

Compte Rendu TP2

A

SABRAN Raphael- DUMAS Maxime

Exercice 1. Variables d'environnement

Question 1

Bash trouve les commandes tapées par l'utilisateur dans PATH. On utilise la commande `printenv PATH`. Cette variable d'environnement contient les chemins des répertoires des commandes. Les différents chemins sont :
`/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin.`

Question 2

La variable HOME permet à la commande `cd` de nous ramener dans notre répertoire personnel. Elle contient le chemin de notre dossier personnel.

Question 3

LANG permet de déterminer la langue que le Shell va utiliser pour communiquer avec l'utilisateur. PWD contient le chemin absolu vers le répertoire courant. OLDPWD contient le chemin absolu vers le répertoire courant précédent. SHELL détermine l'interpréteur utilisé par défaut, car on peut utiliser jusqu'à 6 shells simultanément.

Question 4

On utilise la commande `MY_VAR="coucou"` pour créer la variable MY_VAR contenant la valeur "coucou". On affiche son contenu avec `echo $MY_VAR`.

Question 5

La commande `bash` permet de créer un nouveau shell (une nouvelle session). En conséquence toutes les variables locales créées précédemment ne seront pas connues de cette nouvelle session.

Question 6

En utilisant la commande `export MY_VAR`, MY_VAR étant maintenant une variable d'environnement, elle est globale et donc connue de tous les différentes sessions.

Question 7

On utilise la commande : `export NOMS="SABRAN DUMAS"` afin de créer une variable d'environnement NOMS qui prend comme valeur SABRAN DUMAS

Question 8

On utilise la commande `echo Bonjour à vous deux, $NOMS`, qui permet d'afficher la chaîne de caractère en affichant la valeur de `NOMS`.

Question 9

`Unset` efface de la mémoire la variable passée en paramètre, alors que si l'on donne une valeur vide à une variable, elle va toujours exister dans la mémoire.

Question 10

On utilise la commande `echo '$HOME' = "$HOME"` pour afficher la phrase demandée.

Programmation Bash

La compilation des programmes se fait avec la commande `chmod u+x nom_du_programme.sh`
L'exécution des programmes se fait avec la commande `nom_du_programme.sh`

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.

```
GNU nano 4.3 testpwd.sh
#!/bin/bash

PASSWORD="1234"

read -s -p 'Saisissez votre mot de passe :' mdp

if [ "$mdp" = "$PASSWORD" ] ; then
    echo " Valide"
else
    echo " Non valide"
fi
```

On stocke le mot de passe dans la variable `PASSWORD`. Grâce à la commande `read`, on affiche dans le shell une phrase (`read -p`) et on attend que l'utilisateur écrive son mot de passe qui n'est pas affiché (`-s`). Ce mot de passe est stocké dans la variable `mdp`. On compare les valeurs de `PASSWORD` et de `mdp` grâce au `$` qui prend en compte les valeurs des variables.

Exercice 3

Écrivez un script qui prend un paramètre et utilise la fonction suivante pour vérifier que ce paramètre est un nombre réel :

```
GNU nano 4.3 testreal.sh
#!/bin/bash

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

is_number $1
var=$?

echo "$var"
if [ $var == 0 ] ; then
    echo 'Un réel'
else
    echo 'Pas un réel'
fi
```

En utilisant la fonction `is_number()`, on crée une variable `var` où on lui met comme valeur le code de retour de la dernière commande, soit la dernière valeur retournée par la fonction `is_number($?)`.

Ensuite, si le nombre saisi par l'utilisateur est un réel, alors la fonction renvoie 0, sinon il renvoie 1. On renvoie donc la phrase "Un réel" si `var` est égal à 0, sinon, on renvoie "Pas un réel".

