TP d'Administration

TP3 : Chaînes et tableaux en BASH

Benoit Da Mota

Septembre 2021

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 disponibles dans la plupart des langages se retrouvent dans le BASH.

1 Les boucles

Les boucles (`while`, `for` et `until`) sont bien évidemment présentes en BASH. Nous présentons ici les boucles `while` et `for` qui sont les plus utiles. La boucle `while` répète une séquence d'opérations tant que la condition testée est vraie. La condition est testée grâce au programme « `test` » présent sur votre machine. Pour alléger la syntaxe, cet appel est souvent abrégé grâce aux symboles `[ ]`. Un appel à `man test` donne la liste des tests possibles.

Syntaxe :

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

Le script ci-dessous :

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

Retourne le résultat suivant :

```
0
1
2
3
```

On notera l'utilisation de `expr` et de `$( )`. Le programme `expr` effectue des calculs simples en ligne de commande (consulter l'aide). `$( )` permet d'exécuter un programme de récupérer la sortie (l'affichage) de celui-ci. Dans le script précédent, la sortie de `expr` réaffectée à `CPT` pour incrémenter 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
```

L'exemple suivant affiche les entiers `0`, `1`, `2`, `3`.

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

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 programme comme `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
```

affiche :

```
# index.html
# myfile.dat
# TP.pdf
```

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

Question 1.1 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.

Question 1.2 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 ressembler à :

```
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.png1
```

Ce résultat ne peut être obtenu directement avec un `find`, il faut l'intégrer à un script.

2 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 désignés par leur nom (qui est une chaîne de caractères), le BASH met à disposition un nombre important d'opérateurs de manipulation de chaînes.

- `${#string}` calcule la longueur de la chaîne
- `${string:position}` extrait la sous-chaîne de `string` à partir de `position`
- `${string:position:length}` extrait `length` caractères de la chaîne `string` à partir de `position`
- `${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`.

```
chaine="Bonjour il fait beau"

echo "${#chaine}"         # affiche la longueur : 20
echo "${chaine:3}"        # jour il fait beau
echo "${chaine#*o}"       # njour il fait beau
echo "${chaine##*o}"      # ur il fait beau
echo "${chaine//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 !

2.1 Exercices

Question 2.1.1 Écrire un script qui convertit tous les fichiers d'une extension vers une autre (les extensions seront les arguments du script). On pourra par exemple convertir tous les fichiers se terminant par `.jpeg` en `.jpg` ou encore tous les fichiers en `.tar.gz` en `.tgz`

Question 2.1.2 Compléter le script ci-dessous avec une analyse de tous les paramètres passés en arguments, ceux-ci étant destinés à instancier certaines variables du script.

```
#!/bin/bash

NomFichier=""
NomSortie=""
Recursif=0

...

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

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

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

Et ce quel que soit l'ordre des paramètres. De plus votre script doit interpréter l'argument `-help` qui affiche l'aide documentée de chaque paramètre utilisable par ce script.

Question 2.1.3 Écrire un script qui pour une séquence de chiffres donnée en argument affichera le nombre d'étoiles correspondant à chaque chiffre. Par exemple, `./script.sh 329647` devra retourner :

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

3 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 les crochets. L'accès aux éléments se fait en entourant le nom du tableau d'accolades : `${nomTableau[xx]}`. On peut produire une chaîne de caractères contenant tous les éléments d'un tableau séparés par un espace en invoquant `${nomTableau[@]}` et la longueur (le nombre d'éléments) d'un tableau est obtenue par `${#nomTableau[@]}`. Par exemple :

```
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'initialisation des éléments, soit pour le parcours des éléments.

3.1 Exercices

Question 3.1.1 Écrire un script qui affiche les noms de fichiers et répertoires du répertoire courant en partant de celui qui a le plus de lettres, jusqu'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
```

Question 3.1.2 Écrire un script qui prend comme argument un nom de fichier et qui affiche à l'écran toutes les lignes du fichier entre chevrons (`<` et `>`). 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.

