# TP2 Administration Système

Pierre Dubreuil
Lucie Mourer

07/02/2020

# Exercice 1

## Variables d'environnement

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

Résultat de `printenv PATH`

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/usr/local/cpe/bin:/softwares/bin:/softs/bin
```

### 2. Quelle variable d'environnement permet à la commande cd tapée sans argument de vous ramener dans votre répertoire personnel ?

```
cd $HOME
```

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

`LANG` permet de connaitre le paramètre de langue de base utilisé par les applications du système. Il peut être modifié.
`PWD` donne le repertoire de travail courant du terminal.
`OLD PWD` donne le chemin précédent du repertoire de travail du terminal.
`SHELL` donne le chemin de l'interpréteur de commande préféré de l'utilisateur.
`_` Contient le dernier paramètre de la commande.

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

```
VARLOC="Helloe"
echo $VARLOC
```
---> `Hello`

### 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.

```
bash
echo $VARLOC
```

---> **Cela n'affiche rien car on a changé de session de terminal et on avait créé la variable en locale, donc juste sur la première session du shell.**

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

```
export VARENV="HELLO"
printenv VARENV
```
---> `cela affiche HELLO`

```
bash
printenv VARENV
```
---> **Cela affiche HELLO car la variable d'environnement peut être comprise comme une variable globale reconnue par toutes les sessions du shell**

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="DUBREUIL MOURER"
printenv NOMS
```
**---->** `DUBREUIL MOURER`

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!"
```

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

`unset` supprime la variable et donc désalloue la mémoire qui lui était donnée. Donner une valeur vide à une variable fait qu'elle existe toujours mais quand on fait appel à elle, on pointe sur une valeur vide.

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

```
echo "\$HOME=$HOME"
```
**--->** `$HOME=/fs03/share/users/pierre.dubreuil/home`

## Programmation Bash

```
mkdir script
cd script
export PATH=$PATH:~/script
```

# 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.*

```
nano testpwd.sh
```

```
#!/bin/bash

PASSWORD="HELLO"
read -s -p "Saisissez votre mot de passe: " pwd

if [ $pwd = $PASSWORD ]
then
    echo "C'est le bon mot de passe !"
else
    echo "Nope !"
fi
```

On réalise la saisi du mot de passe que l'on stocke dans la variable `pwd`
On réalise la vérification selon une instruction conditionnelle vérifiant l'égalité entre le contenu de `pwd` et du `PASSWORD` initialisé dans le code.

# Exercice 3. Expressions rationnelle

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

```
#!/bin/bash

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

is_number $1

if [ $? == 0 ] ; then
    echo "C'est un nombre réel"
else
    echo "Ce n'est pas un nombre réel"
fi
```

`$1` représente le contenu du paramètre saisi dans le shell la fonction `is_number` permet de vérifier que le paramètre saisi est un nombre réel grâce à une expression régulière.

# 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)*

```
#!/bin/bash

function is_user()
{
    users=$(cut -d: -f1 /etc/passwd)

    for user in $users ; do
        if [ $1 == $user ] ; then
            return 1
        fi
    done

    return 0
}
if [ -z "$1" ] ; then  #-z permet de vérifier si le contenu de $1 est vide
    echo "Utilisation: ${0##*/} nom_utilisateur"
else
    is_user $1
    if [ $? == 1 ] ; then
        echo "Cet utilisateur existe"
    else
        echo "Cet utilisateur n'existe pas"
    fi
fi
```

En premier lieu, on vérifie si l'utilisateur a bien rentrer un paramètre. Si ce n'est pas le cas on lui indique l'utilisation du programme. `${0##*/}` permet de récupérer le 1er paramètre de la commande tapée soit le nom du fichier lancé auquel on retire le `./` si le fichier n'a pas été exécuté en tant que commande.
Ensuite, on vérifie que l'utilisateur existe grâce à la fonction `is_user`.
Pour ce faire, on récupère les noms d'utilisateurs dans le fichier `passwd` grace à la commande `$(cut -d: -f1 /etc/passwd)`.
On itère ensuite sur la liste récupérée en vérifiant que le nom rentré est dans cette liste.

# 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).*

```
GNU nano 2.2.6 Fichier : facto.sh

#!/bin/bash

nb=$1
fact=1

while [ $nb != 0 ]
do
    fact="$((fact * nb))"
    nb="$((nb - 1))"
done
echo $fact
```

On réalise des fonctions de calcul dans ce sritpt entre des entiers avec les doubles `(())` et on affiche le résultat dans le shell avec `echo $fact`.

# 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

nbrdm=$(( RANDOM % 1000 ) + 1 ))
tentatives=0

while [ "$nb" != "$nbrdm" ] ; do
    read -p "Entrez votre nombre: " nb
    if [ $nb -lt $nbrdm ] ; then
        echo "C'est plus"
    elif [ $nb -gt $nbrdm ] ; then
        echo "C'est moins"
    else
        echo "C'est gagné"
        break
    fi
    tentatives="$((tentatives + 1))"
    if [ "$tentatives" = 10 ] ; then
        echo "C'est perdu !"
        echo "C'était $nbrdm"
        break
    fi
done
```

`-lt` est un paramètre qui représente le < dans la sructure conditionelle. `-gt` représente >. On aurait pu utiliser le paramètre `-eq` pour vérifier l'équivalence entre la valeur saisie et le nombre cherché.

