Nicolas GAZERIAN, Nicolas VERRIERE 4ETI

TP2 - Administration système : Compte Rendu

Exercice 1 :

Pas chez moi, ça dépend de la variable PATH

1. Les commandes tapées par l'utilisateur se trouvent dans les dossiers suivants : /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
2. Si on ne fournit pas d'argument à la commande cd, c'est la valeur de la variable shell HOME qui est utilisée.
3. LANG détermine temporairement sur la session courante la langue que les logiciels utilisent pour communiwuer avec l'utilisateur. PWD affiche le chein jusqu'au répertoire de travail. OLDPWD affiche le répertoire courant précédent la dernière commande cd. SHELL indique quel type de shell est en cours d'utilisation. _ correspond au dernier argument de la commande précédente.

manque des lettres

pas le shell en cours mais celui par défaut

4. Après création de la variable MY_VAR, il est possible de l'afficher, elle ou son contenu, grâce à la fonction echo.
5. La commande bash crée un nouvel interpréteur de commandes. La variable locale n'existe pas dans le nouveau bash car il a été créé par celui où a été créée (mais pas exportée).

Pas français

6. Cette fois, la variable a été créée en utilisant la commande export, ce qui en fait une variable d'environnement. Par conséquent, les bash fils de celu ide départ connaissent aussi la variable. Il est donc possible d'afficher la variable ou sa valeur dans le nouveau bash.
7. *code :* export NOMS="GAZERIAN VERRIERE" printenv NOMS
8. *code :* echo "Bonjour à vous deux, $NOMS"
9. Unset supprime la variable plutôt que simplement la "vider". Le renvoi de la variable aura le même résultat à l'affichage mais unset est plus "propre".
10. *code :* echo '$HOME =' $(pwd)

echo "\$HOME=$HOME" pwd == répertoire courant qui n'a rien a voir avec home!

Programmation Bash

```
**Exercice 2 :**
*code :*
#!/bin/bash

MOTDEPASSE=password

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

if [ $mdp =  $MOTDEPASSE ]; then
        echo "mot de passe correct"
else
        echo "mot de passe incorrect"
fi
```

Attention si mdp est vide => erreur de syntaxe car ce qui sera lu sera
if [= password]
un bon moyen d'éviter ça c'est d'écrire
x$mdp = x$MOTDEPASSE ce qui donnera avec mdp vide:
if [x = xpassword]

Quelques tests effectués :
```
chmod u+x script2.sh
./script2.sh passswordd
mot de passe incorrect
./script2.sh password
mot de passe correct
```

Dans la réalité, le script rendait invisible les mots de passe au moment où il étaient entrés par l'utilisateur mais dans les tests ci-dessus, nous les avons laissés écrits.

Exercice 3 :
code :
```
#!/bin/bash

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

ma_fonction $1

if [ $? == 0 ]; then
        echo "Le paramètre est bien un nombe réel."
else
        echo "ERREUR : Le nombre rentré en paramètre n'est pas réel."
fi
```

peut s'écrire: if ma_fonction $1; then

Quelques tests effectués :
```
chmod u+x script3.sh
./script3.sh 8
  Réponse : Le paramètre est bien un nombe réel.
./script3.sh 8i
  Réponse : ERREUR : Le nombre rentré en paramètre n'est pas réel.
```

Exercice 4 :
code :
```
#!/bin/bash

if [ $# = 0 ]; then
        echo "Utilisation : $0  nom_utilisateur"
else
         getent passwd | grep $1
fi
```

Quelques tests effectués :
```
chmod u+x script4.sh
```

C'est pas contient qui nous intéresse mais s'il existe un utilisateur donné. Dans votre exemple nicolas. S'il y a nicolas et nicolas2. ça doit dire oui mais s'il n'y a que nicolas2 et qu'on demande nicolas ça doit dire non

```
./script4.sh
  Réponse : "Utilisation : ./script4.sh nom_utilisateur"
./script4.sh iuvzecze
  Réponse : ""
./script4.sh nicolas
 Réponse : Liste de tous les utilisateurs dont le nom contient "Nicolas"
```

Exercice 5 :
code :

```
#!/bin/bash

function factorielle {
        n=$1
        if [ $n -eq 0 ]; then
                echo 1
        else
                echo $(( $n * `factorielle $(( $n - 1 ))` ))
        fi
}

echo "Le résultat vaut : $(factorielle $1)"
```

L'utilisation de récursivité est extrêmement bourrin là où une simple boucle fait l'affaire.
à chaque appel récursif vous lancer 3 nouveau bash.

