TP3 *Chaînes et tableaux en BASH*

Dans ce TP, nous allons approfondir nos connaissances sur BASH et nous allons étudier et manipuler les chaînes de caractères ainsi que les tableaux et boucles. Il faut garder à l'esprit que BASH est un interpréteur de commandes et que le langage de script associé est très utile mais n'est pas aussi complet que d'autres langages. Toutefois les principales structures trouvées dans la plupart des langages se retrouvent dans le BASH.

I. Les boucles

Les boucles (`while`, `for` et `until`) sont bien évidemment présentes en BASH.. Nous présentons ici les boucle `while` et `for` qui sont les plus utiles.

La boucle `while` répète une séquence d'opérations tant que la condition testée est vrai. La condition est testée grâce au programme « `test` » présent sur votre machine. Pour alléger la syntaxe, cet appel est souvent abrégé grâce aux symboles `[ ]`.

Syntaxe :

```
while condition ; do
      opération(s)
done
```

La condition est exprimée avec le programme « `test` » présent sur votre machine (abrégé avec `[ ]`). Un appel à « `man test` » donne donc la liste des tests possibles.

Exemple :

```
CPT=0
while [ $CPT -lt 4 ] ; do
      echo $CPT
      CPT=$(expr $CPT + 1)
done
```

affiche :

```
0
1
2
3
```

On notera l'utilisation de « `expr` » et de « `$(...)` ». « `expr` » est un programme installé qui effectue des calculs simples en ligne de commande (consulter l'aide). « `$(...)` » permet d'exécuter un programme présent et de récupérer la sortie (l'affichage) de celui-ci. Dans notre programme d'exemple, la sortie est réaffectée à `CPT` pour faire évoluer le compteur.

La boucle `for` est présente sous deux formes. Une forme classique comme en langage C et une forme plus adaptée aux utilisations dans un shell.

Syntaxe de la forme classique de for :

```
for (( initial ; condition ; action )) ; do
      opération(s)
done
```

Exemple :

```
for (( i=0 ; i<4 ; i=i+1 )) ; do
      echo $i
done
```

affiche :

```
0
1
2
```

```
    3
```

La forme suivante de `for` n'utilise pas de compteur, mais une liste de chaînes séparées par des espaces. Cette liste peut être le résultat d'un `ls`.

```
Syntaxe de la deuxième forme de for :
    for x in liste; do
        opération(s)
    done

Exemple :
    for x in $(ls) ; do
        echo "# $x"
    done
```

Cette dernière forme est très utile pour parcourir le résultat d'un `ls`, `cat`, `grep` ou autre programme.

Exercices

1. Écrire un script qui affiche un menu à l'écran et qui demande à l'utilisateur de choisir une des actions suivantes : afficher la date, afficher le nombre de personnes connectées, afficher la taille disponible sur le disque.
2. Écrire un script qui recherche toutes les images (fichiers .jpg et .png de plus de 1 méga-octet sur le disque dur et qui affiche de manière synthétique la taille puis le nom du fichier. La sortie pourra être quelque chose du type :

```
  1,2M # /usr/share/themes/Adwaita/backgrounds/morning.jpg
  1,4M # /usr/share/themes/Adwaita/backgrounds/bright-day.jpg
  1,8M # /usr/share/openclipart/png/food/fruit/apple_mateya_01.png
  1,1M # /usr/share/openclipart/png/food/fruit/banana_mateya_01.png
  3,0M # /usr/share/openclipart/png/food/vegetables/salad_mateya_01.png
```

   Ceci n'est pas obtenable directement avec un `find`, il faut le combiner avec un script.

II. Les chaînes de caractères

Les chaînes de caractères sont très utiles en BASH. En effet, un shell sert principalement à manipuler les fichiers et programmes présents sur la machine. Ces fichiers et programmes sont manipulés grâce à leur nom (donc une chaîne de caractères), le BASH met à disposition un nombre impressionnant d'opérateurs de manipulation des chaînes.

