


A

TP 2 - Bash

Exercice 1. Variables d'environnement

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

Les commandes tapées par un utilisateur se trouvent dans un des dossiers suivants:

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr^{}es:/usr/local/games:/snap/bin. Bash explore successivement ces dossiers jusqu'à trouver la commande.

On utilise la commande `printenv PATH` pour connaître ces dossiers. La variable d'environnement `PATH` contient les chemins des répertoires contenant les commandes potentiellement tapées par un utilisateur.

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

Grâce à la commande `env` on a accès à la liste des variables d'environnement. On remarque la variable `HOME` contient le chemin du répertoire personnel.

3. Expliquez le rôle des variables `LANG`, `PWD`, `OLDPWD`, `SHELL` et `_`.

`LANG` : indique la langue qui est utilisée par les logiciels et les utilisateurs

`PWD` : indique l'emplacement du dossier courant

`OLDPWD` : indique l'emplacement du dossier courant précédent

`SHELL` : renvoie l'interpréteur shell utilisé : pour nous il s'agit de bash (avec `echo $SHELL` on obtient `/bin/bash`)

`_` : renvoie le dernier argument passé

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

`MY_VAR="bla"` Il s'agit de la variable `MY_VAR` contenant la chaîne "bla"

Pour afficher le contenu de la variable locale : `echo $MY_VAR`

Note : `printenv MY_VAR` ne peut pas fonctionner car `MY_VAR` est une variable locale, et pas une variable d'environnement.

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` crée un nouveau shell. Dans ce shell, `MY_VAR` n'existe plus : elle n'a pas été créée à ce niveau, c'est une variable locale du niveau précédent. `exit` nous permet de rejoindre le shell initial dans lequel `MY_VAR` est toujours présente. On peut connaître le niveau de son shell actuel grâce à `echo $SHLVL`.

6. Transformez `MY_VAR` en une variable d'environnement et recommencez la question précédente. Expliquez.

On transforme `MY_VAR` en variable d'environnement grâce à `export MY_VAR`. Lorsqu'on crée un nouveau `SHELL`, cette variable est conservée car elle est de configuration globale : elle est présente dans tous les `SHELL` jusqu'à sa suppression.

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.

`export NOMS=Bataillon Moreau` crée une variable d'environnement NOMS qui contient uniquement Bataillon. Il a été considéré que les 2 noms étaient des arguments différents. Il faut mettre des cotes ou des guillemets pour indiquer qu'il s'agit d'un seul argument.

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.

```
export NOMS="Bataillon Moreau"
echo Bonjour à vous deux, $NOMS
```

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

Une variable vide existe mais ne contient rien. La commande `unset` détruit la variable et son contenu.

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

`echo '$HOME = '$HOME` Les cotes permettent à bash de ne pas interpréter ce qui s'y trouve.

Programmation Bash

Ajoutez le chemin vers script à votre PATH de manière permanente :

à la fin du fichier `~/ .bashrc`, ajouter: `PATH=$PATH:~/script`

Exercice 2. Contrôle de 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.

Script:

```
#!/bin/bash
PASSWORD="tp"
read -s -p 'Saisissez votre mot de passe : ' mdp
if [ "$mdp" = "$PASSWORD" ]; then
    echo "Le mot de passe correspond"
else
    echo "Le mot de passe ne correspond pas"
fi
```

Dans `PASSWORD`, on stocke le mot de passe témoin. On demande à l'utilisateur de remplir un mot de passe grâce à la commande `read`. On ajoute en option de `read` `-p` pour afficher un message (ici: *Saisissez votre mot de passe* :) et `-s` pour que le mot de passe saisi par l'utilisateur ne s'affiche pas.

Dans le cas où la chaîne rentrée par l'utilisateur correspond à la chaîne rentrée dans `PASSWORD` : le mot de passe correspond. On utilise `$` devant les variables pour avoir accès à leur contenu.

