FICHE 17-18-19 – Scripts : arithmétique, conditions, boucles

1.

```
cows=($(ls /usr/share/cowsay/cows/))
```

- `$(ls /usr/share/cowsay/cows/)` → fournit la liste des fichiers du répertoire fourni
- `($(ls /usr/share/cowsay/cows/))` → convertit la liste en tableau
- `cows=…` → cows reçoit le tableau des noms de fichiers

```
num=$(($RANDOM % ${#cows[*]}))    OU    ((num=$RANDOM % ${#cows[*]}))
```

- `$RANDOM` → génère une valeur entière aléatoire
- `${#cows[*]}` → nombre d'éléments du tableau cows
- `$(($RANDOM % ${#cows[*]}))` → modulo du nbre aléatoire par le nbre d'éléments dans cows
- `num=…` → num reçoit la valeur entière calculée

```
cowsay -f ${cows[num]} "Hello, $USER! I am $(uname -o) on $HOSTNAME"
```

- `${cows[num]}` → valeur de l'élément à l'indice num du tableau cows
- `"Hello, $USER! I am $(uname -o) on $HOSTNAME"` → phrase contenant des variables d'environnement (USER et HOSTNAME pour les noms d'utilisateur et de la machine) et une substitution de commande (uname -o qui fournit le type d'OS de la machine).
- `cowsay -f …` → le personnage en indice num du tableau cows est affiché, avec la phrase spécifiée dans le phylactère

2.

```
for f in $(ls /usr/share/cowsay/cows/); do cowsay -f $f "$f says hi!"; done
```

OU (en passant à la ligne dans le shell)

```
for f in $(ls /usr/share/cowsay/cows/)
do cowsay -f $f "$f says hi!"
done
```

3.

```
#!/bin/bash
################################################################################
# somme.sh
# calcule différentes sommes arithmétiques
################################################################################

echo
echo "  Somme des nombres d'une liste"
echo "  -----------------------------"
liste="1 -3 45 67"
for nbr in $liste; do
    ((somme+=nbr))
    echo "Avec $nbr, la somme partielle vaut $somme"
done
echo "La somme finale vaut $somme"
```

4.

```
#!/bin/bash
#############################################################################
# somme.sh
# calcule différentes sommes arithmétiques
#############################################################################

echo
echo "  Somme des nombres du tableau"
echo "  ----------------------------"
tab=(1 -3 45 67)
cnt=${#tab[*]}
i=0
#while ((i < cnt))
#while test $i -lt $cnt
while [ $i -lt $cnt ]
do
    ((somme+=tab[i]))
    ((i++))
    echo "Avec ${tab[i-1]}, la somme partielle vaut $somme"
done
echo "La somme finale vaut $somme"
```

OU avec reconversion du tableau en liste (mais il n'y a aucun intérêt de convertir une liste en tableau pour reconvertir celui-ci en liste !)

```
tab=(1 -3 45 67)
for nbr in ${tab[*]}; do
    ((somme+=nbr))
    echo "Avec $nbr, la somme partielle vaut $somme"
done
echo "La somme finale vaut $somme"
```

5.

```
echo
echo "  Somme des arguments"
echo "  -------------------"
if [ $# -eq 0 ]; then
    echo "Erreur: Pas d'argument"  > /dev/stderr  OU  >&2
    exit 2
else
    somme=0
    for nbr in $*; do
        ((somme+=nbr))
        echo "Avec $nbr, la somme partielle vaut $somme"
    done
    echo "La somme finale vaut $somme"
fi
```

6.

```
echo
echo "  Somme des nombres lus"
echo "  ----------------------"
echo "Donnez les nombres à lire (un par ligne; Ctrl-D pour terminer)"
somme=0
while read nbr; do
    if [ -z "$nbr" ]; then
        break
    fi
    ((somme+=nbr))
    echo "Avec $nbr, la somme partielle vaut $somme"
done
echo "La somme finale vaut $somme" > /dev/stderr
```

OU avec un *ET logique* pour inclure le test dans la condition de boucle :

```
...
while read -p "Entrez un nombre : " nbr && [ -n "$nbr" ] ; do
   ...
```

REMARQUES :

- cf. `help read` (*builtin command*) : read est évalué à faux si Ctrl-D (fin de fichier) est entré au clavier ➔ sortie de boucle ; pour qu'une ligne vide provoque également la sortie de boucle, il faut ajouter une condition supplémentaire (`test -z line`)
- Remarque : le test `[ -z "$nbr" ]` qui vérifie si une chaîne est de longueur nulle fonctionne aussi si on oublie les guillemets. En effet, dans le cas où $nbr est vide, le teste devient `[ -z ]`. Or, lorsqu'il n'y a qu'un seul paramètre dans un test, le shell teste par défaut si sa longueur est non nulle. Le test devient donc `[ -n "-z" ]` qui vaudra toujours vrai puisque la chaîne "-z" est non vide ! Par contre, cela n'est pas le cas pour le test `-n` qui vérifie si une chaîne est non vide : l'oubli de guillemets dans le test `[ -n "$nbr" ]` ne fonctionnera pas ! En effet, le test -n sur une chaîne vide devrait renvoyer faux. Or, sans les guillemets, le shell interprète le test comme : `[ -n "-n" ]` qui est toujours évalué à vrai puisque la longueur de la chaîne "-n" est non nulle.

