TP2 : Langage de commande Shell d'UNIX

Fichiers de commande

1 Rappels

Script

Les scripts sont des programmes qui permettent d'automatiser certaines tâches répétitives telles que l'administration ou les sauvegardes de fichiers. Ce sont des fichiers de texte en général sans extension ; ils ont parfois l'extension `.sh` ou `.bash`. Ils contiennent les mêmes commandes que celles exécutées en ligne de commande, plus éventuellement des structures itératives (`for`, `while`, `until`) ou conditionnelles (`if`) et des appels à des fonctions. Un script contient en général une commande par ligne. Il peut être paramétré, les paramètres sont désignés par `$1`, `$2`, ... `$9`.

Exécuter un script shell

Pour exécuter un script shell, il suffit de taper `bash` suivi du nom de ce fichier dans une fenêtre de terminal. L'interprète exécute les commandes de manière séquentielle, dans l'ordre où elles apparaissent dans le fichier. Si s'on donne les droits d'exécution au fichier et que le fichier est placé dans un répertoire défini dans la variable `PATH`, il suffit de taper le nom du fichier.

La variable `PATH`

Pour pouvoir exécuter un programme, le système d'exploitation doit pouvoir retrouver le fichier correspondant à cette commande. C'est la variable d'environnement `PATH` qui contient la liste des répertoires que l'interprète explore pour trouver un programme à exécuter. Vos scripts seront donc placés dans le répertoire `$HOME/bin` (voir TP n°1).

2 Avant de programmer

- Vérifiez que le répertoire `$HOME/bin` existe et qu'il apparaît dans votre variable `PATH`.
- Conseil : utilisez l'éditeur de texte **gedit**
- Evitez d'utiliser les caractères accentués qui sont codés différemment dans les éditeurs sous Ubuntu et dans les fenêtres de terminal, ou alors utilisez l'éditeur **nano**.
- Pensez à utiliser votre cours et l'aide en ligne pour la syntaxe des commandes.

3 Scripts à écrire pendant la séance de TD

3.1 Bonjour

Dans votre répertoire `bin`, créez le fichier de commande `bonjour` suivant :

```
#!/bin/bash
# pour appeler ce script : bonjour nom prenom
if test $# -eq 2 ; then
    echo "Bonjour $1 $2 et bonne journee !"
else
    echo "Syntaxe : bonjour nom prenom"
fi
```

Testez le script

3.2 `$?`

Dans le shell, la variable prédéfinie `$?` contient la valeur de sortie de la dernière commande. Par convention la valeur 0 signifie qu'il n'y a pas eu d'erreur. Lorsqu'on utilise la commande `test`, cette variable est modifiée selon que la condition est vraie ou non. Essayez les commandes suivantes :

```
test 0 -ne 1; echo $?
test 1 -ne 1; echo $?
```

3.3 Tester l'existence de fichier et/ou de répertoire

1. Créez dans votre répertoire `bin` un fichier de commandes **testfic** qui reprend l'exemple du cours (ci-contre).
2. Affichez l'aide de la commande.
3. Testez cette commande pour un fichier et un répertoire existants/non existants.
4. Commentez le code :
 - Que représentent le `$#` et le `$1` ?
 - Que fait `test $# -eq 0` ?
 - Que signifie `>&2` ?
 - Que fait `test "$1" = "" || test $1 = "--help"` ?
 - Pourquoi a-t-on écrit : `test "$1" = ""` et non `test $1 = ""` ?
 - Que représentent `[ -f $1 ]` et `[ -d $1 ]` ?
5. Modifiez le script **testfic** en un script nommé **testnfc** de façon à ce qu'il permette de tester l'existence de plusieurs fichiers ou répertoires (nombre indéfini de paramètres).

```
#!/bin/bash
if test $# -eq 0
then
    echo "Pas de parametre !" >&2
fi
if test "$1" = "" || test "$1" = "--help"
then
    echo "usage : testfic nomfichier"
elif [ -f $1 ]; then
    echo "le fichier $1 existe"
elif [ -d $1 ]; then
    echo "le repertoire $1 existe"
else
    echo $1 est absent !
fi
```

3.4 Affichage de la date d'aujourd'hui

Ecrire un script **datedujour** qui affiche :

```
Nous sommes le Xeme jour du Yeme mois de l'annee Z.
```

Pour cela, reprendre le script donné en cours qui affiche la date sous cette forme et le modifier de sorte que les zéros non significatifs n'apparaissent pas ; on pourra aussi tenir compte du 1 et afficher 1er au lieu de 1eme

Rappel : Pour extraire une sous-chaine d'une variable, on utilise la syntaxe suivante :
 `${nomvar:d:lg}`
 `nomvar` est le nom la variable
 `d` est l'indice de début de la sous-chaine (0 est l'indice du premier caractère)
 `lg` est la longueur de la sous-chaine
exemple : si la variable `nomvar=tralala` , alors `${nomvar:1:3}` correspond à la chaine "ral"
si `lg` est négatif, la sous-chaine est tronquée de `lg` caractères
 `${nomvar:1:-2}` correspond à la chaine "rala"

Indication : la commande `d=`date +%d/%m/%Y``
à pour effet de mémoriser dans la variable `d` la date sous la forme JJ/MM/AAAA.

3.5 Tâche de fond

Ecrire un script **horloge** qui lance un processus fils en tâche de fond (par exemple `xclock &`), puis affiche le PID de ce processus fils en utilisant la variable `$!` . Puis utiliser la commande `wait $!` pour attendre la fin de ce fils. Afficher un message avant de se mettre en attente, puis lorsque le processus fils est terminé.

3.6 Sauvegarder des fichiers et des répertoires

Les scripts shell peuvent être utiles pour la réalisation de sauvegardes automatiques.

Ecrire la commande **sauver** qui copie des fichiers passés en paramètres dans le répertoire **sauvegarde** du répertoire courant (à créer s'il n'existe pas). Afficher un message d'erreur si un nom correspond à un répertoire ou si un fichier est absent.