GALLAND Cyprien VATIN Clément

Compte rendu TP2

Bash

Exercice 1. Variables d'environnement

1) bash trouve les commandes tapées par l'utilisateur dans le fichier .bash_history. On peut afficher l'historique de toutes les commandes utilisées soit grâce à la commande `history`, soit grâce à la commande `cat .bash_history`. C'est pas la question. quand je tape ls, comment bash sait qu'il doit executer /bin/ls et pas un autre programme? => Variable PATH

2) La variable d'environnement permettant à la commande *cd* tapée sans argument de nous ramener dans votre répertoire personnel est ~. ~ n'est qu'un raccourci qui n'est pas magique il correspond à une variable d'environnement. => HOME

3)

- variable LANG : langue du système, utilisée tant qu'elle n'est pas contredite par une autre variable.
- variable PWD : Répertoire courant de l'interpréteur de commande.
- variable OLDPWD : Répertoire courant avant l'exécution de la précédente commande *cd*
- variable SHELL : indique l'interpréteur de commande utilisé
- variable _ : indique le dernier paramètre/argument (démonstration :

```
ls -a

echo "$ _" // affiche -a

ls -l

echo "$ _" // affiche -l)
```

dernier argument
ps
echo "$?" affiche ps

Dans le rendu pdf il y a des espaces qui se sont glisser attention a votre compte rendu en markdown qui doit aussi les contenir. ça rend faux les bouts de code donné

4) création d'une variable locale : `CYP = 'HELLO WORLD'` Pour vérifier que la variable à bien été crée, on fait `echo "$ CYP"`. Cela nous affiche bien HELLO WORLD.

5) La commande `bash` ouvre une nouvelle session dans le terminal. En tapant `echo "$ CYP"`, on s'apercoit que la variable crée précedemment n'existe pas. On reviens dans la session initiale par la commande `exit`.

6) On transforme CYP en une variable d'environnement : `export CYP`. En réexecutant `bash` puis `echo "$ CYP"`, on remarque que cette fois la variable existe.

C'est la différence entre variable locale et variable d'environnement : Une variable locale n'existe que sur la session en cours, alors qu'une variable d'environnement est existe pout tous les environnements issus de l'environnement courant.

7) On crée maintenant une nouvelle variable d'environnement :

```
export NOMS = "CYPRIEN GALLAND ET CLEMENT VATIN"
```

En faisant `echo "$ NOMS"`, on peut vérifier le contenu de NOMS (à savoir CYPRIEN GALLAND ET CLEMENT VATIN)

8) On souhaite afficher ”Bonjour à vous deux", suivi du contenu de NOMS. On exécute pour cela la commande `echo "Bonjour à vous deux $ NOMS"`.

9) Donner une valeur vide à une variable fait que la variable continue d'exister. En revanche, utiliser `unset` permet non seulement de vider la variable, mais aussi de la supprimer.

10) Pour écrire la phrase $ HOME =chemin (où chemin est notre dossier personnel d’après bash), on utilise la commande `echo $ PWD`.

non ce n'est pas ce qui est attendu. echo "\$HOME=$HOME"
PWD dépend de là où je suis actuellement, rien a voir avec HOME

Programmation bash

On enregistre à partir de maintenant nos scripts dans un dossier script, que vous créons dans votre répertoire personnel. On ajoute à bash le chemin vers notre dossier script de facon permanente grâce à `PATH = $ PATH : ~/script/`.

Exercice 2. Contrôle de mot de passe

Script permettant de saisir un mot de passe, puis de vérifier si il correspond au contenu d’une variable PASSWORD dont le contenu est codé en dur dans le script, le tout sans afficher le mot de passe saisi par l'utilisateur :

```
echo "Entrez votre mot de passe non crypté"

read -s password // read permet de lire une variable saisie par l'utilisateur; La variable -s permet de faire une saisie cachée.

pass="test"

if [ $pass == $password ];

then echo MOT DE PASSE CORRECT $password

else

echo MOT DE PASSE INCORRECT

fi
```

Attention en markdown pour avoir des bloc code correct
```
votre code sur plusieurs ligne
une autre ligne
```
Attention si pass est vide => erreur de syntaxe car ce qui sera lu sera
if [= test]
un bon moyen d'éviter ça c'est d'écrire
x$pass = x$password ce qui donnera avec pass vide:
if [x = xtest]

Une difficulté de l'exercice est de manipuler soit la variable soit son contenu avec $.

Exercice 3. Expressions rationnelles

Script qui prend un paramètre et utilise une fonction pour vérifier que ce paramètre est un nombre réel, en retournant un message d'erreur si ce n'est pas le cas.

```
function is_number()

{

re=' ^ [+-]?[0-9]+([.][0-9]+)?$'

if ! [[ $1 =~ $re ]] ; then
```

```
return 1 // 1 = False, 0 = True
else
return 0
fi
}
if [ "$1" != "" ] ; then
echo "$1"
else
echo "pas d'arguments"
fi
is_number $1
if [ $? == 0 ] ; then // $? désigne le résultat de la dernière fonction appelée.
echo "C'est un nombre !!!!!
else
echo "Ce n'est pas un nombre !!!!!
fi
```

peut s'écrire plus simple:
if is_number $1; then

On utilise ici `$?` pour récupérer le retour du dernier programme executé pour connaitre le résultat de `is_number`.