➔ Conclusion : **toujours encadrer l'accès à la valeur d'une variable (via $) avec des guillemets !**

7.

Variante (vérifier que la chaîne représente un nombre entier) :

```
if [[ ! "$nbr" =~ ^-?[0-9]+$ ]] ; then
    echo Error: not a number
    continue
fi
((somme+=nbr))
```

8.

```
#!/bin/bash
##############################################################################
# entete.sh
# genere un entete dans un script, comprenant
# une description saisie au clavier et rend le script executable
##############################################################################

ligne="#############################################################"

entete='#!'$SHELL
entete=$entete"\n$ligne"
entete="$entete\n# $1"

echo "Donnez la description du script (<Ctrl-D> pour terminer)"
while read line; do
    entete="$entete\n#\t$line"
done

entete="$entete\n# $USER"
entete="$entete\n# $(date +'%d %B %Y')"
entete=$entete"\n$ligne"

echo -e $entete > "$1"
chmod u+x "$1"
```

9.

```
#!/bin/bash
##############################################################################
# lsd.sh
# affiche la liste des répertoires du répertoire courant
##############################################################################

# version avec liste
for f in $(ls); do
    if [ -d "$f" ]; then
        echo "$f"
    fi
done
```

Remarque : ce script ne fonctionnera pas pour lister les répertoires contenant des espaces dans leur nom puisque l'espace est considéré comme un séparateur dans les listes. Mais dans ce cas, l'erreur provient plutôt de l'introduction d'espaces dans des noms de fichiers/répertoires que dans l'écriture du script !

Voici deux versions qui fonctionneront avec des espaces dans les noms de répertoire :

```
    # version avec globbing
    for f in * ; do
        if [ -d "$f" ]; then
            echo -ne "$f\t"   # affichage en ligne
        fi
    done
    echo  # passage à la ligne
```

```
    # version avec lecture sur un pipe
    ls | while read f ; do
        if [ -d "$f" ]; then
            echo -ne "$f\t"
        fi
    done
    echo
```

10.

```
#!/bin/bash
################################################################################*/
# lsd.sh
# affiche la liste des répertoires passés en argument
################################################################################*/

# Variante 1 : avec déplacement vers le répertoire fourni en paramètre

if [ $# -gt 1 ]; then
    echo "Usage: $0 [directory]"
    exit 2
fi

dir=$(pwd)   # sauvegarde du répertoire courant
if [ -n "$1" ]; then
    if [ ! -d "$1" ]; then
        echo "$1 is not a directory"
        exit 1
    fi
    cd $1   # déplacement vers le répertoire $1
fi

for f in $(ls); do
    if [ -d $f ]; then
        echo $f
    fi
done
cd $dir   # retour au répertoire d'exécution du script

# Affichage similaire à ls (en ligne, couleur) :

for f in $(ls); do
    if [ -d $f ]; then
        echo -ne "\e[1;34m$f\t\e[0;39m"   # rép. en gras & bleu, en ligne
    fi
done
echo   # passage à la ligne
```

```
# Variante 2 : avec concaténation des chemins des répertoires trouvés

#     test des conditions avec des if :

if [ $# -eq 0 ]; then
    rep="."
elif [ $# -eq 1 ]; then
    if [ ! -d "$1" ]; then
       echo "$1 n'est pas un repertoire" > /dev/stderr
       exit 1
    fi
    rep=$1
else
    echo "Usage: $0 [directory]" > /dev/stderr
    exit 2
fi

#     OU test des conditions avec un case :

case $# in
0) rep="."
   ;;
1) if [ ! -d "$1" ]; then
    echo "$1 is not a directory" > /dev/stderr
    exit 1
   fi
   rep=$1
   ;;
*) echo "Usage: $0 [directory]" > /dev/stderr
   exit 2
esac

#     affichage des répertoires de rep avec ls :

# affichage des répertoires de rep
for f in $(ls $rep); do
    if [ -d "$rep/$f" ]; then
        echo " $rep/$f"
    fi
done

#     OU affichage des répertoires de rep avec globbing :

lesFichiers="$rep"/*
for f in $lesFichiers; do
    if [ -d "$f" ]; then
        echo " $f"
    fi
done
```

11. & 12.

```
#!/bin/bash
###############################################################################*/
# smartfind.sh
# recherche de fichiers avec une certaine extension et en option une regex
###############################################################################*/

if [ $# -eq 2 ]; then
    find "$1" -name "*.$2" 2> /dev/null
elif [ $# -eq 3 ]; then
    find "$1" -name "*.$2" 2> /dev/null -exec egrep -H -n --color=always "$3" {} \; | less -R
else
    echo Erreur: incorrect number of arguments > /dev/stderr
    echo Usage: $0 directory extension [regex] > /dev/stderr
fi
```

Option `-H` de `egrep` : afficher le chemin du fichier pour chaque correspondance
Option `-n` de `egrep` : afficher le numéro de la ligne dans le fichier
[Options `--color=always` de `egrep` et `-R` de `less` : mise en couleur de chaque correspondance trouvée]