avec let
let "cpt2=$cpt2 + 1"
#utilisation de la calculatrice bc
# $( cmd ) retourne le resultat de la commande cmd
cpt3=$(echo "$cpt2 + 1" | bc)
echo $cpt1
echo $cpt2
echo $cpt3
exit 0
```

Exercice 11. *Loop*

1 Exécutez et expliquez le fonctionnement du script suivant (utilisez la commande `man` ou cherchez sur internet la signification des commandes que vous ne connaissez pas)

```
#!/bin/bash
DIRNAME=/usr/bin
FILETYPE="shell script"
LOGFILE=logfile
file "$DIRNAME"/* | fgrep "$FILETYPE" | tee $LOGFILE | wc -l
exit 0
```

2 Exécutez et interprétez le script suivant

```
#!/bin/bash
MAX=10000
for((nr=1; nr<$MAX; nr++))
do
  let "t1 = nr % 5"
  if [ "$t1" -ne 3 ]
  then
    continue
  fi

  let "t2 = nr % 7"
  if [ "$t2" -ne 4 ]
  then
    continue
  fi

  let "t3 = nr % 9"
  if [ "$t3" -ne 5 ]
  then
     continue
  fi

  break   # LIGNE MAGIQUE
  done

echo "Nombre = $nr"
exit 0
```

3 Que se passe-t-il si la " ligne magique " est mise en commentaire ? Pourquoi ?

4 Ajoutez des commentaires pour expliquer le fonctionnement du script.

Exercice 12.

1 Écrivez un script qui donne le nombre de fichiers apparaîssant dans le PATH qui sont éxécutables et ceux qui ne le sont pas.
Dans ce script, vous aurez sans doute besoin de définir la variable ISF (Internal Field Separator) comme étant égal à " :".