La programmation shell

Corrigé TP 8

Ceci est un exemple de solution, il y en a d'autres ; l'important est de vérifier que cela marche, mais si votre solution marche aussi, choisissez la plus intéressante. Si certaines commandes n'ont pas encore été utilisées, allez voir le man. Suivant les versions de linux certaines commandes peuvent fonctionner différemment, toujours revoir le man.

1. **que fait le script :**

```
for x in un deux trois quatre
do
   echo x= $x
done                ?
```

Il affiche :

```
x= un
x= deux
x= trois
x= quatre
```

2. **Faites un script (*saluer*) qui vous dit bonjour en affichant votre login («Bonjour, toto») (bien sûr toto est un exemple), quand vous tapez saluer [votre-login] mais aussi, quand vous tapez juste saluer**

Il faut éditer un fichier ***saluer*** et le rendre ensuite exécutable :

```
if [ -n "$1" ]   then
            LOGIN=$1
else
             LOGIN=`whoami`
fi
echo "Bonjour, $LOGIN"
```

3. **tester :** echo 'je suis la commande cataloguée ' $0 etc…..

4. **Faire un script qui affiche la phrase : Le script [params] a [n] paramètre(s), que voici : [paramètres], en remplaçant les mots entre crochets par leur valeur; par exemple :**

```
$ params bla ble bli blo blu
La commande params a 5 paramètre(s), que voici : bla ble bli blo blu
```

echo "La commande $0 a $# paramètre(s), que voicil : $@" **($@ ou $*)** équivalent

5. **Créez une commande qui, lorsqu'elle est appelée, renvoie le nombre d'arguments qui lui ont été fournis, ainsi que le premier de ces arguments.**

```
echo "J'ai reçu $# arguments"
echo "Le premier est $1"
```

6. **écrire un script *combien* qui vous dit combien de personnes sont loguées sur une machine donnée; par exemple :** $ combien nacre

Il y a 5 personne(s) loguées sur nacre en ce moment.

```
N=`who | wc -l
MACHINE=`whoami`
echo "Il y a $N personne(s) logués sur $MACHINE en ce moment"
```

Attention : la réponse est 1 car vous êtes en local et donc seul
Combien de personnes potentielles sur nacre : utiliser la commande ypcat

7. **On veut remplacer le suffixe .htm d'un ensemble de fichiers en -.html.**

Le script suivant lancé avec "**/*.htm" comme paramètre devrait fonctionner :

```
for i in $@
  do
        mv $i `echo $i | sed -e "s/htm$/html/"`
  done
```

8. petit script *op* qui calcule la somme de 2 nombres passés en paramètre ainsi que la division et le reste, puis incrémentation du 1ier argument

op a b

```
a=$1
b=$2
x=`expr $a + $b`

echo "a + b = $x"
# ou
echo $((a + b))

z=`expr a / b` #  division
y=`expr a % b`#  reste
echo  "a div b = $y"
# modulo
echo "a mod b = $y"
inc = `expr $1 +1`
echo "(incrémentation d'une variable)"
echo "a + 1 = $inc"
```

9. **Créer un petit script qui affiche les nombres de 1 à 10 à chaque ligne (avec while), puis leur somme (avec for)**

```
i=0;
while test $i -ne 10
do
        i=$(($i+1));
        echo $i ;
done
1
2
….
```

8
9
10

```
somme=0
for i in 1 2 3 4 5 6 7 8 9 10
do
   somme=`expr $somme + $i`
done
echo "Somme  de 1 a 10 : $somme"
```

10. programme shell reponse_*oui* affichant *OUI* si l'utilisateur a saisi le caractère *o* ou *O*

```
$reponse_oui
echo ″Entrez votre réponse : ″
read rep
case $rep in
      o|O ) echo OUI ;;
      *)    echo Indefini ;;
 esac
```

11. avec la commande date créer un script qui donne
le jour est : mar 4, le mois est : nov

```
jour  = `date| cut –c1-4
mois = `date| cut –c5-8
No    = `date| cut –c9-10
echo  le jour est : $jour $N0, le mois est :  $mois
```

12. même script mais le mois étant abrégé (les 3 premières lettres du jour), le script écrit le jour en entier (utiliser case)

```
jour = `date| cut –c1-4
mois = `date| cut –c5-8
No = `date| cut –c9-10
case $jour in
          lun) j=lundi ;;
          mar) j=mardi ;;
          mer) j=mercredi ;;
          jeu) j=jeudi ;;
          ven) j=vendredi ;;
          sam) j=samedi ;;
          dim) j=dimanche ;;
esac
echo  le jour est : $j $N0, le mois est :  $mois
```

