# Compte Rendu - TP 2 - Binôme 22 B-

## Auteurs et date

Clément MORELLI

Thibault GILG

Date : 07/02/2020

## Exercice 1 : Variables d'environnement

1. `PATH` est une variable d'environnement qui définit la liste des répertoire dans lequel chercher un programme à exécuter. Le contenu de cette variable d'environnement est une liste de répertoires séparés par ":" . Pour afficher son contenu, nous avons tapé la commande `printenv PATH` nous permettant d'afficher où trouver les commandes tapées par l'utilisateur. Les commandes tapées par l'utilisateur se trouvent dans les dossiers : `/usr/local/sbin, /usr/local/bin, /usr/sbin, /usr/bin, /sbin, /bin:/usr/games, /usr/local/games,/snap/bin`. B

2. La variable d'environnement `HOME` contient le chemin vers le répertoire personnel. Ainsi, pour nous ramener au répertoire personnel, nous devons taper `cd $HOME` avec le symbole `$` pour récupérer la valeur de cette variable d'environnement.

3. 
   - La variable d'environnement `LANG` contient le paramètre linguistique de base utilisé par les applications du système pour communiquer avec l'utilisateur ex: `fr_FR.UTF-8`.
   - La variable d'environnement `PWD` contient le chemin du répertoire de travail courant de l'interpréteur de commande.
   - La variable d'environnement `OLDPWD` contient le chemin du dernier répertoire de travail courant avant d'avoir tapé la commande `cd` pour changer de répertoire courant.
   - La variable d'environnement `SHELL` contient le chemin vers l'interpréteur de commande préféré de l'utilisateur tel qu'il est défini dans le fichier `/etc/passwd`. Dans notre cas, sa valeur est `/bin/bash` car l'interpréteur de commande préféré est bash.
   - La variable d'environnement `_` correspond au chemin à emprunter pour trouver le script de la commande appelée. Ainsi, lorsqu'on tape `printenv _`, on obtient `/usr/bin/printenv`.

4. Pour créer la variable locale, nous avons tapé la commande `MY_VAR="ab"`. Ensuite nous l'avons affiché avec la commande `echo $MY_VAR` et nous avons obtenu `ab`, donc la variable existe bien.

5. La commande `bash` permet d'ouvrir une autre session bash. Ainsi, la variable locale `MY_VAR` est vide dans l'autre session bash. Lorsqu'on désire afficher sa valeur, on obtient un résultat vide. Cela est du au faite qu'une variable locale n'est pas commune à toute les sessions.

6. Pour transformer `MY_VAR` en une variable d'environnement, nous faisons la commande `export MY_VAR`. Lorsqu'on ouvre une autre session bash, la variable d'environnement existe bien, on peut afficher sa valeur. Cela est du au fait qu'une variable d'environnement existe pour toutes les sessions.

7. On crée la variable d'environnement avec la commande `export NOMS="MORELLI GILG"`. Pour vérifier son contenue on fait la commande `printenv NOMS`.

8. On écrit la commande `echo Bonjour à vous deux $NOMS !` afin d'afficher le message souhaité.

9. La commande `unset` sert à supprimer une variable. La variable n'existe plus, contrairement à une variable vide qui, elle, existe. La commande `printenv` ne renvoie rien.

10. On effectue la commande `echo '$HOME = ' $HOME` pour afficher le message souhaité. On utilise des guillemets simples pour afficher une chaîne littérale et non des guillemets doubles qui permettent d'afficher une chaîne interprétée.

# Programmation Bash

Pour enregistrer nos scripts dans un dossier script dans notre répertoire personnel, nous nous plaçons dans notre répertoire personnel avec la commande`cd` . Puis, nous créons le dossier script avec `mkdir script` . Ainsi, nous pouvons donc nous placé dans ce répertoire avec `cd script` . Pour finir, nous ajoutons le chemin vers script à votre PATH de manière permanente avec `PATH=$PATH:/le_chemin_complet_vers_script` .

## Généralités :

1. **Édition de fichier :** Pour crée chaque fichier nous utilisons la commande nano qui est un éditeur de texte très simple. `nano testpwd.sh`. Ensuite, l'éditeur de texte s'ouvre directement dans la console bash.

2. **En-tête du script :** Chaque script commence par la ligne `#!/bin/bash`. Cette ligne d'en-tête appelle un interpréteur de commande, dans notre cas nous appelons Bash, le shell par défaut dans un système linux. Le *sha-bang* (`#!`) en tête de cette ligne indique à notre système que ce fichier est un ensemble de commandes pour l'interpréteur indiqué, cela correspond à un *nombre magique*. Après le *sha-bang* se trouve le chemin vers le programme qui interprète les commandes de ce script.

