



B-

# TP n°2 Administration Système

---

## Variables d'environnement

1- Afin de trouver l'ensemble des dossiers  bash contenant les commandes tapés par l'utilisateur, on cherche à afficher le contenu de la variable d'environnement *PATH*. Dans un premier temps, il est important de noter la différence entre variable d'environnement et variable locale. Une variable est dite locale lorsqu'elle n'existe que dans le terminale (ou bash) courant. A l'inverse, une variable d'environnement, comme son nom l'indique existe au travers de l'ensemble des bashes instanciés par l'utilisateur, quelques soit leur nombre.

Par la suite, pour revenir à la question d'origine, on  la commande : *printenv PATH* . Le résultat affiché à l'écran est :

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
```

2- La variable d'environnement associée au répertoire personnel est *HOME* .

Commandes :

- *cd \$HOME* => retourne dans repertoire perso
- *pwd* => permet de s'en assurer en affichant l'emplacement du répertoire courant

3- Les variables explicitées sont :



- *HOME* => contient la location du répertoire personnel de l'utilisateur
- *LANG* => définit la langue par défaut
- *PWD* => pointe vers le répertoire courant (sans argument, équivalent à *pwd*)
- *OLDPWD* => contient le répertoire précédent la dernière commande *cd* ( équivalent à *cd -*)
- *SHELL* => contient la localisation de l'interpréteur de commande
- *\_* => contient la dernière variable d'environnement exécutée

**Note** : *printenv | grep VAR* => recherche une variable (locale/d'environnement) qui contient la chaîne de caractère **VAR**. Par exemple **MYVAR = toto** .

4- Commandes :

- Création d'une variables locale : *MY\_VAR="abc"*
- Vérification de son existence : *printenv | grep MY\_VAR*

5- On observe que, après avoir tapé la commande *bash* (*bash* est un interpréteur de langage de commande *sh-compatible* qui exécute les commandes depuis l'entrée standard ou depuis un fichier), la variable locale *MY\_VAR* n'existe pas, en accord avec l'explication explicitée en 1- .

6- En revanche une variable d'environnement existe même après avoir tapé  la commande *bash*. Pour créer une variable d'environnement, on  la commande *export MY\_VAR2=tototo* ou *declare -x MY\_VAR=tototo* .

7-Il est possible de créer aussi des variables d'environnement avec un espace, en procédant de la manière suivante :

- `export NOMS="tom ludo"` => création de la variable d'environnement avec un espace
- `printenv | grep NOMS` => vérification de l'existence de cette même variable

8- Il est possible, au travers de la commande `echo`, d'afficher des chaînes de caractères concaténées, et même de concaténer des chaînes de caractères avec des variables contenant des chaînes de caractères. Par exemple :

- `echo bonjour a vous deux, $NOMS` => affiche "bonjour a vous deux tom ludo"

9- Pour supprimer une variable, on utilise la commande `unset`. De ce fait, on note, par exemple : `unset NOM`. Il est important de noter que la création d'une variable "vide" et la suppression de cette dernière est totalement différent du point de vue du système. La création d'une variable engendre un lien entre cette dernière et une case mémoire : la variable créée prend une place respective dans la mémoire du système (même si la valeur est vide). La suppression d'une variable, quant à elle, engendre la libération d'une case mémoire du système (en soit, la "place" est libérée en mémoire du système).

10- Commande:

- `echo '$HOME'=$HOME` => permet d'afficher \$HOME = /home/tomludo (répertoire personnel)

## Scripts

### Exercice n°2

Énoncé : 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.

**Script :**

```
#!/bin/bash                                #permet de commencer un script bash

read -s -p 'Entre mdp:' mdp                #demande à l'utilisateur de rentrer un
mot de passe

if [ $mdp = "password" ]; then              #vérification si le mot de passe est bon
    echo "mdp correct"
else
    echo "mdp incorrect"
fi                                           #permet de fermer une boucle "if"
```

### Exercice n°3

Énoncé : Écrivez un script qui prend un paramètre et utilise la fonction suivante pour vérifier que ce paramètre est un nombre réel. Il affichera un message d'erreur dans le cas contraire.

**Script :**

```
#!/bin/bash

function is_number

{
    re='^[+-]?[0-9]+([.][0-9]+)?$'      #identification de la signature
d'un nombre réel

    if [[ $1 =~ $re ]] ; then          #test si le premier argument de
la fonction est un réel

        echo "reel"

        return 1

    else

        echo "error"

        return 0

    fi

}

read -p 'Saisir un nombre réel' a      #saisi du nombre par l'utilisateur

echo "  on a $( is_number $a ) "
```

**Exercice n°4**

Enoncé : 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\_scriptnom\_utilisateur", où *nom\_du\_script* le nom de votre script récupéré automatiquement (si vous changez le nom de votre script, le message doit changer automatiquement).

**Script :**

```
#!/bin/bash

if [ $# -eq 0 ]; then

    echo" $0 nom_utilisateur "

else

    while(("${#}")); do
```

```
a=$(grep -w "$1" /etc/passwd|wc -l)

if [$a -eq 0 ]; then

    echo "l'utilisateur n'existe pas"

else

    echo "l'utilisateur existe"

fi

shift

done

fi
```

### Exercice n°5

Enoncé : Écrivez un programme qui calcule la factorielle d'un entier naturel passé en paramètre (on supposera que l'utilisateur saisit toujours un entier naturel).

**Script :**

```
#!/bin/bash

facto() {

n=$1

if [[ $n = 0 ]]; then

    echo 1

else

    echo$(( n * $( facto $(( n-1 )) ) ))

fi

}

echo$( facto $1 )
```

### Exercice n°6

Enoncé : 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).

**Script :**

```
#!/bin/bash

n=$(( ( $RANDOM % 1000 ) + 1 ))

read -p 'proposer un nombre' nb

while [[ $nb != $n ]]

do

if [[ $nb < $n ]];then

    echo " c est +"

    read -p 'propose un nombre' nb

else

    echo " c est -"

    read -p 'propose un nombre' nb

fi

done

echo ' c est ca, bien joué '
```



## Exercice n°7

Enoncé :

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)
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.

**Script :**

```
#!/bin/bash

read -p 'entre 3 entiers entre -100 et 100' a b c

if [[ $a < $b ]]; then
```

```
    if [[ $b < $c ]]; then
        echo " le max est $c et le min est $a "
    else
        if [[ $a < $c ]]; then
            echo " le max est $b et le min est $a "
        else
            echo "le max est $b et le min est $c"
        fi
    fi
else
    if [[ $a < $c ]]; then
        echo " le max est $c et le min est $b "
    else
        if [[ $b < $c ]]; then
            echo" le max est $a et le min est $b"
        else
            echo " le max est $a et le min est $c "
        fi
    fi
fi

echo "moyenne : $(( ($a + $b + $c) / 3))"
```