13. Ecrire un script qui recherche le fichier passé en argument dans le répertoire courant, il se termine lorsqu'il le trouve, variante : le fichier est lu (n'est pas en argument) si le fichier n'existe pas, on redemande de le saisir.

```
while ls | grep $1 > /dev/null
do
   echo je suis en attente du fichier $1
   echo sleep 10
done
echo le fichier $1 existe
```

Variante

```
echo -e "Entrez un nom de fichier"
read fich                ou  read -p "Entrez un nom de fichier : " fich
while [ -z "$fich" ]
```

```
do
        echo -e "Saisie à recommencer"
        read fich
done
```

1. **dev/null** désigne la *poubelle*, tout ce que la commande ls | grep $1 peut retourner est redirigé à la *poubelle*.
2. avec exemple d'attente grâce à sleep

Remarque : L'option **–p** de **read** affiche une chaîne d'appel avant d'effectuer la lecture ; la syntaxe à utiliser est : **read –p** *chaîne_d_appel* **[** *var* **…]**

Ex : $ **read -p "Entrez un nom de fichier : " fich**

14. faire un menu qui affiche choix 1-: calcul de ma moyenne
choix 2- calcul de la somme
choix 3 – le minimum
choix 9 – Quitter

```
echo "***********  Menu General  *************"
echo "<1>    calcul de la moyenne
echo "<2>    calcul de la somme
echo "<3>    le minimum
echo "<9>    Quitter
echo "****************************************"

echo -n "Choix "          # quelle est la signification de –n?
read choix                # peut etre passé en parametre
case $choix in
   1) echo "Vous avez choisi  le menu 1"     ou  calcul de la moyenne ;;
   2) echo "Vous avez choisi  le menu 2"  ou    calcul de la somme ;;
   3) echo "Vous avez choisi le menu 3"  ou    le minimum   ;;
   9) echo "Vous avez choisi  le menu 9"    ou termine              ;;
   *) echo "Choix incorrect"  ;;
esac
```

autre solution

```
if [ $choix == 1 ] then
   echo "moyenne"
elif [ $choix == 2 ]
  then
      echo "somme"
  elif [ $choix == 3 ]
       then
            echo "minimum"
        elif [ $choix == 9 ]
            then
                 echo "Quitter"
             else
                   echo "Choix incorrect"
       fi
```

15. **Réalisez un shell-script qui, en fonction de l'heure courante (retournée par commande date,** uniquement l'heure**) affiche "Bonjour" entre 0h et 12h, "Bon après-midi" de 12h à 17h, et "Bonne soirée" de 17h à 24h**

```
echo `date | cut –c12-20’
heure  =  `date | cut –c12-13’
if [ $heure -lt 12 ]  then
        echo "Bonjour !"
elif [ $heure -lt 17 ]
  then
        echo "Bon après-midi !"
  else
        echo "Bonsoir !"
fi
```

16. **Transformer la chaîne suivante en majuscule, idem pour une chaîne quelconque passée en paramètre avec le script *minmax***

```
chaine="Bonjour, comment allez VOUS aujourd'hui ?"
  echo $chaine | tr 'A-Z' 'a-z'
```

minmax chaine

```
  echo $1 | tr 'A-Z' 'a-z'
```

17. **Un ensemble de noms de fichiers sont en majuscules. On veut tout convertir en minuscule**

```
#!/bin/sh
for i in $@;
      do
            mv $i `echo $i | tr '[:upper:]' '[:lower:]'`
      done;
```

18. **Créez un script *rang* qui affiche l'indice de son premier argument dans la liste des arguments suivants. Par exemple,**

rang toto tata titi toto tutu

renvoie le rang de toto dans la liste tata titi toto tutu, c'est-à-dire 3.

```
ARG=$1
shift
N=0
for i in $@
  do
        N=`expr $N + 1`
        [ x$ARG = x$i ] && echo $N
  done
```

Complément

19. **On ne s'intéresse ici qu'à des fichiers contenant un mot par ligne. Écrire un script qui compte le nombre de mots contenant une des lettres «r», «s» ou «t», et parmi eux, ceux qui ont au moins deux telles lettres. On donnera aussi le nombre de mots ne contenant aucune voyelle. Cela donnera par exemple :**

Dans le fichier titi, vous avez :
45 mots contenant «r», «s» ou «t», et parmi eux,

