

Shell (3)

1 Aiguillage et pattern matching

1.1 EXERCICE Écrivez et testez le script suivant :

affichageoptions

```
opt_a=0
opt_b=0
opt_f=0
f=0

USAGE="echo Usage: 'basename $0' [-a] [-b] [-f nom] ... ; exit 1"

if test $# = 0
then
    eval $USAGE
fi

while [ $# -gt 0 ]
do
case $1 in
    -a )
        opt_a=1
        ;;
    -b )
        opt_b=1
        ;;
    -f )
        opt_f=1
        shift
        f=$1
        ;;
    -* )
        eval $USAGE
        ;;
    esac
    shift
done
```

```
echo opt_a=$opt_a
echo opt_b=$opt_b
echo opt_f=$opt_f
echo f=$f
```

1.2 EXERCICE En utilisant la structure de contrôle **case**, écrire un script **backup** tel que **backup** $\langle nom_1 \rangle$ $\langle nom_2 \rangle$... recopie le fichier $\langle nom_i \rangle$ en $\langle nom_i \rangle$.OLD. L'option **-s** permet de choisir le suffixe ; l'option **-b** produit une sauvegarde de la sauvegarde si elle existe déjà.

1.3 EXERCICE Modifier **ifdef** de façon à ce qu'il

- accepte l'option **-C** : ajout du commentaire

```
/* Fichier  $\langle nom \rangle$ .h */
```

- rajoute automatiquement le suffixe **.h** s'il n'est pas présent dans $\langle nom \rangle$.

2 Redirections et manipulation de fichiers

2.1 EXERCICE Essayez de comprendre ce que font les petits scripts suivants :

mystere

```
#!/bin/sh

exec 3<fichier
( while read ligne
do
    echo Ligne $ligne
done ) <&3
```

mystere_bis_et_repetita

```
#!/bin/sh

exec 5<fichier
( while read ligne
do
    echo Ligne $ligne
done ) <&5
```

mystere2

```
#!/bin/sh
```

```
exec 4<fichier
exec 5<fichier2
(read ligne
  (while read ligne
    do
      echo Ligne $ligne
    done
  ) <&4
echo Ligne $ligne
) <&5
```

étrange

```
exec 4<fichier
exec 5>fichier3
(while read ligne
do
  echo Ligne $ligne
done
) <&4 >&5
```

3 Visibilité des variables

Il existe des variables prédéfinies sous Unix. Vous avez déjà vu la variable `$HOME`. Il existe une commande `set` permettant de lister l'ensemble des variables définies à un instant donné.

3.1 EXERCICE Affichez l'ensemble des variables définies dans l'environnement courant.

3.2 EXERCICE Définissez une variable de nom `UN` sans l'exporter (avec la commande `UN=un`), et une variable de nom `DEUX` que vous exporterez (avec la commande `export DEUX=deux`).

3.3 EXERCICE Affichez les valeurs de ces deux variables dans le shell courant.

3.4 EXERCICE Lancez un nouveau shell. Affichez la valeur des deux variables. Que remarquez-vous ? Que pouvez-vous en conclure ?

3.5 EXERCICE Écrivez dans le fichier script `proc` la commande permettant d'affecter à la variable de nom `TROIS` la valeur `trois`, sans l'exporter et toujours à l'intérieur de ce script, visualisez le contenu de la variable `TROIS`.

3.6 EXERCICE Exécutez la suite de commandes suivantes et expliquez les résultats :

```
set | grep TROIS
chmod 700 proc
proc
set | grep TROIS
```

3.7 EXERCICE Vous avez constaté que la variable `TROIS` n'a pas été trouvée dans le shell courant. Pensez-vous que le résultat aurait été différent si la variable avait été exportée? Testez-le³⁰.

REMARQUE : Pour que l'exécution du fichier `proc` puisse modifier l'environnement courant (i.e., le shell courant), il faut exécuter le script en le précédant de la commande `.` (point) : `. proc` ou bien de la commande `source : source proc`.

3.8 EXERCICE Testez cette dernière commande, et vérifiez que la variable `TROIS` est bien définie.

REMARQUE : Grâce à cette commande, il est possible de prendre en compte des modifications apportées aux fichiers de configuration du `bash`.

3.9 EXERCICE Écrivez et testez le script suivant :

proc2

```
#!/bin/sh
QUATRE=quatre
(
  CINQ=cinq
  (
    SIX=six
    echo SIX=$SIX
    echo CINQ=$CINQ
    echo QUATRE=$QUATRE
  )
  echo SIX=$SIX
  echo CINQ=$CINQ
  echo QUATRE=$QUATRE
)
echo SIX=$SIX
echo CINQ=$CINQ
echo QUATRE=$QUATRE
```

Que pouvez vous conclure quant au rôle des parenthèses?

3.10 EXERCICE Dans votre *xterm* tapez directement la commande :

```
(TREIZE=XIII ;echo TREIZE=$TREIZE) ;echo TREIZE=$TREIZE
```

D'après vous, est-ce votre *shell* d'origine qui exécute les instructions entre parenthèses? Quel est le *shell* qui exécute l'instruction sans parenthèse?

³⁰Attention : la commande `export` n'est pas reconnue par le shell `sh`. Pour utiliser cette commande utilisez dans votre script le `bash`, en écrivant comme première ligne : `#!/bin/bash`.