TP2 BASH par Kamil BENNIS et Fabien VERDIER

Exercice 1. Variables d'environnement

la variable PATH contient l'ensemble des dossiers utiliser pour la recherche des commandes taper

Bash trouve les commandes tapées par l'utilisateurs dans le dossier `/bin`, la variable d'environnement home permet à la commande cd tapée sans argument de nous ramener dans notre répertoire personnel.

Parmi les variables d'environnements on trouve :

- `LANG` Le paramètre linguistique de base utilisé par les applications du système.
- `PWD` Le répertoire de travail courant de l'interpréteur de commande.
- `OLPWD` determine le dossier avant la dernière commande cd utilisée.
- `SHELL` L'interpréteur de commande préféré de l'utilisateur tel qu'il est défini dans le fichier `/etc/passwd`.

On a crée une variable locale MY_VAR `MY_VAR=...` et pour vérifier son existense on peut l'afficher `echo $MY_VAR`. Dans ce cas là, la variable agit comme une variable locale, et donc la commande `bash` ne permet pas d'afficher cette variable. Ce qui n'est pas le cas quand MY_VAR est défini en tant que variable d'environnement.

Pourécrire une commande qui affiche ”Bonjour à vous deux, binôme1 binôme2 !” en utilisant la variable NOMS qu'on aura créer au préable on peut taper faire un alias en taper la commande `bonjour ="Bonjour à vous deux," ; echo $NOMS` echo "Bonjour à vous deux, $NOMS"

Donner une valeur vide à une variable indique au système l'existence d'une valeur alors que la commande `unset` provoque la supression totale de la variable. Uniquement si la variable est exporter

Pour afficher le chemin de notre dossier personnel on peut utiliser la commande `echo '$HOME=' ; echo $HOME` echo "\$HOME=$HOME"

PROGRAMMATION BASH

Pour permettre l'éxecution d'un programme bash : `chmod u+x mon_script.sh` et pour l'éxécuter : `./mon_script.sh`

Exercice 2. Contrôle de mot de passe

Le script suivant demande un mot de passe à l'utilisateur et vérifie s'il correspond au mot de passe codé en dur dans le programme (ici "milou"). La commande `read -s -p` lit ce que rentre l'utilisateur sans montrer les caractères (`-s`) et en affichant un message (`-p`).

```
#!/bin/bash
PASSWORD="milou"
read -s -p 'Veuillez saisir votre mot de passe : ' PASSWORD_test
echo ''
if [ $PASSWORD = $PASSWORD_test ]; then
        echo 'Mot de passe correct !'
else
```

Attention si PASSWORD_test est vide alors le script produit une erreur plutôt utiliser la comparaison
x$PASSWORD = x$PASSWORD_test
si PASSWORD_test est vide il y aura au moins une comparaison avec x

```
        echo 'Raté'
fi
```

Exercice 3. Expressions rationnelles

Le script suivant vérifie si l'argument que l'on passe au programme est bien un nombre réel. On utilise une fonction. Le "return" de la dernière fonction utilisée est stocké dans la variable `$?`. Pour tester l'égalité entre deux valeurs numériques, on utilise `-eq`.

```
#!/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 'Ceci est un nombre réel !'

else
        echo 'Erreur !'
fi
```

Peut s'écrire plus simplement
if is_number $1; then

Exercice 4. Contrôle d'utilisateur

Le code suivant vérifie si le premier argument que l'on passe au programme est bien dans la liste des utilisateurs. Si on ne passe pas d'argument, le programme indique la syntaxe à utiliser. Nous avons stocké tous les noms d'utilisateur dans un tableau avec la commande `cat /etc/passwd | awk -F: '{print $ 1}'`, puis nous avons vérifié si le nom passé par l'utilisateur était bien dans le tableau.

```
#!/bin/bash
if [ $# -eq 0 ]; then
        echo "Utilisation : $0 nom_utilisateur"
else

tab=$(cat /etc/passwd | awk -F: '{print $ 1}')
present=0

for user in $tab
        do
        if [ $user = $1 ]; then
                echo "L'utilisateur est bien présent !"
                present=1
        fi
```

un break pourrait être bien ici

```
        done
fi

if [ $present -eq 0 ]; then
        echo "L'utilisateur n'est pas présent !"
fi
```

Exercice 5. Factorielle

Dans ce script, on calcule la factorielle d'un nombre passé en argument avec une simple itération.

```
#!/bin/bash
facto=1
for i in $(seq 1 $1); do
        facto=$(($facto * $i));
done
echo $facto
```

Exercice 6. Le juste prix

Ce code est le classique jeu du juste prix. Le programme choisit un nombre entre 0 et 1000 et l'utilisateur doit deviner le nombre.

```
#!/bin/bash
nombre_choisi=$(( RANDOM % 1000 ) +1 ))
nombre_user=-1
while [ $nombre_choisi -ne $nombre_user ]
do
read -p "Veuillez entrer un nombre entre 1 et 1000 : " nombre_user
if [ $nombre_choisi -eq $nombre_user ]; then
        echo "Bravo, vous avez gagné !"
elif [ $nombre_user -lt $nombre_choisi ]; then
        echo "C'est plus !"
elif [ $nombre_user -gt $nombre_choisi ]; then
        echo "C'est moins !"
fi
```

Pourrait n'être mis qu'après le done sans test. Si on sort de la boucle c'est qu'on a juste

Exercice 7. Statistiques

Ce script calcule le minimum, le maximum et la moyenne des arguments passés au programme.

```
#!/bin/bash
max=$1
min=$1
moy=$1
N=$#
while [ $# -gt 1 ]; do
```

```
        shift
        echo "argument : $1"
        echo "nb args : $#"
        if [ $max -lt $1 ]; then
                max=$1
        fi
        if [ $min -gt $1 ]; then
                min=$1
        fi
        moy=$(( $moy + $1 ))
done

moy=$(( $moy/$N ))

echo "Le minimum est : "
echo $min
echo "Le maximum est : "
echo $max
echo "La moyenne est : "
echo $moy
```