- `${#string}` calcule la longueur de la chaîne.
- `${string:position}` extrait la sous-chaîne de « `string` » à partir de « `position` ».
- `${string:position:length}` extrait « `length` » caractères de -chaîne « `string` » à partir de « `position` ».
- `${string#motif}` efface la plus petite expansion de « `motif` » en partant du début de « `string` ».
- `${string##motif}` efface la plus grande expansion de « `motif` » en partant du début de « `string` ».
- `${string%motif}` efface la plus petite expansion de « `motif` » en partant de la fin de « `string` ».
- `${string%%motif}` efface la plus grande expansion de « `motif` » en partant de la fin de « `string` ».
- `${string/motif/replacement}` remplace la première occurrence de « `motif` » dans « `string` » avec « `replacement` ».
- `${string//motif/replacement}` remplace toutes les occurrences de « `motif` » dans « `string` » avec « `replacement` ».

```
chaineA="Bonjour il fait beau"
```

```
echo ${#chaineA}              # affiche la longueur : 20
echo ${chaineA:3}             # jour il fait beau
echo ${chaineA#*o}            # njour il fait beau
echo ${chaineA##*o}           # ur il fait beau
echo ${chaineA//o?/aa}        # Baajaar il fait beau
```

Vous aurez remarqué que l'on peut utiliser des expressions régulières pour définir les motifs. Malheureusement elles n'ont pas exactement la même forme que les expressions régulières des autres programmes comme `grep` et `find`. Il faut donc se méfier !

Exercices

1. Écrire un script qui convertit tous les fichiers d'une extension vers une autre (les extensions sont à prendre en paramètres). Exemple, convertir tous les fichiers se terminant par `.jpeg` en `.jpg` ou encore tous les fichiers en `tar.gz` en `tgz`.
2. Écrire un script qui analyse tous les paramètres passés en arguments et qui analyse proprement ces arguments de manière à renseigner correctement certaines variables du script.

   Étant donné le squelette de script suivant :

   ```
   # !/bin/bash

   NomFichier=""
   NomSortie=""
   Recursif=0

   …

   echo "Recursif : $Recursif"
   echo "Fichier d'entrée : $NomFichier"
   echo "Fichier de sortie : $NomSortie"
   ```

   l'appel à `./monscript.sh -r -fichier coucou.txt -sortie toto.txt` affichera :

   ```
   Recursif : 1
   Fichier d'entrée : coucou.txt
   Fichier de sortie : toto.txt
   ```

   et ceci quelque soit l'ordre des paramètres. De plus votre script doit interpréter l'argument « -help » qui affiche l'aide documenté de chaque paramètre utilisable par ce script.
3. Écrire un script qui pour une séquence de chiffres donnés en arguments affichera le nombre d'étoiles correspondantes à chaque chiffre. Exemple :

   ```
   ./script.sh 329647

   affichera :

   ***
   **
   *********
   ******
   ****
   *******
   ```

III. Les tableaux

Les versions récentes de BASH permettent de manipuler des tableaux à une dimension. Chaque élément du tableau peut-être initialisé avec la notation utilisant des `[ ]`. L'accès aux éléments se fait grâce aux `{ }` : `${nomVariable[xx]}`. Tous les éléments du tableau peuvent être affichés sous forme d'une chaîne en utilisant `${nomVariable[@]}` et la longueur (le nombre d'éléments) d'un tableau est obtenu par `${#nomVariable[@]}`.

```
Exemples :
elts[0]=zero
elts[1]=un
elts[2]=deux

echo "Tableau: ${elts[@]}"              # Affiche : zero un deux
echo "Nombre d'éléments: ${#elts[@]}"   # Affiche : Nombre d'éléments: 3

elts=(zero un deux)
# équivalent à l'initialisation précédente mais plus compacte
```

Les tableaux montrent tout leur intérêt lorsqu'ils sont utilisés avec une boucle soit pour l'initialiser soit pour le parcourir.

Exercices

1. Écrire un script qui affiche les noms de fichiers et répertoires en partant de celui qui a le plus de lettres à celui qui a le moins de lettres. On pourra obtenir le résultat suivant :

```
minizinc-tute.pdf
android-sdk-linux
nimfibo_out2.txt
log_lightsOn.txt
nimfibo_out.txt
GdriveVincent
baker_out.txt
qbf_out.txt
EBP_DIVERS
VirtualBox
Documents
EmbossPad
MiniZinc
MPG.pdf
Desktop
Dropbox
builds
Unix
Safe
Vms
```

2. Écrire un script qui prend comme argument un nom de fichier et qui affiche à l'écran toutes les lignes du fichier entre chevrons (< ... >). Pour cela il faut charger les lignes du fichier dans un tableau (une ligne par case) et ensuite parcourir le tableau pour afficher un chevron ouvrant, la ligne puis un chevron fermant.

   Astuce : Pour récupérer une ligne par case, il faut modifier la variable d'environnement `IFS` au moment où vous récupérez les lignes. Il faut la positionner à « `$'\r\n'` » puis après la restaurer à sa valeur initiale.
3. Écrire un script qui affiche l'espace disponible du disque sur lequel on se trouve. Le résultat sera obtenu par analyse de la commande « `df -h .` ».