## Exercice n°2 : Contrôle de mot de passe

**Objectif :** écrire un script *testpwd.sh* qui demande de saisir un mot de passe et vérifie s'il correspond ou non au contenu d'une variable PASSWORD dont le contenu est codé en dur dans le script. De plus, le mot de passe saisi par l'utilisateur ne doit pas s'afficher.

**Codage :**

Tout d'abord, on crée le fichier testpwd.sh, avec`nano testpwd.sh`. Puis, on commence le script avec la ligne `#!/bin/bash`. Ainsi, nous pouvons commencer à écrire notre script. Nous commençons par écrire une première fonction pour demander à l'utilisateur son mot de passe.

```
function saisirmdp {
        read -p "Saisir le mot de passe : " -s mdp
        echo $mdp
}
```

On commence par définir une fonction que l'on appelle `saisirmdp`. Ensuite, nous utilisons la commande `read` afin de demander à l'utilisateur de rentrer son mot de passe qui sera stocké dans une variable `mdp`. L'option `-p` nous permet d'afficher un texte au moment de l'appel du `read` et `-s` nous permet de faire une

saisie caché du mot de passe. Puis, nous utilisons la commande `echo` afin de récupérer en dehors de cette fonction le mot de passe rentré.

Ensuite, nous écrivons une seconde fonction qui vérifiera si le mot de passe rentré correspondra au mot de passe contenu dans une variable `PASSWORD` dont le contenu est codé en dur dans le script.

```
function verifmdp {
        if [[ $PASSWORD = $1 ]] ;then
                echo -e "\nLe mot de passe est le bon"
        else
                echo -e "\nLe mot de passe n'est pas le bon"
        fi
}
```

On commence par définir la fonction que l'on appelle `verifmdp`. Ensuite, nous créons à l'aide d'un `if` une comparaison entre le contenu de la variable `PASSWORD` codé en dur dans le script et le mot de passe donné en paramètre de la fonction que l'on appelle avec le `$1`. Ainsi, si la comparaison implique l'égalité on affiche à l'aide du `echo` "Le mot de passe est le bon" sinon on affiche "Le mot de passe n'est pas le bon". Avec `echo`, nous utilisons l'option `-e "\ntexte"` afin de revenir à la ligne dans l'affichage sur le bash.

Enfin, nous pouvons passé au programme principal.

```
PASSWORD="azerty"
m=$(saisirmdp)
verifmdp $m
```

On commence par coder en dur dans le script la mot de passe dans une variable appelé `PASSWORD`. Puis, nous stockons dans une variable `m` qui est le résultat de l'appel à la fonction `saisirmdp`, c'est à dire le mot de passe rentré par l'utilisateur. Pour finir, nous appelons la fonction de vérification du mot de passe `verifmdp` en lui donnant en paramètre le contenu de `m` c'est à dire le mot de passe rentré par l'utilisateur.

**Script complet :**

```
#!/bin/bash

function saisirmdp {
        read -p "Saisir le mot de passe : " -s mdp
        echo $mdp
}

function verifmdp {
        if [[ $PASSWORD = $1 ]] ;then
                echo -e "\nLe mot de passe est le bon"
        else
                echo -e "\nLe mot de passe n'est pas le bon"
        fi
}
```

```
PASSWORD="azerty"
m=$(saisirmdp)
verifmdp $m
```

**Résultat console bash :**

```
clements-macbook:Script clement$ ./testpwd.sh
Saisir le mot de passe :
Le mot de passe est le bon
clements-macbook:Script clement$ ./testpwd.sh
Saisir le mot de passe :
Le mot de passe n'est pas le bon
```

## Exercice 3 : Expressions rationnelles

**Objectif :** écrire un script qui prend un paramètre et utilise la fonction donnée pour vérifier que ce paramètre est un nombre réel.

**Codage :**

Tout d'abord, on crée le fichier *ex3.sh*, avec `nano ex3.sh`. Puis, on commence le script avec la ligne `#!/bin/bash`. Ainsi, nous pouvons commencer à écrire notre script.

La fonction donnée dans l'énoncé permet de vérifier si le paramètre rentré en ligne de commande est un nombre. Cette dernière renvoie 1 si c'est le cas et 0 sinon.

```
function is_number()
{
        re='^[+-]?[0-9]+([.][0-9]+)?$'
        if ! [[ $1 =~ $re ]] ; then
            return 1
        else
        return 0
    fi
}
```

Ainsi, il faut utiliser la valeur de retour de cette fonction. Pour cela, nous testons 3 conditions :

- si aucun paramètre n'a été rentré en ligne de commande, on affiche un message d'erreur. Pour cela, nous testons si le nombre de paramètre rentré est strictement inférieur à 1 avec la variable `$#` qui est égale au nombre de paramètres rentrés en ligne de commande.
- si la fonction renvoie 1, on affiche que le paramètre rentré est un réel.
- si la fonction renvoie 0, on affiche que le paramètre rentré n'est pas un réel.

