

# TP 2 - Bash

## Exercice 1. Variables d'environnement

1. Dans quels dossiers bash trouve-t-il les commandes tapées par l'utilisateur?

Dossiers indiqué par la variable PATH

Les commandes tapées par l'utilisateur sont stockées dans les fichiers liés au PATH, bash va aller dans chacun de ces fichiers successivement jusqu'à trouver la commande : on utilise la commande "printenv PATH" pour les trouver. On obtient alors :

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
```

2. Quelle variable d'environnement permet à la commande cd tapée sans argument de vous ramener dans votre répertoire personnel?

Si on tape cd sans argument, la valeur de la variable shell HOME est utilisée pour nous ramener à notre dossier personnel.

3. Explicitiez le rôle des variables LANG, PWD, OLDPWD, SHELL et \_.

LANG : stocke la langue utilisée sur le système PWD : stocke le chemin d'accès au repertoire **courant**

OLDPWD : stocke le chemin d'accès au repertoire visité précédemment SHELL : stocke le type de

shell utilisé \_ : stocke le dernier argument envoyé en entrée

**par défaut**

**de la commande précédente**

4. Créez une variable locale MY\_VAR(le contenu n'a pas d'importance). Vérifiez que la variable existe.

On crée MY\_VAR avec "MY\_VAR="variable", puis on exécute "echo \$MY\_VAR" pour vérifier son existence : on obtient en sortie "variable".

5. Tapez ensuite la commande bash. Que fait-elle? La variable MY\_VAR existe-t-elle? Expliquez. A la fin de cette question, tapez la commande exit pour revenir dans votre session initiale.

La commande bash sert à créer un nouveau shell. Dans ce nouveau shell la variable MY\_VAR n'existe plus car c'est une variable locale.

6. Transformez MY\_VAR en une variable d'environnement et recommencez la question précédente. Expliquez.

Pour transformer MY\_VAR en une variable d'environnement (= globale), on exécute "export MY\_VAR="testvar" && printenv MY\_VAR". Si on recrée un nouveau shell, on peut cette fois-ci afficher le contenu de MY\_VAR grâce à la commande echo car c'est désormais une variable d'environnement.

7. Créer la variable d'environnement NOMS ayant pour contenu vos noms de binômes séparés par un espace. Afficher la valeur de NOMS pour vérifier que l'affectation est correcte.

Même principe que la question précédente : on exécute "export NOMS='BLAISE DEROT' && printenv NOMS" puis on affiche sa valeur grâce à "echo \$NOMS" : on obtient en sortie "BLAISE DEROT".

8. Ecrivez une commande qui affiche "Bonjour à vous deux, binôme1 binôme2!" (où binôme1 et binôme2 sont vos deux noms) en utilisant la variable NOMS.

On execute "echo "Bonjour à vous deux, \$NOMS !" et on obtient en sortie "Bonjour à vous deux, BLAISE DEROT".

9. Quelle différence y a-t-il entre donner une valeur vide à une variable et l'utilisation de la commande unset?

Si on donne une valeur vide à une variable elle existe toujours et reste modifiable, mais si on utilise la commande unset, la variable n'existe plus et n'est plus stockée en mémoire.

10. Utilisez la commande echo pour écrire exactement la phrase :\$HOME =chemin(où chemin est votre dossier personnel d'après bash)

On utilise un antislash pour afficher \$HOME en brut sans afficher le chemin d'accès au repertoire personnel, on execute donc "echo "\$HOME = \$HOME". **Disparu du pdf l'?**

## Programmation Bash

### Exercice 2. Contrôle du mot de passe

Écrivez 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. Le mot de passe saisi par l'utilisateur ne doit pas s'afficher.

```
#!/bin/bash

vraimdp=roroledozo
read -s -p 'Entrez votre mot de passe : ' mdp

if [ $mdp = $vraimdp ]; then
    echo "Le mot de passe correspond"
else
    echo "Le mot de passe ne correspond pas"
fi
```

si mdp est vide => erreur de syntaxe  
tester plutôt x\$mdp = x\$vraimdp  
si mdp est vide alors ça teste:  
x = xroroledozo

Ici on demande à l'utilisateur d'entrer son mot de passe en entrée grâce à la commande 'read', il sera alors stocké dans 'mdp' et non affiché sur le shell grâce au '-s'. On vérifie ensuite s'il correspond au mot de passe de l'utilisateur stocké dans 'vraimdp'.

### Exercice 3. Expressions rationnelles

Ecrivez un script qui prend un paramètre et utilise la fonction suivante pour vérifier que ce paramètre est un nombre réel, Il affichera un message d'erreur dans le cas contraire.

```
#!/bin/bash
function nombre_reel {
    re='^[+-]?[0-9]+([.][0-9]+)?$'
    if ! [[ $1 =~ $re ]] ; then
        return 1
    else
```

```

        return 0
    fi
}

nombre_reel $1
output = $?

if [ $output == 0 ]; then
    echo "C'est un nombre réel"
else
    echo "ERREUR : Ce n'est pas un nombre réel"
fi

```

plus simple de faire `if nombre_reel $1; then`

La fonction `nombre_reel()` fait le travail de vérification pour déterminer si le paramètre est un nombre réel ou non. Il suffit ensuite de vérifier si la sortie de cette fonction est 0 ou 1 pour conclure sur la nature du paramètre.

## Exercice 4. Contrôle d'utilisateur

Écrivez 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)

```

#!/bin/bash

if [ $# = 0 ]; then
    echo "Utilisation : $0 'nom_utilisateur' "
else
    if [ -z "$(cut -d: -f1 /etc/passwd | grep -x $1 )" ] ; then
        echo "L'utilisateur entré n'existe pas"
    else
        echo "L'utilisateur entré existe"
    fi
fi

```

Ici, Si l'utilisateur n'entre aucun paramètre on le détecte avec `'$#'` qui stocke le nombre de paramètres entrés, sinon on cherche si l'utilisateur est dans les fichiers d'utilisateurs pour conclure sur l'existence ou non de l'utilisateur entré en paramètre.

## Exercice 5. Factorielle

Écrivez un programme qui calcule la factorielle d'un entier naturel passé en paramètre (on supposera que l'utilisateur saisit toujours un entier naturel).