Exercice 4

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)

```
GNU nano 4.3 testuser.sh
#!/bin/bash

var=0
juste=0
if [ $# == 0 ] ; then
    echo "Utilisation : $0"
else
    for param in $*; do
        id -u $param > /dev/null 2>&1
        if [ "$?" == 0 ] ; then
            echo "$param valide"
        else
            echo "$param non valide"
        fi
    done
fi
```

La première condition vérifie si l'utilisateur a entré le bon nombre d'argument. Sinon, on lui affiche "Utilisation : nom_du_script nom_utilisateur". Si le bon nombre de paramètres sont rentrés (juste le nom de l'utilisateur), on prend l'ensemble des paramètres avec `$*`. Ensuite, on prend ce nom d'utilisateur (`$param`) et on redirige l'erreur au même endroit que la sortie dans `/dev/null` (donc l'affichage du mot de passe ne se fera pas car on est dans `/dev/null`) grâce à la commande `id -u $param > /dev/null 2>&1`. Ainsi, on peut voir si il y a une erreur (si il est présent ou non). Si il n'y a pas d'erreur, alors l'utilisateur est valide, sinon il est non valide.

Exercice 5

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

```
GNU nano 4.3                                exo5.sh
#!/bin/bash

export var="1"
for (( i="1" ; i<$1+1 ; i++)) ; do
    ((var=var*i))
done
echo $var
```

On crée une variable `var` que l'on initialise à 1. Dans la boucle `for`, vu que l'on fait des opérations mathématiques il doit forcément y avoir les doubles parenthèses. L'indice `i` prend les valeurs de 1 à la valeur saisie par l'utilisateur puis on actualise la valeur de `var`.

Exercice 6

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`).

```
GNU nano 4.3                                exo6.sh
#!/bin/bash

Nombre=$(( $RANDOM % 1000 + 1 ))

read -p "Devinez le prix ! " Nombre_utilisateur
while [ $Nombre_utilisateur -ne $Nombre ]
do
    if [ $Nombre_utilisateur -lt $Nombre ] ; then
        read -p "C'est plus ! " Nombre_utilisateur
    else
        read -p "C'est moins ! " Nombre_utilisateur
    fi
done
echo "Gagné!"
```

On prend un nombre aléatoire entre 1 et 1000 avec la ligne `$RANDOM % 1000 + 1`. On demande à l'utilisateur de proposer un nombre que l'on stockera dans la variable `Nombre_utilisateur`. Tant que les 2 nombres ne sont pas égaux (condition de la boucle `while $val1 -ne $val2`), si le nombre saisi par l'utilisateur est plus petit que le nombre à trouver (`$val1 -lt $val2`), alors on affiche une

phrase et le prochain nombre saisi par l'utilisateur actualisera la valeur de la variable

Nombre_utilisateur.

Exercice 7

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.

```
GNU nano 4.3                                exo7.sh
#!/bin/bash
function is_number()
{
    re='^[+-]?[+-9]+([.][+-9]+)?$'
    if ! [[ $1 =~ $re ]] ; then
        return 1
    else
        return 0
    fi
}
erreur=0
var=?
nombre_param=0
for param in $* ; do
    ((nombre_param=nombre_param+1))
    is_number $param
    if [ $var -eq 1 ] ; then
        echo 'ERREUR'
        ((erreur=erreur + 1))
    fi

    if [ $param -gt 100 -o $param -lt -100 ] ; then
        echo 'Il faut un nombre compris entre -100 et 100'
        ((erreur=erreur + 1))
    fi
done
if [ $nombre_param -ne 3 ] ; then
    echo 'Il faut 3 paramètres'
    ((erreur=erreur+1))
fi
```

```
if [ $erreur -eq 0 ] ; then
    min=$1
    max=$1
    somme=0
    moy=0
    for param in $*;do
        if [ $min -gt $param ] ; then
            min=$param
        fi
        if [ $max -lt $param ] ; then
            max=$param
        fi
        somme=$(( $somme+$param ))
    done
    moy=$(( $somme/$# ))
    echo 'minimum =' $min
    echo 'maximum =' $max
    echo 'moyenne =' $moy
fi
```

En utilisant la fonction `is_number`, on fait différents tests sur les paramètres mis en entrée de notre script. La commande `$` prend en compte l'ensemble des paramètres d'entrée et on test chacun d'entre eux, que l'on met dans la variable `param`. On test si le nombre est un réel et si il est compris entre -100 et 100. Pour chaque paramètre, on incrémente un compteur global comptant le nombre de paramètres au total. Si ce dernier n'est pas égal à 3 (`$nombre_param -ne 3`) alors il y a une erreur. Ensuite, si il n'y a pas d'erreur, on peut passer dans la boucle permettant de calculer le minimum, maximum et la moyenne. J'initialise par défaut la valeur de `min` et de `max` au premier paramètre puis je fais des tests sur chacun pour trouver le min et le max. Je met à jour une variable `somme` qui me permettra de calculer la moyenne à la fin. Une fois le min et le max trouvé, je calcule la moyenne (grâce à la commande `$#` qui renvoi le nombre de paramètres) puis j'affiche le min, le max et la moyenne des 3 nombres.