**Script complet :**

```
#!/bin/bash

function is_number()
{
        re='^[+-]?[0-9]+([.][0-9]+)?$'
        if ! [[ $1 =~ $re ]] ; then
                return 1
        else
                return 0
        fi
}

if [[ $# -lt 1 ]] ; then
        echo "Aucun paramètre n'a été rentré"
elif is_number $1 = 1 ; then
        echo "Le paramètre $1 est un réel"
else
        echo "Le paramètre $1 n'est pas un réel"
fi
```

**Résultat console bash :**

```
Clements-Macbook:script clement$ ./stat.sh 123
Le paramètre 123 est un réel
Clements-Macbook:script clement$ ./stat.sh 1ZE
Le paramètre 1ZE n'est pas un réel
Clements-Macbook:script clement$ ./stat.sh
Aucun paramètre n'a été rentré
```

## Exercice 4 : Contrôle d'utilisateur

**Objectif :** écrire un script qui vérifie l'existence d'un utilisateur dont le nom est donné en paramètre du script. Si le script est appelé sans nom d'utilisateur, il affiche le message : "Utilisation : nom_du_script nom_utilisateur", où nom_du_script est le nom de votre script récupéré automatiquement (si vous changez le nom de votre script, le message doit changer automatiquement).

**Codage :**

Tout d'abord, on crée le fichier *ex4.sh*, avec `nano ex4.sh`. Puis, on commence le script avec la ligne `#!/bin/bash`. Ainsi, nous pouvons commencer à écrire notre script.

Nous avons donc créé une fonction `controle_utilisateur()` pour vérifier si l'utilisateur existe bien.

```
controle_utilisateur()
{
        if [[ $# -lt 1 ]] ; then
            echo "Utilisation : $0 nom_utilisateur"
    elif [[ $(cut -f1 -d : /etc/passwd | grep -c $1) -ge 1 ]] ; then
```

```
            echo "L'utilisateur existe"
        else
        echo "L'utilisateur n'existe pas"
    fi
        }
```

Nous avons testé 3 conditions:

- `if [[ $# -lt 1 ]] ; then` pour vérifier que l'utilisateur rentre bien un paramètre. Pour afficher le message demandé qui change automatiquement si le nom du script change lui aussi, nous avons utilisé la commande `echo "Utilisation : $0 nom_utilisateur"`. La variable `$0` correspond au premier élément rentré par l'utilisateur, ici, le nom du programme.
- `elif [[ $(cut -f1 -d : /etc/passwd | grep -c $1) -ge 1 ]] ; then`. La liste des utilisateurs se trouve dans le fichier "/etc/passwd". Cependant, ce fichier contient également d'autres informations. Pour extraire uniquement la liste des utilisateurs, nous avons utiliser la commande `cut -f1 -d :`. Ensuite, nous avons utilisé la commande `grep -c` (renvoie le nombre d'occurrence d'un mot dans un fichier) que nous avons pipé avec la commande précédente afin d'avoir le nombre d'occurrence du paramètre rentré dans la liste des utilisateurs. En testant si ce nombre est supérieur ou égale à 1, nous pouvons déterminer si l'utilisateur rentré existe bien.
- Si l'utilisateur ne figure pas dans la liste, on affiche un message d'erreur.

Nous avons ensuite appelé la fonction pour le paramètre rentré avec la commande `controle_utilisateur $1`.

**Script complet :**

```
#!/bin/bash

controle_utilisateur()
{
        if [[ $# -lt 1 ]] ; then
            echo "Utilisation : $0 nom_utilisateur"
    elif [[ $(cut -f1 -d : /etc/passwd | grep -c $1) -ge 1 ]] ; then
            echo "L'utilisateur existe"
        else
        echo "L'utilisateur n'existe pas"
    fi
        }

controle_utilisateur $1
```

**Résultat console bash :**

```
Clements-Macbook:script clement$ ./stat.sh
Utilisation : ./stat.sh nom_utilisateur
Clements-Macbook:script clement$ ./stat.sh fghj
L'utilisateur n'existe pas
```

```
Clements-Macbook:script clement$ ./stat.sh admin
L'utilisateur existe
```

## Exercice 5 : Factorielle

**Objectif :** écrire un script qui calcule la factorielle d'un entier naturel passé en paramètre (on supposera que l'utilisateur saisit toujours un entier naturel).

**Codage :**

Tout d'abord, on crée le fichier *testpwd.sh*, avec`nano fact.sh`. Puis, on commence le script avec la ligne `#!/bin/bash`. Ainsi, nous pouvons commencer à écrire notre script.