On compile avec `chmod u+x testpwd.sh`

On lance le script en appelant `testpwd.sh`

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 :

Script :

```
#!/bin/bash

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

is_number $1
if [ $? -eq 1 ];then
  >> echo "erreur : pas réel"
else
  >> echo "réel"
fi
```

On crée la fonction et on l'appelle avec en paramètre le premier paramètre saisi par l'utilisateur (\$1).

\$? : contient la dernière valeur retournée par une fonction

On compare \$? avec 1. Si ils sont égaux, on est dans le cas où l'argument rentré par l'utilisateur n'est pas réel et on affiche un message d'erreur. Sinon, il est réel.

Note: il faut fermer les `if` avec des `fi`

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)

Script:

```
#!/bin/bash
if [ $# -eq 0 ] ; then
  >> echo $0 "nom_utilisateur"
else
  >> if [ -z "$(cut -d: -f1 /etc/passwd | grep -x $1 )" ] ; then
  >>     echo "L'utilisateur n'existe pas"
  >> else
  >>     echo "L'utilisateur existe"
  >> fi
fi
```

On compare le nombre d'arguments entrés par l'utilisateur (\$#) avec 0 : si l'utilisateur n'a pas entré de paramètre (\$#=0), on lui indique sous quel forme il doit rentrer sa commande (*Utilisation :nom_du_script nom_utilisateur*). On utilise \$0 pour avoir accès au nom du script.

`cut -d: -f1 /etc/passwd | grep -x lib`

`cut` : commande qui récupère des zones spécifiques d'un fichier

-d : indique le symbole qui délimite les colonnes (ici 😊)

-f1 : indique la colonne que l'on sélectionne dans le fichier

/etc/passwd : fichier où sont stockés entre autre les identifiants des différents utilisateurs de la machine. Grâce à cette première partie de la commande, on obtient la liste des identifiants de utilisateurs de la machine.

| : passe la sélection à la commande suivant le pipe

grep : trouve les occurences du terme passé en paramètre (ici lib).

-x : fais en sorte que les occurences correspondent exactement au terme passé en paramètre.

On trouve le nom d'utilisateur dans la liste des utilisateurs. Si il n'y en pas de ce nom, on obtient une chaîne vide, sinon, on obtient le nom de l'utilisateur.

-z : vérifie que la chaîne est vide

Si la chaîne est vide : il n'y a pas d'utilisateur de ce nom. Sinon, cet utilisateur existe.

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).

Script:

```
#!/bin/bash

val=1

for i in $(seq 1 $1)
do
    val=$(( $val * $i ))
done

echo $val
```

L'indice `i` prend des valeurs de 1 à la valeur rentrée par l'utilisateur (`$1`). On actualise la valeur de `val` en multipliant chaque `i` avec la valeur de `val` précédente. On retourne la valeur de `val` qui est le résultat du factoriel.

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`).

Script:

```
#!/bin/bash
Nb_rand=$(( $RANDOM % 1000 + 1))

read -p 'Quel est le juste prix ?' Nb_util
while [ $Nb_util -ne $Nb_rand ]
do
    >> if [ $Nb_util -lt $Nb_rand ]; then
    >>     read -p 'C est plus ! Essaye encore !' Nb_util

    >> else
    >>     read -p 'C est moins ! Essaye encore !' Nb_util
    >> fi
done

echo 'Bravo ! Gagné !'
```

On génère un nombre random grâce à `$RANDOM`, et on en indique les valeurs min et max avec le `%`. On demande à l'utilisateur un nombre et, tant que ce nombre est différent du nombre random, on lui demande un nouveau nombre en lui donnant des indications.

`$num_utilisateur -lt $num_random` : teste si `num_utilisateur < num_random`. Si oui : Le nombre à trouver est plus grand.

