TD n°2 : Programmation shell

CORRECTION

Exercice 1 – Exemple de programme shell

Solution :

1) Ce programme crée le répertoire fourni en argument, et au besoin tous les sur-répertoires intermédiaires nécessaires. **Note :** voir l'option '`-p`' de `mkdir`.

2) `mystere /home/dupont/test/projet`

Exercice 2 – La boucle while

Solution :

```
#!/bin/sh

if [ $# -eq 0 ]; then
        echo "sans argument"
        exit 0
fi

while [ $# -gt 0 ]; do
    echo $1
    shift
done
```

Exercice 3 – La commande read

Solution :

```
#!/bin/sh

exec < $1               # redirection de l'entrée standard
while read ligne ; do
      echo ">$ligne"
done
```

Exercice 4 – La boucle for

Solution :

```
#!/bin/sh

for rep in *; do                # pour chaque fichier "rep" du repertoire courant
        if [ -d "$rep" ]; then # si rep est un repertoire
                echo $rep       # affiche rep
        fi
done
```

Exercice 5 – Opérateurs sur les chaînes

Solution :

Ce programme détermine si l'utilisateur dont le nom est donné comme argument est connecté.

```
#!/bin/sh

w=`who | grep $1`
if [ -z "$w" ]; then echo "$1 n'est pas connecté"; fi
```

Exercice 6 – Les conditionnelles imbriquées

Solution :

```
#!/bin/sh

if [ $# -eq 2 ]; then           # si le nombre d'arguments est égal à 2 :
        rep="."                 # variable rep vaut le repertoire courant = "."
        droit=$1                # variable droit vaut le premier argument $1
        ext=$2                  # variable ext vaut le deuxieme argument $2
elif [ $# -eq 3 ]; then         # si le nombre d'arguments est égal à 3 :
        rep=$1                    # variable rep vaut le premier argument $1
        droit=$2                  # variable droit vaut le deuxieme argument $2
        ext=$3                    # variable ext vaut le troisieme argument $3
else
        echo "erreur : donnez 2 ou 3 arguments" 1>&2
        exit 1
fi

for fich in "${rep}"/*"${ext}"; do  # pour chaque fichier "fich" dans rep/*ext:
        chmod g$droit "$fich"       # changer le droit du groupe "g" sur "fich"
        echo $fich
done
```

Exercice 7 – L'instruction case

A noter que l'option `-i` de la commande `rm` fait exactement ce que nous voulons.

Solution :

```
#!/bin/sh

while [ $# -ne 0 ]; do
        fich=$1; shift
## ou : for fich in $*; do
        repeat=1
        while [ $repeat -eq 1 ]; do
                echo "Voulez-vous reellement effacer le fichier \"$fich\" ?"
                read reponse

                case "$reponse" in
                        [Oo][Uu][Ii])
                                rm "$fich"
                                echo "suppression confirmée"
```

```
                                repeat=0
                                ;;
                        [Nn][Oo][Nn])
                                echo "suppression abondonnée"
                                repeat=0
                                ;;
                        *)
                                echo "reponse invalide"
                                ;;
                esac
        done
done
```

Exercice 8 – La commande basename

Solution :

```
#!/bin/sh

if [ $# -ne 2 ]; then
        echo "erreur : donnez 2 arguments." 1>&2
        exit 1
fi

vieux=$1
nouveau=$2

for fich in *"${vieux}"; do
     base=`basename "$fich" "$vieux"`
     mv "$fich" "$base$nouveau"
     if [ $? -ne 0 ]; then
       echo "je ne peux pas renommer \"$fich\" en \"$base$nouveau\"" 1>&2
     fi
done
```

La commande `basename` est ici utilisée pour récupérer dans la variable `base` le nom du fichier sans son extension représentée par la variable `$vieux`. Le nouveau nom du fichier (avec sa nouvelle extension) est donc facile à reconstruire : c'est le nom du fichier `base`, suivi de la nouvelle extension `$nouveau`.

Exercice 9 – Guillemets, quotes ou back quotes ?

Solution :

1)

```
#!/bin/sh

echo "Entrer le nom d'un répertoire : "
read dir
echo "Le répertoire $dir contient les fichiers suivants :"
ls "$dir"
```

2)

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

```
echo "Entrer le nom d'un répertoire : "
read dir

if [ $# -ne 0 ]; then
    echo "Erreur : Le script doit être appelé avec 1 argument" 1>&2
    exit 1
fi

if [ ! -d "$dir" ]; then
    echo "Erreur : \"$dir\" n'est pas un répertoire" 1>&2
    exit 1
fi

if [ ! -r "$dir" ]; then
    echo "Erreur : je ne peux pas lire le répertoire \"$dir\"" 1>&2
    exit 1
fi

# On essaye d'aller dans le répertoire $dir pour utiliser
#  pwd pour déterminer son nom absolu
cd "$dir" 2> /dev/null

if [ $? -ne 0 ]; then
    # on garde alors le nom donné
    cmd="ls \"$dir\""
else
    dir=`pwd`
    cmd="ls"
fi

echo "Le répertoire \"$dir\" contient les fichiers suivants :"
exec $cmd
```

Exercice 10 – Les expressions régulières

Solution :

1) Il faut indiquer que l'on veut le début de la ligne, avec le chapeau. Afin de préciser que la ligne commence par un '`a`' minuscule ou majuscule, il y a deux façons de faire :

- Utiliser l'option `-i` qui fait ignorer la différence entre les majuscules et le minuscules.
- Dire que l'on cherche un '`a`' ou un '`A`' en utilisant les crochets.

Enfin, il faut protéger les signes contre le shell, pour qu'il ne les interprète pas; on met donc l'expression entre apostrophes.

Il faut donc écrire :

```
grep -i '^a' fichier
```

ou

```
grep '^[aA]' fichier
```

2) C'est le dollar (`$`) qui représente la fin de la ligne. Il faut donc écrire :

```
grep 'rs$' fichier
```

3) `grep '[0-9]' fichier`

4) `grep '^[A-Z]' fichier`

5) `grep '^[BEQ]' fichier`

6) Le point d'exclamation n'a pas de signification particulière avec `grep`, on peut donc le mettre tel quel :
```
grep '!$' fichier
```

7) Pour que `grep` interprète littéralement le caractère '.' et ne le considère plus comme spécial, il faut le faire précéder d'un backslash (`\`).
```
grep '\.$' fichier
```

8) Les caractères spéciaux sont protégés par les crochets. On peut donc écrire :
```
grep '[^.,;:?!]$' fichier
```

On peut aussi utiliser l'option `-v`, qui prend toutes les lignes où ne figure pas une chaîne de caractères donnée; dans ce cas, on tape :
```
grep  -v '[.,;:?!]$' fichier
```

9) On tape au choix :
```
grep '[a-zA-Z]r' fichier'
```

ou

```
grep '[[:alpha:]]r' fichier'
```

10) C'est le symbole `\<` qui désigne un début de mot. La première lettre du mot est indifférente, la seconde est un '`r`'. On écrit donc :
```
grep '\<.r' fichier
```