Nous commençons par écrire une première fonction pour demander à l'utilisateurs à quel nombre il veut appliquer factoriel.

```
function entrervaleur {
        read -p "Saisir une valeur : " fact
        echo $fact
}
```

On commence par définir une fonction que l'on appelle `entrervaleur`. Ensuite, nous utilisons la commande `read` afin de demander à l'utilisateur de rentrer le nombre qui sera stocké dans une variable `fact`. L'option `-p` nous permet d'afficher un texte au moment de l'appel de `read` . Puis, nous utilisons la commande `echo` afin de récupérer en dehors de cette fonction le nombre rentré.

Ensuite, nous écrivons une seconde fonction qui calculera le factoriel du nombre rentré par l'utilisateur.

```
function calculfact {
        resultat=1
        fact=$1
        while [ "$fact" -ne 0 ]; do
                let resultat=$resultat*$fact
                let fact=$fact-1
        done
        echo "$1 ! = $resultat"
}
```

On commence par définir la fonction que l'on appelle `calculfact`. Ensuite, nous initialisons le résultat à 1 et nous récupérons dans la variable `fact`le nombre rentré par l'utilisateur que nous avons donné en paramètre de la fonction avec le `$1`. Nous créons à l'aide d'un `while` une boucle qui s'applique tant que le nombre rentré par l'utilisateur (`fact`) est différent de 0 (0!=1) . Ainsi, dans la boucle on commence par multiplier le résultat par la valeur de `fact`. Puis, on enlève 1 à la valeur de `fact`. Lorsque `fact` est égale à 0 on sort de la boucle et on affiche le calcul et la valeur du résultat avec `echo "$1 ! = $resultat"`. La commande `let` est une des commandes permettant d'effectuer des calculs.

Enfin, nous pouvons passer au programme principal.

```
val=$(entrervaleur)
calculfact $val
```

On commence par stocker dans une variable `val` le résultat de l'appel à la fonction `entrervaleur` c'est à dire le nombre rentré par l'utilisateur. Pour finir, nous appelons la fonction de calcul de factoriel `calculfact` en lui donnant en paramètre le contenu de `val` c'est à dire le nombre rentré par l'utilisateur.

**Script complet :**

```
#!/bin/bash

function entrervaleur {
        read -p "Saisir une valeur : " fact
        echo $fact
}

function calculfact {
        resultat=1
        fact=$1
        while [ "$fact" -ne 0 ]; do
                let resultat=$resultat*$fact
                let fact=$fact-1
        done
        echo "$1 ! = $resultat"
}

val=$(entrervaleur)
calculfact $val
```

**Résultat console bash :**

```
clements-macbook:script clement$ ./fact.sh
Saisir une valeur : 5
5 ! = 120
```

## Exercice n°6 : Le juste prix

**Objectif :** écrire un script qui génère un nombre aléatoire entre 1 et 1000 et demande à l'utilisateur de le deviner. Le programme écrira "C'est plus !", "C'est moins !" ou "Gagné !" selon les cas.

**Codage :**

Tout d'abord, on crée le fichier *justeprix.sh*, avec `nano justeprix.sh`. Puis, on commence le script avec la ligne `#!/bin/bash`. Ainsi, nous pouvons commencer à écrire notre script. Nous commençons par écrire une première fonction pour demander à l'utilisateur quel nombre il veut rentrer.

```
function entrerval {
        read -p "Saisir une valeur : " val
        echo $val
}
```

On commence par définir une fonction que l'on appelle `entrerval`. Ensuite, nous utilisons la commande `read` afin de demandé à l'utilisateur de rentrer le nombre qui sera stocké dans une variable `val`. L'option `-p` nous permet d'afficher un texte au moment de l'appel de `read` . Puis, nous utilisons la commande `echo` afin de récupérer en dehors de cette fonction le nombre rentré.

Ensuite, nous écrivons une seconde fonction qui calculera le factoriel du nombre rentré par l'utilisateur.

```
function verif {
        gagne=0
        while [ $gagne -eq 0 ]; do
                tentative=$(entrerval)
                if [[ $tentative -eq $JP ]]; then
                        echo "C'est gagné"
                        gagne=1
                fi
                if [[ $tentative -lt $JP ]]; then
                        echo "C'est plus"
                fi
                if [[ $tentative -gt $JP ]]; then
                    echo "C'est moins"
        fi
        done
}
```

