TP10 : SCRIPTS AVANCES

1 SELECT/RANDOM

Dans un fichier de script **selection.sh**, copier/coller la boucle select suivante :

```
select choix in "afficher RANDOM" "quitter"; do
  echo "vous avez saisi : $REPLY"
  case "$choix" in
     "afficher RANDOM" )
          echo "RANDOM = $RANDOM"
          ;;
     "quitter" )
          echo "bye bye"
          break
          ;;
     * )
          echo 'mauvais choix !'
          ;;
   esac
done
```

Saisir différents choix (et notamment plusieurs fois le choix 1) puis quitter. S'assurer d'avoir compris le mode opératoire de cette boucle.

2 COMPTE BANCAIRE

Écrire un script **gestion.bash** dont le rôle est de gérer un compte "bancaire". Ce script fonctionnera avec deux arguments : le premier indique le sens (**-c** pour crédit ou **-d** pour débit) de l'opération et le second la somme à débiter ou à créditer sur le compte. Le script demandera également de façon interactive (par saisie au clavier) un libellé pour l'opération. Il n'est pas demandé de vérifier que le second argument est bien numérique, ni que les fichiers peuvent bien être modifiés ou créés, même si en pratique il faudrait le faire.

Après avoir vérifié le nombre d'arguments et que le premier est bien **-c** ou **-d**, le script doit modifier les deux fichiers **operations.txt** et **solde.txt**, contenus ou créés dans le répertoire de l'utilisateur, comme suit :

1. ajouter, à la fin du fichier operations.txt, une ligne de la forme :
   `date tabulation operation tabulation montant tabulation libelle`

où tabulation est le caractère de tabulation, qui sépare les 4 champs :

- date qui suit le format jj/mm/aaaa et qu'on obtient en exécutant la commande date avec l'argument (date) **+%x** ;
- operation qui vaut **crédit** ou **débit** ;
- montant qui est le montant (entier) de la transaction, communiqué en argument du script ;

- libelle qui est le libellé saisi par l'utilisateur ;

2. Ecraser le fichier solde.txt qui contient le solde précédent, et y stocker le nouveau solde. Si solde.txt n'existait pas, alors le solde précédent est nul.

3 LE JEU DU PENDU

Écrire un script **pendu.bash** qui permet de jouer au pendu. Pour cela, on dispose dans SupportCours d'un fichier mots contenant une liste de mots, à raison d'un par ligne. Au démarrage, le script utilise la variable **RANDOM** pour prendre un mot au hasard dans mots, qui sera le mot à trouver. Attention, **RANDOM** contient une valeur aléatoire comprise entre 0 et 2^{15} - 1 donc ne correspondant forcément à un numéro de ligne existant du fichier mots.

1. Le script doit alors écrire le premier caractère de ce mot ainsi que le dernier, et entre les deux autant de points qu'il y a de caractères à trouver.
2. Une lettre est alors demandée à l'utilisateur.
    a. Si cette lettre figure parmi les caractères du mot non encore découverts, alors le joueur vient de découvrir une lettre qui sera par la suite toujours affichée à la place du (ou des) point(s) correspondant(s) avec les autres caractères découverts.
    b. Sinon, le joueur a fait une erreur et le pendu commence à s'afficher. Le joueur a droit à 7 erreurs.
3. S'il découvre la totalité du mot, alors il a gagné et le script se termine avec un code de retour à 0. S'il fait plus de 7 erreurs, il a perdu et le code de retour est 1.

On aura besoin d'accéder à des caractères du mot à trouver. Pour cela, il y a plusieurs possibilités parmi lesquelles :
→travailler directement sur les (sous-)chaînes. Soit une variable **mot** (contenant une chaîne de caractère), les opérations suivantes peuvent être utilisées :

- **${mot:i:1}** est le caractère de **mot** d'indice **i** (le premier est à l'indice 0)
- **${mot: -1}** est le dernier caractère de **mot**
- **${mot:0:i}** est le début de **mot** jusqu'au caractère d'indice **i** non compris. Cette sous-chaîne est éventuellement vide
- **${mot:i+1}** est la fin de **mot** à partir du caractère d'indice **i+1**. Cette sous-chaîne est éventuellement vide

Voici un pendu complet (le joueur a perdu s'il voit cela s'afficher) :

```
==============
  ||  /     |
  || /      |
  ||/       O
  ||       --|--
  ||         |
  ||        / \
  ||
 /||
//||
==============
```

Exemple du déroulement du jeu:

```
$ ./pendu.sh
p.........n : lettre ? a
p...a.a...n : lettre ? d
Erreur : plus que 6 erreurs permises
```

```
===============
p...a.a...n : lettre ? t
p...a.at..n : lettre ? i
p...a.ati.n : lettre ? o
p...a.ation : lettre ? g
Erreur : plus que 5 erreurs permises

   ||
   ||
   ||
   ||
   ||
   ||
   ||
  /||
 //||
===============
p...a.ation : lettre ? e
p.e.a.ation : lettre ? r
pre.aration : lettre ? p
Gagné : preparation
```