2. Généralisez le programme à un nombre quelconque de paramètres (pensez à SHIFT)

```

GNU nano 4.3                                exo7.sh
#!/bin/bash
function is_number()
{
    re='^[+-]?[+-9]+([.][+-9]+)?$'
    if ! [[ $1 =~ $re ]] ; then
        return 1
    else
        return 0
    fi
}
erreur=0
var=$?
nombre_param=0
for param in $* ; do
    ((nombre_param=nombre_param+1))
    is_number $param
    if [ $var -eq 1 ] ; then
        echo 'ERREUR'
        ((erreur=erreur + 1))
    fi
    if [ $param -gt 100 -o $param -lt -100 ] ; then
        echo 'Il faut un nombre compris entre -100 et 100'
        ((erreur=erreur + 1))
    fi
done

if [ $erreur -eq 0 ] ; then
    min=$1
    max=$1
    somme=0
    moy=0
    for param in $*;do
        if [ $min -gt $param ] ; then
            min=$param
        fi
        if [ $max -lt $param ] ; then
            max=$param
        fi
        somme=$(( $somme+$param ))
    done
    moy=$(( $somme/$# ))
    echo 'minimum = ' $min
    echo 'maximum = ' $max
    echo 'moyenne = ' $moy
fi

```

Pour généraliser ce programme, il suffit d'enlever la condition sur le nombre de paramètres pour que cela fonctionne.

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.


```

GNU nano 4.3                                exo7.sh
#!/bin/bash
function is_number()
{
    re='^[+-]?[+-9]+([.][+-9]+)?$'
    if ! [[ $1 =~ $re ]] ; then
        return 1
    else
        return 0
    fi
}
erreur=0
var=?
nombre_notes=0
tableau=()

read -p 'Combien de notes vont-êtr   ren  r  es ? ' taille_tableau
while [ $nombre_notes -ne $taille_tableau ]
do
    read -p 'Saisissez une note : ' tableau[$(($nombre_notes))]
    ((nombre_notes=nombre_notes+1))
done

for param in ${tableau[*]} ; do
    is_number $param
    if [ $var -eq 1 ] ; then
        echo 'ERREUR'
        ((erreur=erreur + 1))
    fi
    if [ $param -gt 20 -o $param -lt 0 ] ; then
        echo 'Il faut un nombre compris entre 0 et 20'
        ((erreur=erreur + 1))
    fi
done

if [ $erreur -eq 0 ] ; then
    min=${tableau[1]}
    max=${tableau[1]}
    somme=0
    moy=0
    for param in ${tableau[*]};do
        if [ $min -gt $param ] ; then
            min=$param
        fi
        if [ $max -lt $param ] ; then
            max=$param
        fi
        somme=$(( $somme+$param ))
    done
    moy=$(( $somme/$taille_tableau ))
    echo 'minimum =' $min
    echo 'maximum =' $max
    echo 'moyenne =' $moy
fi

```

Pour que les notes soient stock  es dans un tableau, on d  finit au pr  alable une taille du tableau correspondant au nombre de notes que l'utilisateur va rentrer. Tant que le nombre de notes saisies est diff  rent de la taille du tableau, l'utilisateur rentre une note puis elle est stock  e dans le tableau. On fait ensuite la m  me chose que pr  c  demment mais la variable `param` va parcourir tous les   l  ments du tableau avec `${tableau[/]}`. On donne au `min` et au `max` la premi  re valeur du tableau

(`${tableau[1]}`). Enfin, pour la moyenne, on divise la somme des notes par le nombre de notes saisies par l'utilisateur.