On commence par définir la fonction que l'on appelle `verif`. Ensuite, nous initialisons la variable gagne à 0. Nous créons à l'aide d'un `while` une boucle qui s'applique tant que *gagne* est égale à 0. Ainsi, dans la boucle on commence par récupérer dans la variable `tentative` qui est le résultat de l'appel à la fonction `entrerval` c'est à dire le nombre rentré par l'utilisateur. Ainsi, à chaque passage dans la boucle l'utilisateur rentre une nouvelle tentative. Ensuite, avec 3 `if` on compare sa tentative au juste prix `JP` dont le contenu est codé en dur dans le script. Il y aura donc un `if` qui regarde si la tentative rentrée est égale au juste prix, si c'est le cas on affiche à l'aide de la commande `echo` "C'est gagné" et on passe la gagne à 1 afin de sortir de la boucle `while`. Un autre `if` regarde si la tentative rentrée est supérieure au juste prix, si c'est le cas on affiche à l'aide de la commande `echo` "C'est moins" . Enfin, le dernier `if` regarde si la tentative rentré est inférieur au juste prix, si c'est le cas on affiche à l'aide de la commande `echo` "C'est plus" .

Enfin, nous pouvons passé au programme principal.

```
JP=$(( $RANDOM % 1000 ))
verif
```

On commence par stocker dans une variable `JP` le juste prix c'est à dire le nombre que l'utilisateur devra trouver qui est générer aléatoirement à l'aide de RANDOM, la séquence `% 1000` nous permets de choisir un nombre aléatoire compris entre 0 et 1000. Pour finir, nous appelons la fonction de recherche du juste prix `verif` qui demandera une tentative à l'utilisateur jusqu'à ce qu'il trouve.

**Script complet :**

```
#!/bin/bash

function entrerval {
        read -p "Saisir une valeur : " val
        echo $val
}

function verif {
        gagne=0
        while [ $gagne -eq 0 ]; do
                tentative=$(entrerval)
                if [[ $tentative -eq $JP ]]; then
                        echo "C'est gagné"
                        gagne=1
                fi
                if [[ $tentative -lt $JP ]]; then
                        echo "C'est plus"
                fi
                if [[ $tentative -gt $JP ]]; then
                    echo "C'est moins"
        fi
        done
}

JP=$(( $RANDOM % 1000 ))
verif
```

**Résultat console bash :**

```
clements-macbook:script clement$ ./justeprix.sh
Saisir une valeur : 1
C'est plus
Saisir une valeur : 5
C'est moins
Saisir une valeur : 3
C'est gagné
```

## Exercice n°7 : Statistiques

1. **Objectif :** écrire un script qui prend en paramètres trois entiers (entre -100 et +100) et affiche le min, le max et la moyenne.

**Codage :**

Tout d'abord, on crée le fichier *stat1.sh*, avec`nano stat1.sh`. Puis, on commence le script avec la ligne `#!/bin/bash`. Ainsi, nous pouvons commencer à écrire notre script. Nous commençons par écrire une première fonction pour vérifier les nombres passés en paramètres.

```
function is_not_entier {
        re='^[+-]?[0-9]+$'
        if ! [[ $1 =~ $re ]] ; then
                echo "Ce n'est pas un entier"
                exit
        fi
        if [[ 100 -lt $1 || $1 -lt -100 ]]; then
                echo "Nombre non compris entre -100 et 100"
                exit
        fi
}
```

On commence par définir une fonction que l'on appelle `is_not_entier`. Ensuite, avec un `if`, on compare le nombre passé en paramètre de la fonction à une variable `re` qui symbolise un entier positif ou négatif. Ainsi, si le nombre n'en est pas un, on affiche avec `echo` "Ce n'est pas un entier" et on quitte le programme avec la commande `exit` sinon, on va au second `if` qui lui vérifie que le nombre passé en paramètre de la fonction est compris entre -100 et 100 si ce n'est pas le cas on affiche avec `echo` "Nombre non compris entre -100 et 100" et on quitte le programme avec la commande `exit`. `||` équivaut à un OU logique.

Ensuite, nous écrivons une seconde fonction qui calculera la moyenne des 3 nombres rentrés par l'utilisateur.

```
function moyenne {
        moyenne=0
        moyenne=$(echo "( $moyenne + $1 + $2 + $3 ) / 3" | bc -l )
        echo "la moyenne de ces trois chiffres est : $moyenne"
}
```

On commence donc par initialiser la moyenne à 0. Puis, on envoie à l'aide de la commande `echo` la chaîne de caractère qui symbolise le calcul de la moyenne en additionnant les 3 paramètres donnés à la fonction et en divisant par 3 à la commande `bc` qui permet de faire des calculs. On ajoute `-l` afin de gérer une moyenne qui ne serait pas entière. Puis, on affiche la moyenne à l'aide de la commande `echo`.