Question 3.1.3 Écrire un script qui affiche l'espace disponible du disque sur lequel on se trouve. Le résultat sera obtenu par analyse de ce qu'affiche la commande `df -h .`

Question 3.1.4 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` (cette fois-ci on doit analyser le résultat complet de `df`).

4 Création d'une galerie de photos

Le but de ce petit projet est d'écrire un script permettant de créer et de mettre à jour automatiquement une galerie d'images. Les images que l'on souhaite ranger se trouvent initialement dans une archive (vous pouvez récupérer le fichier `images.tar` sur `Moodle`). Une fois l'archive décompressée, toutes les images se trouvent en vrac dans un répertoire. On souhaite créer un script qui permette de les ranger dans des répertoires en fonction de la date de la prise de vue de la photo. Ainsi, on aura d'abord un répertoire par année, puis un répertoire par mois, puis par jour. Les photos seront déplacées pour être rangées dans cette arborescence. Avant de détailler le sujet voici un petit exemple d'une exécution du script (appelé ici `exif_mv.sh`) :

```
./exif_mv.sh --src images --dest galerie --mask '*.jpg'
images/greylag_goose.jpg: OK.
images/crocodile.jpg: pas de date.
images/frog.jpg: OK.
images/tiger.jpg: OK.
```

```
images/elephant.jpg: OK.
images/Malard.jpg: pas de date.
4 fichier(s) traité(s).
```

Le script a été exécuté sur l'archive d'images que vous avez téléchargée, on remarque que certaines images n'ont pas de date, elles ont donc été ignorées. Voici l'arborescence obtenue après le traitement des huit images ayant une date :

```
find galerie/
galerie/
galerie/2006
galerie/2006/05
galerie/2006/05/29
galerie/2006/05/29/greylag_goose.jpg
galerie/2016
galerie/2016/03
galerie/2016/03/27
galerie/2016/03/27/tiger.jpg
galerie/2009
galerie/2009/12
galerie/2009/12/28
galerie/2009/12/28/elephant.jpg
galerie/2007
galerie/2007/08
galerie/2007/08/23
galerie/2007/08/23/frog.jpg
```

Pour obtenir les informations d'une image (une photo), le programme `exiftool` peut être utilisé. Il permet d'analyser le fichier image afin d'obtenir certaines informations. L'information qui nous intéresse est la date qui se trouve la plupart du temps dans le champ « `Date/Time Original` ».

4.1 Exercices

Question 4.1.1 Écrire un script qui prend en argument un répertoire source puis un répertoire destination, et qui crée la galerie dans le répertoire destination en fonction des images contenues dans le répertoire source. Les arguments seront passés comme sur l'exemple précédent. Il est conseillé d'ajouter un argument « `-help` » qui affichera l'aide du script. Lorsqu'une image ne contient pas d'information sur la date de la prise de vue, elle sera laissée dans le répertoire source. De plus, le script affichera un message d'erreur si tous les arguments nécessaires ne sont pas fournis. Il créera également les sous-répertoires nécessaires dans le répertoire destination (les années, mois et jours qui n'ont pas déjà été créés).

Question 4.1.2 Modifier ce script en ajoutant une option qui permet de modifier le comportement du script pour qu'il demande à l'utilisateur la date à mettre lorsqu'il n'y en a pas dans le fichier. Le script pourra donc être exécuté sans le paramètre, dans ce cas, il ignore les images sans date, soit avec, dans ce cas, il demande à l'utilisateur la date pour ce fichier. Pour assigner une date à un fichier image qui en est dépourvu, on peut utiliser le programme `exiftool` en ajoutant le champ `datetimeoriginal` dans la photo.

Question 4.1.3 Enfin, il se peut que certains fichiers qui ne sont pas des images viennent perturber la bonne exécution de notre script (un simple fichier texte présent dans le répertoire source par exemple). Modifier le script en ajoutant un paramètre permettant de traiter uniquement les fichiers dont le motif correspondra à celui passé en argument.