`$(( RANDOM % 1000 ) + 1 ))` permet de récupérer un nombre aléatoire compris entre 1 et 1000.
On choisi de laisser un nombre de tentative limitée pour donner une fin au jeu.

# 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.*

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

```
#!/bin/bash
LC_NUMERIC=C

function is_int()
{
    re='^[+-]?[0-9]+$'
    ret=0
    if ! [[ $1 =~ $re ]] ; then
        ret=0
    else
        if [[ $1 -lt 100 && $1 -gt -100 ]] ; then
            ret=1
        fi
    fi
    return $ret
}

min=""
max=""
moy=0
i=0

for nb in "$@"
do
    is_int $nb
    if [ $? = 1 ] ; then
        if [ -z $min ] ; then
            min=$nb
        fi
        if [ -z $max ] ; then
            max=$nb
        fi
        if [ $nb -lt $min ] ; then
            min=$nb
        elif [ $nb -gt $max ] ; then
            max=$nb
        fi
```

```
        moy="$((moy + nb))"
        i="$((i + 1))"
    else
    echo "$nb n'est pas conforme, il doit être entier entre -100 et 00. Il
n'est pas pris en compte"
    fi
done


moy=$(echo "$moy / $i" | bc -l)
echo "Le min est $min"
echo "Le max est $max"
printf "La moyenne est %.3f \n" $moy
```

Dans ce sript on assure que le nombre est entier et compris entre -100 et 100 avec des paramètres déjà utilisés précédement. La difficulté de cet excercice était surtout présente sur le calcul de moyenne avec des nombres flottants car bash ne fait des divisions qu'avec des entiers à l'origine. On doit rajouter les paramètres `| bc -l` à la commande pour mettre le contenu de la division dans `moy`, ainsi que définir le paramètre `LC_NUMERIC=C`
Pour limiter à 3 le nombre de chiffres après la virgule du nombre flottant qui seront affiché on utilise `printf "%.3f"`

## 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.*

```
#!/bin/bash
LC_NUMERIC=C

function is_int() {
    re='^[+-]?[0-9]+$'
    ret=0
    if ! [[ $1 =~ $re ]]; then
        ret=0
    else
        if [[ $1 -lt 100 && $1 -gt -100 ]]; then
            ret=1
        fi
    fi
    return $ret
}

min=""
max=""
moy=0
i=0
j=0
tab=()

read -p "Combien de nombre voulez-vous rentrer ?
```

```
 " nbnb

while [ $j -lt $nbnb ]; do
    read -p "Nombre $((j + 1)) : " nb
    is_int $nb
    if [ $? = 1 ]; then
        tab[$j]=$nb
        if [ -z $min ]; then
            min=$nb
        fi
        if [ -z $max ]; then
            max=$nb
        fi
        if [ $nb -lt $min ]; then
            min=$nb
        elif [ $nb -gt $max ]; then
            max=$nb
        fi
        moy="$((moy + nb))"
        i="$((i + 1))"
        j="$((j + 1))"
    else
        echo "$nb n'est pas conforme, il doit être entier entre -100 et
100. Il n'est pas pris en compte"
    fi

done

moy=$(echo "$moy / $i" | bc -l)
echo "Le min est $min"
echo "Le max est $max"
printf "La moyenne est %.3f \n" $moy
echo ${tab[*]}
```

Pour ce qui est de la gestion du tableau et de son affichage, on instantie un tableau vide que l'on remplie au fur et à mesure grâce à la commande `tab[$j]=$nb`.
Le nombre de colonne est équivalente au nombre de nombres `nbnb` saisi dans le shell. On fait une boucle tant que `j` est inférieur ( `-lt` ) au nombre `nbnb` et on rentre dans le tableau `tab` à la colonne `j` le nombre saisi correspondant à la position de `j` .
Le reste du scrip reste sensiblement le même.

# Exercice 8.

*Écrivez un script qui affiche les combinaisons possibles de couleurs*

```
#!/bin/bash

foreground=(1 4 5 7 30 31 32 33 34 35 36 37)
background=(40 41 42 43 44 45 46 47)
```

```
printf "FG \ BG   "
for bg in "${background[@]}"
do
    printf "   $bg"
done
printf "\n"

for fg in "${foreground[@]}"
do
    printf "$fg         "
    if [ $fg -lt 10 ] ; then
        printf " "
    fi
    for bg in "${background[@]}"
    do
        echo -en "\e[${bg};${fg}mBash \e[0m"
    done
    printf "\n"
done
```

On commence par définir les tableaux de couleurs disponibles pour l'arrière et le premier plan.
Ensuite on affiche la première ligne du tableau qui sera notre légende des couleurs d'arrière plan.
Ensuite, on itère sur les couleurs du premier plan puis sur l'arrière plan où pour chacune de ces dernière on affiche la ligne.

```
if [ $fg -lt 10 ] ; then
    printf " "
fi
```

Ce test nous permet juste d'aligner les lignes les unes aux autres.