Ensuite, nous écrivons une troisième fonction qui calculera le maximum des 3 nombres rentrés par l'utilisateur.

```
function max {
        max=$1
        if [[ $max -lt $2 ]]; then
                max=$2
        fi
        if [[ $max -lt $3 ]]; then
```

```
                max=$3
        fi
        echo "la maximum de ces chiffres est : $max"
}
```

On commence donc par initialiser le maximum à la valeur du premier paramètre. Ensuite, on regarde avec un `if` si le second paramètre est plus grand et si c'est le cas, il devient le nouveau maximum. Puis, avec un nouveau `if`, on regarde si le troisième paramètre est plus grand et si c'est le cas, il devient le nouveau maximum. Enfin, on affiche à l'aide de la commande `echo` le maximum.

Ensuite, nous écrivons une quatrième fonction qui calculera le minimum des 3 nombres rentrés par l'utilisateur.

```
function min {
        min=$1
        if [[ $2 -lt $min ]]; then
                min=$2
        fi
        if [[ $3 -lt $min ]]; then
                min=$3
        fi
        echo "le minimum de ces trois chiffres est : $min"
}
```

On commence donc par initialiser le minimum à la valeur du premier paramètre. Ensuite, on regarde avec un `if` si le second paramètre est plus petit et si c'est le cas, il devient le nouveau minimum. Puis, avec un nouveau `if` on regarde si le troisième paramètre est plus petit et si c'est le cas, il devient le nouveau minimum. Enfin, on affiche à l'aide de la commande `echo` le minimum.

Enfin, nous pouvons passer au programme principal.

```
a=$1
is_not_entier $a
b=$2
is_not_entier $b
c=$3
is_not_entier $c
moyenne $a $b $c
max $a $b $c
min $a $b $c
```

On récupère les 3 paramètres dans `a,b,c` tout en vérifiant bien qu'ils sont des entiers positifs ou négatifs et compris entre -100 et 100. Ensuite, on appelle les fonctions `moyenne, max, min` en leur donnant ces trois nombres en paramètres.

**Script complet :**

```
#!/bin/bash

function is_not_entier {
        re='^[+-]?[0-9]+$'
        if ! [[ $1 =~ $re ]] ; then
                echo "Ce n'est pas un entier"
                exit
        fi
        if [[ 100 -lt $1 || $1 -lt -100 ]]; then
                echo "Nombre non comprit entre -100 et 100"
                exit
        fi
}

function moyenne {
        moyenne=0
        moyenne=$(echo "( $moyenne + $1 + $2 + $3 ) / 3" | bc -l )
        echo "la moyenne de ces trois chiffres est : $moyenne"
}

function max {
        max=$1
        if [[ $max -lt $2 ]]; then
                max=$2
        fi
        if [[ $max -lt $3 ]]; then
                max=$3
        fi
        echo "la maximum de ces chiffres est : $max"
}

function min {
        min=$1
        if [[ $2 -lt $min ]]; then
                min=$2
        fi
        if [[ $3 -lt $min ]]; then
                min=$3
        fi
        echo "le minimum de ces trois chiffres est : $min"
}

a=$1
is_not_entier $a
b=$2
is_not_entier $b
c=$3
is_not_entier $c
moyenne $a $b $c
max $a $b $c
min $a $b $c
```

**Résultat console bash :**

```
clements-macbook:script clement$ ./stat1.sh 12 12 13
la moyenne de ces trois chiffres est : 12.33333333333333333
la maximum de ces chiffres est : 13
le minimum de ces trois chiffres est : 12
clements-macbook:script clement$ ./stat1.sh 12 a 78
Ce n'est pas un entier
clements-macbook:script clement$ ./stat1.sh 12 234 78
Nombre non comprit entre -100 et 100
clements-macbook:script clement$ ./stat1.sh 12 -234 78
Nombre non comprit entre -100 et 100
```

2. **Objectif 2 :** Généraliser le programme à un nombre quelconque de paramètres.

**Codage :**

Tout d'abord, on crée le fichier *stat2.sh*, avec`nano stat2.sh`. Puis, on commence le script avec la ligne `#!/bin/bash`. Ainsi, nous pouvons commencer à écrire notre script. Nous commençons par écrire une première fonction pour vérifier les nombres passé en paramètres. C'est la même fonction que `is_not_entier`.

Enfin, nous pouvons passé au programme principal. On commence par initialisé nos variables.

```
indice=0
i=0
nb=$#
sum=0
moyenne=0
max=$
min=$1
```

- `$#` ressort le nombre de paramètres.
- `max=$1, min=$1` initialise max et le min au premier parametre.
- `sum=0` initialise la somme à 0.