Exercice 4. Contrôle d'utilisateur

Script vérifiant 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 alors le message : "Utilisation :nom_du_script nom_utilisateur" (le nom de script étant récupéré dynamiquement).

```
me="$(basename "$(test -L "$0" && readlink "$0" || echo "$0")")"
if [ "$1" != "" ] ; then
echo "$1"
else
echo "Utilisation : $ me User"
fi
aze=":"
```

bien compliquer un simple "$0" aurait suffit

```
var=$ 1

chaine=$ var$ aze

echo $ chaine

grep -c -e "^$chaine" /etc/passwd

if [ $? -eq 0 ] ; then

echo "L'utilisateur existe"

else

echo "Cet utilisateur n'existe pas"``fi
```

On exploite le contenu du fichier /etc/passwd pour savoir si l'utilisateur existe ou non.

Exercice 5. Factorielle

Programme qui calcule la factorielle d'un entier naturel passé en paramètre

```
function factorial()

{

if (( $1 < 2 )); then

echo 1

else

echo $(( $1 * $(factorial $(( $1 - 1 ))) ))

fi

}

factorial $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.

$1 représente ici le nombre entré sur la ligne de commande.

Exercice 6. Le juste prix

Script qui génère un nombre aléatoire entre 1 et 1000, et demande à l'utilisateur de le deviner, à l'aide d'indications de type "plus" ou "moins"

`nb_a_deviner=$(( $RANDOM % 1000 +1 ));` // % 1000 permet de génerer entre 0 et 999. Pour aller de 1 à 1000, on rajoute donc +1.

```
test=0

while [ $test == 0 ]

do
```

```
echo "Entrez un nombre "

read nombre

if [ $ nombre == $ nb_a_deviner ]

then

test=1

elif [ "$ nombre" -gt "$ nb_a_deviner" ]

then

echo "c'est moins"

else

echo "c'est plus"

fi

done

echo "C'est gagné"
```

ce test là pourrait être la condition de sortie de la boucle et le test ne regarder que plus grand ou plus petit

On utilise `-gt` pour comparer des nombres et non des chaines de caractères(sinon on aurait utilisé >).

Exercice 7. Statistiques

1) 2) Script prenant en paramètres trois entiers (entre -100 et +100) et affichant le min, le max et la moyenne, puis généralisation à un nombre quelconque de paramètres :

```
function is_number()

{

re='^ [+-]?[0-9]+([.][0-9]+)?$'

if ! [[ $1 =~ $re ]] ; then

return 1

else

return 0

fi

}

for rep in $ *

do

echo "Regardons si $rep est un entier"
```

```
if [ "$1" != "" ] ; then
echo "$1"
else
echo "pas d'arguments"
fi
is_number $rep
if [ $ ? == 0 ] ; then
echo "C'est un nombre !!!!!"
else
echo "Ce n'est pas un nombre, interruption de la commande"
exit 1
fi
min=$ rep
max=$ rep
mean=0
done
for rep in $          in $???
do
mean=$ (($ mean+$ rep))
if [ "$ rep" -gt "$ max" ]
then
max=$ rep
elif [ "$ rep" -lt "$ min" ]
then
min=$ rep
fi
done
mean=$ (($ mean / $#))
```

```
echo "Le max est $max,le min est $min, la moyenne est $mean"
```

On réutilise ici les fonctions des exercices précédents

3) Généralisation en saisissant et stockant les entiers au fur et à mesure dans un tableau :

```
function is_number()

{

re='^ [+-]?[0-9]+([.][0-9]+)?$'

if ! [[ $1 =~ $re ]] ; then

return 1

else

return 0

fi

}

nb_entre=0

declare -a myarray

is_number $nb_entre

while [ $ ? == 0 ]

do

echo "Entrez un nombre pour l'ajouter à la liste, ou une lettre pour arrêter la
saisie"

myarray+=( "$nb_entre" )

read nb_entre

is_number $ nb_entre

done

echo "fin de saisie"

unset myarray[0]

max=$ {myarray[0]}

min=$ {myarray[1]}

echo "min vaut $ min"

mean=0
```

manque des choses nb_entre = 0 et il est ajouté au tableau directement

```
for (( index=0;index<=$ {#myarray[@]};index++ ))

do

echo "$ {myarray[1]}"

mean=$ [$ {myarray[$ index]}+$ mean]

k=$ {myarray[$ index]}

j=$(( $k ))

echo "j vaut : $j"

if [[ $j > $ max ]]

then

max=$ {myarray[$index]}

elif [[ $j < $ min ]]

then

min=$ {myarray[$index]}

fi

done

echo "la somme vaut $ mean"

mean=$ [$ mean / ${#myarray[@]}]

echo "Le max est : $max , le min est $min, la moyenne est $mean "

echo "the array contains ${#myarray[@]} elements"
```

Cette variante est plus compliquée car elle ajoute la manipulation de tableau, ce qui nescessite une synthax appropriée.