Quelques tests effectués :

```
chmod u+x script5.sh
./script5.sh 3
  Réponse : 6
./script5.sh 5
  Réponse : 120
./script5.sh 6
  Réponse : 720
```

Exercice 6 :
Code :

```
#!/bin/bash

number=$(( $RANDOM % 1000 + 1 ))
read -p 'Devine le nombre' proposition
#echo "$number"


comparaison () {
        if [ $proposition -lt $number ]; then
                read -p 'C est plus!' proposition
                comparaison proposition
        elif [ $proposition -gt $number ]; then
                read -p 'C est moins!' proposition
                comparaison proposition
        else
                echo "Gagné!"
        fi
}
```

vous n'utilisez visiblement pas l'argument, il n'y a aucun usage du $1 fait. La récursivité ici est donc inutile car inutilisé correctement.

```
comparaison proposition

*Quelques tests effectués :*
chmod u+x script6.sh
./script6.sh
Réponse : "Devine le nombre50
C est plus!500
C est plus!750
C est moins!625
C est moins!575
C est plus!600
C est plus!613
C est moins!617
C est moins!608
C est moins!604
C est plus!606
Gagné!"
./script6.sh
Réponse : "Devine le nombre500
C est moins!250
C est moins!125
C est plus!124
C est plus!180
C est moins!160
C est plus!170
C est moins!165
C est moins!163
C est moins!162
C est moins!121
C est plus!161
Gagné!"

Chaque nombre en fin de ligne est la tentative de l'utilisateur après la
consigne de début de chaque ligne.


**Exercice 7 :**
Question 1)
*Code :*
#!/bin/bash

for param in "$@"; do
        if [ $param -lt -100 ] || [ $param -gt 100 ] ; then
                echo "Utilisation : Veuillez entrer 3 nombres entre -10$
                exit
        fi
done

MOY=$((  $(( $1 + $2 + $3 )) / 3 ))
echo $MOY

MIN=$1
for param in "$@"; do
```

```
        if [ $param -lt $MIN ] ; then
                MIN=$param
        fi
done
echo $MIN

MAX=$1
for param in "$@"; do
        if [ $param -gt $MAX ] ; then
                MAX=$param
        fi
done
echo $MAX


*Quelques tests effectués :*
chmod u+x script5.sh
./script7.sh 70 80 90
80
70
90
./script7.sh 1 2 -300
Utilisation : Veuillez entrer 3 nombres entre -100 et 100



Question 2)
Nous n'avons pas eu besoin d'utiliser la commande shift mais le résultat
est le suivant :
*Code :*
#!/bin/bash

for param in "$@"; do
        if [ $param -lt -100 ] || [ $param -gt 100 ] ; then
                echo "Utilisation : Veuillez entrer 3 nombres entre -10$
                exit
        fi
done

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

MOY=$((  $(( $somme )) / $# ))
echo $MOY

MIN=$1
for param in "$@"; do
        if [ $param -lt $MIN ] ; then
                MIN=$param
        fi
done
```

Une seule boucle aurait suffit

```
echo $MIN

MAX=$1
for param in "$@"; do
        if [ $param -gt $MAX ] ; then
                MAX=$param
        fi
done
echo $MAX
```

Quelques tests effectués :

```
./script7.sh 1 2 3 4 5 6 7
  Réponse : "28
4
1
7"
./script7.sh 70 80 90 70 80
  Réponse : "390
78
70
90"
./script7.sh 70 80 90 150
  Réponse : "Utilisation : Veuillez entrer 3 nombres entre -100 et 100"
./script7.sh -100 100 0 50 -50 60
  Réponse : "60
10
-100
100"
```