Ensuite, nous pouvons passer à la boucle `while`.

```
while [ $indice -lt $nb ]; do
        is_not_entier $1
        let sum=$(( $sum + $1 ))
        shift
        if [[ $max -lt $1 ]]; then
                max=$1
        fi
        if [[ $1 -lt $min ]]; then
                min=$1
        fi
```

```
        let indice=$(( $indice + 1 ))
done
```

On réalise cette boucle tant qu'il reste des paramètres à traiter. On commence donc avec le premier paramètre en vérifiant que c'est un entier compris entre -100 et 100 avec la fonction `is_not_entier`. Puis, on ajoute à la variable `sum` ce nombre. Ensuite, on fait la commande `shift` qui va alors décaler les paramètres vers la gauche c'est-à-dire que le paramètre 2 devient le 1, le 3 devient le 2 etc... Cela nous permet de comparer l'ancien paramètre 1 au nouveau pour déterminer avec des `if` qui est le plus petit, le plus grand des deux. Enfin, on augmente l'indice de 1. On réalise cette boucle autant de fois qu'il y a de paramètres.

Ensuite, on calcule la moyenne et on affiche le maximum, le minimum et la moyenne avec la commande`echo`.

```
moyenne=$(echo "$sum / $nb" | bc -l )
echo "la moyenne de ces trois chiffres est : $moyenne"
echo "la maximum de ces chiffres est : $max"
echo "la minimum de ces chiffres est : $min"
```

**Script complet :**

```
#!/bin/bash

function is_not_entier {
        re='^[+-]?[0-9]+$'
        if ! [[ $1 =~ $re ]] ; then
                echo "Ce n'est pas un entier"
                exit
        fi

        if [[ 100 -lt $1 || $1 -lt -100 ]]; then
                echo "Nombre non comprit entre -100 et 100"
                exit
        fi
}

indice=0
nb=$#
declare -a NB=()
sum=0
moyenne=0
max=$1
min=$1
while [ $indice -lt $nb ]; do
        is_not_entier $1
        NB[${#NB[*]}]=$1
        let sum=$(( $sum + $1 ))
        shift
```

```
        if [[ $max -lt $1 ]]; then
                max=$1
        fi
        if [[ $1 -lt $min ]]; then
                min=$1
        fi
        let indice=$(( $indice + 1 ))
done
moyenne=$(echo "$sum / $nb" | bc -l )
echo "la moyenne de ces trois chiffres est : $moyenne"
echo "la maximum de ces chiffres est : $max"
echo "la minimum de ces chiffres est : $min"
```

```
**Résultat console bash :**
clements-macbook:script clement$ ./stat2.sh 12 12 13
la moyenne de ces trois chiffres est : 12.33333333333333333
la maximum de ces chiffres est : 13
la minimum de ces chiffres est : 12
clements-macbook:script clement$ ./stat2.sh 12 12 13 45 32
la moyenne de ces trois chiffres est : 22.80000000000000000000
la maximum de ces chiffres est : 45
la minimum de ces chiffres est : 12
```

3. **Objectif 3 :** modifier le programme pour que les notes ne soient plus données en paramètres, mais saisies et stockées au fur et à mesure dans un tableau.

**Codage :**

Tout d'abord, on crée le fichier *stat3.sh*, avec `nano stat3.sh`. Puis, on commence le script avec la ligne `#!/bin/bash`. Ainsi, nous pouvons commencer à écrire notre script. Nous commençons par écrire une première fonction pour vérifier les nombres passés en paramètres.

```
function is_not_entier {
        re='^[+-]?[0-9]+$'
        if ! [[ $1 =~ $re ]] ; then
                echo "Ce n'est pas un entier"
                exit
        fi
        if [[ 100 -lt $1 || $1 -lt -100 ]]; then
                echo "Nombre non compris entre -100 et 100"
                exit
        fi
        echo "$1"
}
```

C'est la même fonction que `is_not_entier` avec un petit ajout. En effet, il y a toujours les deux `if`. Le premier `if` vérifie que c'est un entier positif ou négatif, le second vérifie si il est compris entre -100 et 100. On

ajoute `echo "$1"` afin que si les deux `if` ne sont pas vérifiés, c'est-à-dire que si l'on a bien un entier compris entre -100 et 100, on puisse récupérer en dehors de cette fonction le nombre rentré.

Ensuite, nous écrivons une seconde fonction pour demander à l'utilisateurs à quel nombre il souhaite rentrer.

```
function entrerval {
        read -p "Saisir une valeur : " val
        is_not_entier $val
}
```

On commence par définir une fonction que l'on appelle `entrerval`. Ensuite, nous utilisons la commande `read` afin de demander à l'utilisateur de rentrer le nombre qui sera stocké dans une variable `val`. L'option `-p` nous permet d'afficher un texte au moment de l'appel du `read` . Puis, nous utilisons la commande `is_not_entier $val` afin d'appeler la fonction `is_not_entier` pour vérifier que le contenu saisi par l'utilisateur répond a nos critères énumérés précédemment.