```

#!/bin/bash

function factorielle {

```

```

parametre=$1
if [ $parametre -eq 0 ]; then
    echo 1
else
    echo $(( $parametre * factorielle $(( $parametre - 1 )) ))
fi
}
echo "Résultat : $(factorielle $1)"

```

Bourrin! chaque appel a factorielle crée un shell pour exécuter la fonction  
Une simple boucle fait l'affaire pas besoin de récursivité pour ça

On se sert ici d'une fonction recursive pour calculer la factorielle d'un nombre. Si c'est 0 on renvoie directement 1, sinon on rappelle la fonction factorielle avec n-1 jusqu'à arriver à factorielle 0, qui renvoie 1, puis on remonte jusqu'à retourner a factorielle n, pour enfin afficher le resultat du calcul.

## Exercice 6. Le juste prix

Écrivez 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 (vous utiliserez \$RANDOM).

```

justePrix(proposition, nombre) {
    if [ $proposition -gt $nombre ]
    then
        echo "C'est moins !"
    else
        echo "C'est plus !"
    fi
}

proposition=-1
nombre=$(( $RANDOM % 1000 + 1 ))

while [ $proposition -ne $nombre ]
do
    read -p 'Saisissez un nombre entre 1 et 1000 : ' proposition
    justePrix(proposition, nombre)
done

echo "Gagné !"

```

une fonction ne s'écrit pas comme ça  
CF vos autres codes...

aucune chance que proposition et nombre soit identique puisqu'il s'agit de deux mots différents  
Un appel de fonction:  
~~justePrix \$proposition \$nombre~~

Dans ce script, on initialise un nombre aléatoire entre 1 et 1000, puis on se sert d'une boucle while pour appeler en boucle la fonction 'justePrix' qui renvoie "C'est plus" ou "C'est moins", tant que l'utilisateur ne trouve pas le nombre recherché.

## Exercice 7. Statistiques

1. Écrivez un script qui prend en paramètres trois entiers (entre -100 et +100) et affiche le min, le max et la moyenne. Vous pouvez réutiliser la fonction de l'exercice 3 pour vous assurer que les paramètres sont bien des entiers.

2. Généralisez le programme à un nombre quelconque de paramètres (pensez à SHIFT)
3. Modifiez votre programme pour que les notes ne soient plus données en paramètres, mais saisies et stockées au fur et à mesure dans un tableau.
- 4.

```
#!/bin/bash

for parametre in "$@"; do
    if [ $parametre -lt -100 ] || [ $parametre -gt 100 ] ; then
        echo "Paramètres invalides"
        exit
    fi
done

MOYENNE=$(( $(($1 + $2 + $3)) / 3 ))
echo "La moyenne est $MOYENNE"

MINIMUM=$1
for parametre in "$@"; do
    if [ $parametre -lt $MINIMUM ]; then
        MINIMUM=$parametre
    fi
done
echo "Le minimum est $MINIMUM"

MAXIMUM=$1
for parametre in "$@"; do
    if [ $param -gt $MAXIMUM ]; then
        MAXIMUM=$parametre
    fi
done
echo "Le maximum est $MAXIMUM"
```

Min et Max pourrait être fait en un seul parcours de boucle.

Tout d'abord on vérifie si les 3 paramètres entrés sont bien des nombres entre -100 et 100, sinon on renvoie "Paramètres invalides". Pour la moyenne on effectue un simple calcul de moyenne avec les 3 paramètres d'entrées et pour le maximum et le minimum, on fixe le 1er paramètre comme étant min (respectivement max) et on vérifie si les deux autres paramètres sont supérieurs (respectivement inférieurs) au premier. Si oui on fixe ce paramètre comme min (respectivement max).

2.

```
#!/bin/bash

for parametre in "$@"; do
    if [ $parametre -lt -100 ] || [ $parametre -gt 100 ] ; then
        echo "Paramètres invalides"
        exit
    fi
done
```

```
done

somme=0
for parametre in "$@"; do
    somme=$(( somme+$parametre ))
done

MOYENNE=$(( $(($somme)) / $# ))
echo "La moyenne est $MOYENNE"

MINIMUM=$1
for parametre in "$@"; do
    if [ $parametre -lt $MINIMUM ] ; then
        MINIMUM=$parametre
    fi
done
echo "Le minimum est $MINIMUM"

MAXIMUM=$1
for parametre in "$@"; do
    if [ $parametre -gt $MAXIMUM ] ; then
        MAXIMUM=$parametre
    fi
done
echo "Le maximum est $MAXIMUM"
```

Les trois opérations peuvent être  
fait en une seul boucle

Ici on modifie uniquement le calcul de la moyenne, on recupere le nombre de paramètres grace a '\$#' et on procède de la meme manière que précédemment.