Programmation système et réseaux

Corrigé du TP 3: Scripts shell

Arnaud Giersch Guillaume Latu

Introduction au scripts shells (bash)

1. Placer les commandes qui suivent dans le fichier *~/temp/bonjour*, puis essayer le script :

   ```
   #!/bin/bash
   # Cette première ligne de commentaire est lue et
   # sert a déterminer quel shell sera appelé pour exécuter
   # ce script !
   # pour appeler ce script : ./bonjour nom prénom
   if test $# -eq 2; then
       echo "Bonjour $2 $1 et bonne journée !"
   else
       echo "Syntaxe : $0 nom prénom"
   fi
   ```

   Dans ce script, on utilise la structure conditionnelle *if then else fi* du shell. Pour savoir si la condition est vraie ou fausse, le shell exécute la commande parenthésée. Faites un `man test` pour comprendre comment s'utilise `test`.

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. Essayer les commandes `test 0 -ne 1; echo $?` puis `test 1 -ne 1; echo $?`.

3. Écrire un script qui affiche "fichier présent" si l'argument de la commande correspond à un fichier qui existe (pour cela utiliser la commande `test`).

   Correction :

   ```
   #!/bin/bash
   test -f $1 && echo fichier présent
   ```

4. Écrire un script qui affiche "répertoire présent" si l'argument de la commande correspond à un répertoire qui existe.

   Correction :

   ```
   #!/bin/bash
   test -d $1 && echo répertoire présent
   ```

5. Écrire un script qui lance un processus en tâche de fond (par exemple `xclock &`), puis qui affiche le PID de ce fils en utilisant la variable `$!`. Utiliser la commande `wait $!` pour attendre la fin de ce fils au sein du script.

   Correction :

   ```
   #!/bin/bash
   xclock &
   echo PID: $!
   wait $!
   ```

Scripts shells simples

1. Créer une commande qui, lorsqu'elle est appelée, affiche le nombre d'arguments qui lui sont fournis, ainsi que le premier de ces arguments, s'il y en a au moins un.

   Correction :

   ```
   #!/bin/bash
   echo Il y a $# arguments
   test $# -ge 1 && echo Le premier argument est: $1
   ```

2. Que fait le script suivant :

   ```
   #!/bin/bash
   ((test $1 -lt $2) && (echo '$1 < $2')) || (echo '$2 < $1')
   ```

   Après avoir testé ce script, remplacez les apostrophes (`'`) par des guillemets (`"`). Constatez que l'exécution de ce script produit un résultat différent.
   Récrivez ce script en utilisant une structure conditionnelle *if then else fi*.

   Correction : `if test $1 -lt $2; then echo '$1 < $2'; else echo '$2 < $1'; fi`

3. Que fait le script suivant ?

   ```
   #!/bin/bash
   ls $1 2>/dev/null
   if test $? -eq 0; then echo fichier existe; else echo fichier inexistant; fi
   ```

   Récrivez ce script en utilisant l'option `-e` de `test` et en supprimant l'appel à `ls`.

4. Écrire un script `coupe` qui prend trois arguments, dans l'ordre : un nom de fichier et deux entiers *n* et *p*, et qui affiche les lignes *n* à *p* du fichier. Pour cela, vous utiliserez les commandes `head` et `tail`.

   Correction :

   ```
   #!/bin/bash
   head -$2 $1 | tail +$1
   ```

Encore des scripts shells

1. Un processus shell possède des variables auxquelles sont affectées certaines valeurs. Voyons certaines commandes bash pour manipuler les variables. La liste des variables s'obtient avec la commande `set`. Pour affecter une valeur à une variable de type chaîne de caractère, on utilise la syntaxe `VARIABLE="valeur"`. Lorsqu'une variable est définie, elle est accessible depuis le shell où elle a été définie. Pour qu'une variable soit accessible par des processus fils d'une certain shell, il faut exporter ces variables avec la commande `export VARIABLE`. La liste des variables exportées est accessible avec la commande `env`. Pour affecter le résultat d'une expression arithmétique à une variable, on peut utiliser `let` (commande de bash) de la manière suivante : `let VAR2=$VAR1+1`. Tapez (dans cet ordre) les commandes suivantes, et comprenez le résultat :

   ```
   VAR1=11; VAR2=22; let VAR3=$VAR1+1; export VAR2
   env | grep VAR.=
   set | grep VAR.=
   bash
   env | grep VAR.=
   set | grep VAR.=
   exit
   ```

2. Écrire un script affichant les nombres de 1 à jusqu'à la valeur passée en paramètre du script à l'aide d'une boucle *while do done*.

   Correction :

   ```
   #!/bin/bash
   i=1
   while test $i -le $1; do echo $i; let i=$i+1; done
   ```

Modifier ensuite le script pour utiliser la structure *until do done*.

Correction :

```
#!/bin/bash
i=1
until test $i -gt $1; do echo $i; let i=$i+1; done
```

Modifier enfin le script pour calculer la moyenne de ces nombres de tous ces nombres.

Correction :

```
#!/bin/bash
i=1; s=0
while test $i -le $1; do let i=$i+1; let s=$s+$i; done
let m=$s/$1
echo Moyenne: $m
```

3. Examiner puis modifier le script qui suit. Vous ferez en sorte que lorsque l'utilisateur répond « o », alors :
   (a) si `$file` est un répertoire alors on affiche le contenu de ce répertoire (commande `ls`) ;
   (b) si `$file` est un fichier de type lien on affiche ce lien ;
   (c) si `$file` est un fichier non exécutable on affiche le fichier (commande `less`).

   Si l'utilisateur répond « n », rien ne se passe, et s'il répond « q » le script doit s'interrompre. Vous utiliserez une structure *case in esac* pour considérer les trois réponses possibles : o, n et q.

   Correction :

```
#!/bin/bash
prog=`basename $0`
if [ -z $1 ]; then
    cat <<EOF
 Usage : $prog repertoire
EOF
    exit 0
fi
cd $1
for file in *; do
    echo -n "affiche : ${file} (o|n|q): "
    read reponse
    case "${reponse}" in
    o|oui|y|yes)
        if [ -d $file ]; then
            ls $file
        elif [ -h $file]; then
            echo -n lien vers; ls -l $file | cut -d">" -f2-
        elif ! [ -x $file ]; then
            echo "$file pas executable"
        else
            echo "$file executable"
        fi;fi;fi
        ;;
    n|no|non)
        ;;
    q)
        exit 0
        ;;
    *)
        echo "réponse incorrecte: $reponse"
        ;;
    esac
done
```