12 contiennent deux de ces lettres au moins.
Il y a aussi 10 mots ne contenant aucune voyelle.

```
RST=`grep "[rst]" $1 | wc -l`
DEUX=`grep "[rst].*[rst]" $1 | wc -l`
CONS=`grep -v "[aeiou]" $1 | wc -l`

echo "Dans ce fichier, vous avez| :";
echo "$RST mots contenant «r», «s» ou «t» et parmi eux,";
echo "$DEUX contiennent deux de ces lettres au moins.";
echo "Il y a aussi $CONS mots ne contenant aucune voyelle.")
```

20. **On veut chercher toutes les occurrences des quatre éléments (terre, air, eau, feu) dans la première partie de *Germinal*.**

    On veut aussi que le résultat soit placé dans un fichier, et que le résultat soit classé : toutes les lignes qui contiennent «air», puis toutes celles qui contiennent «eau», etc.

```
FICHIER=resultat
MOTS="terre air eau feu"

[ -w   $FICHIER ] && echo "Le fichier $FICHIER existe déjà" && exit 1
for i in $MOTS
  do
        echo "Lignes contenant $i" >> $FICHIER
        grep $i zola*.txt >> $FICHIER
        echo >> $FICHIER
  done;
```

21. **Chercher le mot «mine» dans les chapitres 3, 4 et 5 de la première partie de *Germinal*, et obtenir un fichier dans lequel figure le nombre d'occurences du mot dans les fichiers, avec le numéro des lignes**

```
#!/bin/sh
nombre=0
for i in zola[345].txt
do
        echo "Fichier $i :"
        numligne=0
        while read ligne
        do
                res=`echo $ligne | tr -s ' ' '\n' | grep '^mine$' | uniq -c | tr -s ' ' | cut -d" " -f2`
                numligne=`expr $numligne + 1`
                if [ ! -z $res ]       then
                        echo "$numligne : $ligne"
                        nombre=`expr $nombre + $res`
                fi
        done
done
echo "Le nombre d'occurences du mot mine est" $nombre
```

22. **Créez un script *coupe* qui prend trois arguments, le premier étant un nom de fichier et les deux autres des entiers *l* et *l'*, et qui affiche les lignes comprises entre *l* et *l'* dans le fichier. Par exemple :**

```
coupe fic 4 8
```

    affichera les lignes 4 à 8 du fichier `fic`. Affichez des messages en cas d'erreur (nombre de paramètres incorrect, fichier inexistant, etc).

```
[ $# != "3" ]   &&    echo   "Nombre de paramètres incorrects" && exit 1
[ ! -r $1 ] && echo "Fichier $1 inexistant" && exit 1

cat $1 | head -8 | tail  -4
```

les 2 1ere lignes peuvent se faire avec des tests, cette écriture est raccourcie

23. **Écrire un script permettant d'appliquer divers filtres sur un fichier. Ce script est lancé avec un argument, qui doit être un nom de fichier appelé fichier de travail; dans le cas contraire, on affiche un message d'erreur. On attend ensuite une commande en mode interactif, qui peut être :**
    - end : le programme s'arrête;
    - cherche : le programme lit alors une ligne au clavier contenant un motif et une autre contenant un nom de fichier, puis écrit dans ce fichier les lignes du fichier de travail contenant le motif;
    - tete ou fin : le programme lit une ligne au clavier contenant un entier *n* puis une ligne contenant un nom de fichier. Il écrit ensuite les *n* premières (resp. dernières) lignes du fichier de travail dans le fichier précisé;
    - autre : message d'erreur.

```
if [ $# != "1" ]  then
        echo "Indiquez un nom de fichier";
         exit 1;
fi
if [ ! -r $1 ]  then
         echo "Fichier $1 inexistant"
        exit 1
fi
TRAVAIL=$1
while true
do
        read CMD
        case $CMD in
                end)    exit 0;;

                cherche)  echo -n "Motif   ? ";   read MOTIF;
                          echo -n "Fichier ? ";  read FICHIER;
                          if [ -w $FICHIER ] then
                                  echo "Le fichier $FICHIER existe déjà"
                          else
                                  grep $MOTIF $TRAVAIL > $FICHIER
                          fi;;
                tetelfin)
                          echo -n "Lignes  ? "; read LGN;
                          echo -n "Fichier ? "; read FICHIER;
                           if [ -w $FICHIER ]  then
                                   echo "Le fichier $FICHIER existe déjà"
                          else
                                  case $CMD in
                                      tete)  head  -n $LGN < $TRAVAIL > $FICHIER ;;
                                       fin)  tail    -n $LGN < $TRAVAIL > $FICHIER ;;
                                  esac
                            fi ;;
              *)      echo "Commande inexistante";;
              esac
done
```