Administration système - TP 2

Auteurs :

BIARD Gauthier

MARION Audran

Exercice 1 : Variables d'environnement

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

Les commandes saisies par l'utilisateur se trouvent dans les fichiers qui sont indiqués par la variable d'environnement **PATH**.

On peut afficher ces répertoires avec la commande :`'printenv PATH'`.

> Cette commande nous retourne donc :
> `/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 ?

Il s'agit de la variable d'environment `'$HOME'` qui permet cela.

> Quand nous exécutons `'cd'`, cela équivaut à exécuter `'cd $HOME'`

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

- **LANG** : détermine la langue que les logiciels utilisent pour communiquer avec l'utilisateur.

> si on tape `'printenv LANG'`, le bash retourne `'fr_FR.UTF-8'`

- **PWD** : contient le répertoire de travail courant de l'interpréteur de commande.
- **OLDPWD** : contient le répertoire précédent.

> Celui vers lequel nous retournons en exécutant `'cd -'`.

- **SHELL** : interpréteur de commande préféré de l'utilisateur tel qu'il est défini dans le fichier « /etc/passwd » .
- _ : La dernière commande saisie par l'utilisateur connecté

Dernier argument utilisé et non dernière commande. Peut être une commande si la commande n'a pas utiliser d'argument

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

```
MY_VAR=test
echo $MY_VAR
test
```

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.

Lorsque nous faisons cela, nous ouvrons une nouvelle session shell donc les variables sont réinitialisées. La variable *MY_VAR* n'existe donc pas dans cette session shell.

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

```
export MY_VAR=test
```

La variable MY_VAR est désormais une variable d'environnement. Elle sera donc valide sur tous les shells tant que le shell créateur n'est pas fermé. Tout les shell issus du shell qui a produit l'export

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='BIARD MARION'
printenv NOMS
BIARD MARION
```

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.

```
echo "Bonjour à vous deux, $NOMS !"
```

En retour, nous avons : `Bonjour à vous deux, BIARD MARION !`

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

Si la valeur est vide, la variable reste définie et peut donc être utilisée et modifiée mais elle est vide.

Avec la commande unset, la variable sera totalement supprimée de la mémoire du shell. En cas de lecture, cela générera une erreur.

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

Il suffit de taper dans la ligne de commande :

```
echo "\$HOME=$HOME"
```

Programmation Bash

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.

```
testpwd.sh:

#!/bin/bash

PASSWORD = "azerty"

read -s -p 'Saisir votre mot de passe : ' password

echo -e '\n'

if [ $password = "mdp" ]; then
        echo -e "Mot de passe valide"
else
        echo -e "Mot de passe invalide"
fi
```

attention si password est vide il y a une erreur tester
x$password = x"mdp"
si password est vide le test est x = xmdp alors que sans c'est rien = "mdp" et du coup erreur de syntaxe

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 :

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

Il affichera un message d'erreur dans le cas contraire.

Le code est donc le suivant :

```
#!/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 0 ]; then
    echo 'C est un reel'
else
    echo 'Ce n est pas un reel'
fi
```

peut s'écrire if is_number $1; then

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)

CheckUser.sh:

```
#!/bin/bash

if [ $# = 0 ]
then
        echo "Pour utiliser : $0 nom_utilisateur"
else
        if cut -d: -f1 /etc/passwd | grep -q $1
        then
                echo "L'utilisateur $1 est dans la base de données."
        else
                echo "L'utilisateur $1 n'est pas dans la base de données."
        fi
fi
```

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).

factoriel.sh

```
#!bin/bash

fact=1

for i in $(seq 2 $1);do
  fact = $(($fact * $i))
done
echo $fact
```

> Si nous éxecutons le programme sans rentrer de paramètre, le bash nous retourne la valeur 2 par défaut.

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).

```
#!/bin/bash
```

```
RDM=$((1 + RANDOM % 1000));
REP=0;

read -p 'Saisissez un nombre entre 1 et 1000 : '

while [ $REP -ne $RDM ];
do
        if [ $REP -gt $RDM ]; then
                read -p "C'est moins ! : " REP
        elif [ $REP -lt $RDM ]
        then
                read -p "C'est plus ! : " REP
        fi
done

echo "Vous avez gagné !";
```

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.

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

MAX=-100;
MIN=100;
MOY=0;

for i in $(seq 1 3);        for i in $* avec $i pour avoir la valeur associé
do
        if is_number ${!i} ; then
                if [ ${!i} -lt $MIN ]; then
                        MIN=${!i}
                fi
                if [ ${!i} -gt $MAX ]; then
                        MAX=${!i}
                fi

                MOY=$(($MOY+${!i}))
        else
```

```
                echo "${!i} n'est pas un nombre entiers"
        fi
done
MOY=$(($MOY/3))

echo "MIN: $MIN / MOY: $MOY / MAX: $MAX";
```

Si il y a erreur, ça ne sert à rien de continuer.

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

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

MAX=-100;
MIN=100;
MOY=0;
NBVALUE=0;

while (("$#"));
do
        if is_number $1 ; then
                if [ $1 -lt $MIN ]; then
                        MIN=$1
                fi
                if [ $1 -gt $MAX ]; then
                        MAX=$1
                fi
                NBVALUE=$(($NBVALUE+1));
                MOY=$((MOY+$1));
        else
                echo "$1 n'est pas un nombre entiers"
        fi
        shift
done
MOY=$(($MOY/$NBVALUE))
echo "MIN: $MIN / MOY: $MOY / MAX: $MAX";
```

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.

```bash
#!/bin/bash

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

GET=0;
MAX=-100;
MIN=100;
MOY=0;

TAB=();

while [ $GET -lt 101 ] && [ $GET -gt -101 ]
do
        read -p 'Saisissez la note : ' GET

        if [ $GET -lt 101 ] && [ $GET -gt -101 ]; then
                if is_number $GET ; then

                        TSIZE=${#TAB[*]}
                        TAB[$TSIZE+1]=$GET

                        if [ $GET -lt $MIN ]; then
                                MIN=$GET
                        fi
                        if [ $GET -gt $MAX ]; then
                                MAX=$GET
                        fi

                        MOY=$(($MOY+$GET))
                else
                        echo "$GET n'est pas un nombre entiers"
                fi
        fi
done
TSIZE=${#TAB[*]}
echo "Generate Table ... ($TSIZE value(s))";
echo ${TAB[*]};
MOY=$(($MOY/$TSIZE))
echo "MIN: $MIN / MOY: $MOY / MAX: $MAX";
```

Mélange saisie des nombres/ stockage dans un tableau avec le traitement.