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)

Script:

```
#!/bin/bash

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

erreur=0
for param in $*; do
    is_number $param
    if [ $? -eq 1 ]; then
        erreur=$((erreur+1))
    fi
done

if [ $erreur -eq 0 ]; then
    moy=0
    max=$1
    min=$1
    for param in $*; do
        if [ $max -lt $param ]; then
            max=$param
        fi
        if [ $min -gt $param ]; then
            min=$param
        fi
        moy=$((moy+$param))
    done
    moy=$((moy/$#))
    echo 'moyenne : ' $moy
    echo 'minimum : ' $min
    echo 'maximum : ' $max
else
    echo "Au moins un des parametres n'etait pas un nombre"
fi
```

Lors de l'appel de la fonction, l'utilisateur rentre en paramètres tous les nombres qu'il souhaite voir comparés et moyennés. A l'aide d'une boucle `for` sur `$*` (la variable qui contient tous les paramètres rentrés par l'utilisateur) et de l'appel de la fonction `is_number`, le programme vérifie que l'utilisateur a bien rentré des nombres réels. Lorsqu'un des paramètres n'est pas un nombre réel, on incrémente une variable `erreur` (initialisée à 0). A la fin de la boucle, si la variable `erreur` n'est pas nulle cela signifie qu'au moins un des paramètres n'est pas un nombre réel, par conséquent le programme envoie un message d'erreur et s'arrête. Si tous les paramètres sont des nombres réels, on réalise une seconde boucle `for` sur les paramètres. Par comparaison avec chaque paramètre on trouve le minimum et le maximum qu'on stocke dans des variables. Pour calculer la moyenne, durant la boucle on additionne tous les paramètres dans une variable. A la sortie de la boucle, celle-ci est divisée par le nombre de paramètres (`$#`) pour obtenir la moyenne.

Note:

Une boucle `for` doit commencer par `do` et se terminer par `done`

Pour faire un calcul : `$(($variable_1 signe_calcul $variable_2))`

`$num1 -eq $num2` : Teste si les deux nombres sont égaux

`$num1 -ne $num2` : Teste si les deux nombres sont différents

`$num1 -lt $num2` : Teste si `num1 < num2`

`$num1 -le $num2` : Teste si `num1 ≤ num2`

`$num1 -gt $num2` : Teste si `num1 > num2`

`$num1 -ge $num2` : Teste si `num1 ≥ num2`

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.

Script:

```
#!/bin/bash

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

read -p 'Combien de valeurs voulez vous ?' taille
tableau=()
compt=0
while [ $compt -ne $taille ]
do
    read -p 'Donnez moi un nombre: ' tableau[$(($compt))]
    compt=$((compt+1))
done

erreur=0
for param in ${tableau[*]}; do
    is_number $param
    if [ $? -eq 1 ]; then
        erreur=$((erreur+1))
    fi
done

if [ $erreur -eq 0 ]; then
    moy=0
    max=${tableau[1]}
    min=${tableau[1]}
    for param in ${tableau[*]}; do
        if [ $max -lt $param ]; then
            max=$param
        fi
        if [ $min -gt $param ]; then
            min=$param
        fi
        moy=$((moy+$param))
    done
    moy=$((moy/$taille))
    echo 'moyenne : ' $moy
    echo 'minimum : ' $min
    echo 'maximum : ' $max
else
    echo "Au moins un des parametres n'etait pas un nombre"
fi
```

Afin d'adapter le programme précédent pour que l'utilisateur puisse rentrer les valeurs dans un tableau, nous commençons par demander à l'utilisateur le nombre de valeurs qu'il a à traiter et nous stockons ce nombre dans une variable. Ensuite, à l'aide d'une boucle while tant que cette taille n'est pas atteinte, nous remplissons le tableau. Afin de remplir la *i*ème case du tableau nous utilisons : `tableau[$(($i))]`.

Pour parcourir les valeurs contenues dans le tableau avec une boucle for, nous utilisons : `for param in ${tableau[*]}`. `param` prend toutes les valeurs contenues dans le tableau : en effet, `tableau[*]` contient toutes les valeurs du tableau .

Le reste du programme reste inchangé.

Notes: Copier VM->session `scp -P 2222 marialice@localhost:script/nom_fichier.sh ~/Documents`