Ensuite, nous écrivons une troisième fonction pour demander à l'utilisateur combien de nombre il souhaite rentrer.

```
function cb {
        read -p "Saisir le nombre de valeur que vous voulez rentrer : " a
        echo $a
}
```

On commence par définir une fonction que l'on appelle `cb`. Ensuite, nous utilisons la commande `read` afin de demander à l'utilisateur de rentrer le nombre qui sera stocké dans une variable `a`. L'option `-p` nous permet d'afficher un texte au moment de l'appel de`read` . Puis, nous utilisons la commande `echo` afin de récupérer en dehors de cette fonction le nombre rentré.

Enfin, nous pouvons passer au programme principal qui reste le même à quelques exceptions près. On commence par initialiser nos variables.

```
indice=0
declare -a NB=()
sum=0
moyenne=0
max=-101
min=101
nb=$(cb)
```

- `$#` ressort le nombre de paramètres.
- `max=$1, min=$1` initialise max et le min à des nombres qui sont respectivement inférieur et supérieur à ce qu'ils peuvent être.
- `sum=0` initialise la somme à 0.
- `declare -a NB=()` crée un tableau NB vide

- `nb=$(cb)` : on stocke dans une variable `nb` le résultat de l'appel à la fonction `cb`, c'est-à-dire le nombre de nombre que l'on veut rentrer.

Ensuite, nous pouvons passer à la boucle `while`.

```
while [ $indice -lt $nb ]; do
        e=$(entrerval)
        NB[${#NB[*]}]=$e
        let sum=$(( $sum + $e ))
        if [[ $max -lt $e ]]; then
                max=$e
        fi
        if [[ $1 -lt $min ]]; then
                min=$e
        fi
        let indice=$(( $indice + 1 ))
done
```

On réalise cette boucle tant que l'utilisateur n'a pas rentrer tous les nombres qu'il voulait rentrer. On commence donc par stocker dans une variable `e` le résultat de l'appel à la fonction `enterval` c'est-à-dire le nombre rentré par l'utilisateur vérifiant les conditions requises. Puis, on ajoute à la variable `sum` ce nombre, on l'ajoute aussi au tableau crée. Ensuite, on compare avec un `if` l'ancien maximum au nombre rentré par l'utilisateur pour déterminer le plus grand des deux. On compare aussi avec un `if` l'ancien minimum au nombre rentré par l'utilisateur pour déterminer le plus petit des deux. Enfin, on augmente l'indice de 1. On réalise cette boucle autant de fois que l'utilisateur veut rentrer de nombre.

Ensuite, on calcule la moyenne et on affiche le maximum, le minimum et la moyenne avec la commande `echo`.

```
moyenne=$(echo "$sum / $nb" | bc -l )
echo "la moyenne de ces trois chiffres est : $moyenne"
echo "la maximum de ces chiffres est : $max"
echo "la minimum de ces chiffres est : $min"
```

**Script complet :**

```
#!/bin/bash

function is_not_entier {
        re='^[+-]?[0-9]+$'
        if ! [[ $1 =~ $re ]] ; then
                echo "Ce n'est pas un entier"
                exit
        fi
        if [[ 100 -lt $1 || $1 -lt -100 ]]; then
                echo "Nombre non comprit entre -100 et 100"
                exit
```

```
        fi
        if [[ $1 =~ $re ]] ; then
                echo "$1"
        fi
}

function entrerval {
        read -p "Saisir une valeur : " val
        is_not_entier $val
}

function cb {
        read -p "Saisir le nombre de valeur que vous voulez rentrer : " a
        echo $a
}

indice=0
declare -a NB=()
sum=0
moyenne=0
max=$1
min=101
nb=$(cb)

while [ $indice -lt $nb ]; do
        e=$(entrerval)
        NB[${#NB[*]}]=$e
        let sum=$(( $sum + $e ))
        if [[ $max -lt $e ]]; then
                max=$e
        fi
        if [[ $e -lt $min ]]; then
                min=$e
        fi
        let indice=$(( $indice + 1 ))
done

moyenne=$(echo "$sum / $nb" | bc -l )
echo "la moyenne de ces trois chiffres est : $moyenne"
echo "la maximum de ces chiffres est : $max"
echo "la minimum de ces chiffres est : $min"
```

**Résultat console bash :**

```
clements-macbook:script clement$ ./stat3.sh
Saisir le nombre de valeur que vous voulez rentrer : 3
Saisir une valeur : 32
Saisir une valeur : 56
Saisir une valeur : 65
la moyenne de ces trois chiffres est : 51.00000000000000000000
la maximum de ces chiffres est : 65
la minimum de ces chiffres est : 32
```