Introduction à la Programmation en Shell Bash

I. Introduction :

Sous Unix, on appelle ***shell*** l'interpréteur de commandes qui fait office d'interface entre l'utilisateur et le système d'exploitation. Les shells sont des interpréteurs : cela signifie que chaque commande saisie par l'utilisateur (ou lue à partir d'un fichier) est syntaxiquement vérifiée puis exécutée.

I.1. Fichiers Shell, script shell

Lorsqu'un traitement nécessite l'exécution de plusieurs commandes, il est préférable de les sauvegarder dans un fichier plutôt que de les retaper au clavier chaque fois que le traitement doit être lancé. Ce type de fichier est appelé ***fichier de commandes*** ou ***fichier shell*** ou encore ***script shell***.

La notation **#!** en première ligne d'un fichier script shell précise au shell courant quel interpréteur doit être utilisé pour l'exécuter (par exemple pour spécifier un interpréteur bash, la première ligne doit être **#!/bin/bash**).

Il est à remarquer que *Bash* distingue les caractères majuscules des caractères minuscules.

Un commentaire débute avec le caractère # et se termine avec la fin de la ligne. Un commentaire est ignoré par le shell.

- **Création d'un script shell :** La création d'un script nécessite les étapes suivantes

1- Ouvrir un fichier texte sous unix (Utiliser **Vi nom_script**)

2- Commencer à écrire le script shell en commencant par taper sur la première ligne **# !/bin/bash**

- **Exécution d'un script shell :** l'exécution peut se faire de deux façons

1. Il s'agit d'un fichier ordinaire qui n'a pas le droit d'exécution, il faut alors changer ses droits :

```
$ chmod u+x nom_script
```

Puis, l'exécution se fera en tapant :

```
$ ./nom_script
```

2. Taper directement

```
$ bash mon_fichier_script_bash
```

Lors de l'appel d'un script, ce dernier peut avoir un ensemble d'arguments:

```
$ bash mon_fichier_script_bash argument1 argument2 …
```

Exemple :

```
$ bash script_calcul_somme 5 3
```

La commande **echo**

Cette commande affiche ses arguments sur la sortie standard en les séparant par **un espace** et en terminant par **retour à la ligne**.

Exemple :

```
echo bonjour tout le monde
bonjour tout le monde
```

Dans cet exemple, la commande **echo** est invoquée avec quatre arguments : ***bonjour***, ***tout***, ***le*** et ***monde***. On remarquera que les espacements entre les arguments ne sont pas conservés lors de l'affichage : un seul caractère **espace** sépare les mots affichés. En effet, le shell prétraite la commande, éliminant les ***blancs*** superflus.

Pour conserver les espacements, il suffit d'entourer la chaîne de caractères par une paire de **guillemets** :

Exemple :

```
$ echo "bonjour tout le monde"
bonjour tout le monde
```

On dit que les *blancs* ont été protégés de l'interprétation du shell.

Remarque :

```
echo `commande` => Affiche le résultat de l'exécution echo
$(commande) => Affiche le résultat de l'exécution
echo $var                => Affiche la valeur d'une variable
echo "message"          => Affiche le mot message
echo "message1 $var message2 `cmd`..." => Affiche message1 suivi de la valeur de la variable var
suivi de message2 suivi du résultat de l'exécution de commande cmd…
```

Exemple : script hello_world

```
$ cat > hello.sh
# !/bin/bash
# Un premier programme shell        (commentaire)
echo "Hello World !"
CTRLD + D
$ bash hello_word.sh      Exécution du script:
Hello World !                       (Résultat du script)
```

II. Les variables

Une variable est une case mémoire qu'on désigne par un nom, qu'on peut afficher sa valeur (écriture ou affichage). La variable peut prendre une valeur soit d'une manière directe dans le programme (affectation =), soit à partir d'une entrée standard (lecture « read »). La variable peut être utilisée dans des expressions à évaluer.

- **Affectation directe**

Syntaxe : *nom*=[*valeur*]

Exemple :

X=bonjour

echo $x *affichera* bonjour *sur la sortie standard qui est l'écran*

Il est impératif que le nom de la variable, le symbole = et la valeur à affecter ne forment qu'une seule chaîne de caractères (sans espace).

- **Affectation par lecture**

Elle s'effectue à l'aide de la commande interne **read**. Celle-ci lit une ligne entière sur l'entrée standard.

Syntaxe : read [*var1* ...]

Exemple :

echo "Écrivez votre nom puis votre prénom : " read nom prenom echo "Nom : $nom" echo "Prénom : $prenom"	**Ce qui donnera :** Écrivez votre nom puis votre prénom : Hugo Victor Nom : Hugo Prénom : victor

L'option **–p** de **read** affiche une chaîne d'appel avant d'effectuer la lecture ; la syntaxe à utiliser est **read –p** *chaîne_d_appel* **[** *var* **…]**

Exemple :

$ read -p "Entrez votre prénom : " var_prenom

Entrez votre prenom :

Manel $ echo $var_prenom

Manel

- **Suppression des ambiguïtés**

Pour éviter les ambiguïtés dans l'interprétation des références de paramètres, on utilise la syntaxe **${***paramètre***}**.

Exemple :

```
$ x=bon
$x1=jour
$ echo $x1 => valeur de la variable x1
jour
$ echo ${x}1 => pour concaténer la valeur de la variable x à la chaîne "1"
bon1
```

II.1. Paramètres de position et paramètres spéciaux

- **Paramètres de position :**

Un **paramètre de position** est référencé par un ou plusieurs chiffres : 1, 2, … L'assignation des valeurs à des paramètres de position s'effectue :

- soit à l'aide de la commande interne **set.**
- soit lors de l'appel d'un fichier script shell .

Remarque : Nous nous limiterons dans ce manuscrit à l'assignation de valeurs aux paramètres de position lors de l'appel d'un fichier de script shell.

Dans un fichier script shell, les paramètres de position sont utilisés pour accéder aux valeurs des arguments qui ont été passés lors de son appel : cela signifie qu'au sein du fichier shell, les occurrences de **$1** sont remplacées par la valeur du premier argument, celles de **$2** par la valeur du deuxième argument, etc.

Remarque *:* **$0** contient le nom complet du programme shell qui s'exécute

- **Paramètres spéciaux :**

Un **paramètre spécial** est référencé par un ***caractère spécial***. L'affectation d'une valeur à un paramètre spécial est effectuée par le shell. Pour obtenir la valeur d'un paramètre spécial, il suffit de placer le caractère **$** devant le caractère spécial qui le représente.

Un paramètre spécial très utilisé est le paramètre # (à ne pas confondre avec le début d'un commentaire). Celui-ci contient le nombre de paramètres de position ayant une valeur. Pour obtenir sa valeur on utilisera **$#**

Le paramètre spécial * contient la liste des valeurs des paramètres de position initialisés. Pour obtenir cette liste on utilisera **$***.

Les paramètres spéciaux :	On en déduit alors :
0 désigne le nom du script	**$0** désigne la valeur du nom du script
# désigne le nombre de paramètres	**$#** désigne la valeur du nombre de paramètres
* désigne la liste des paramètres	**$*** désigne la valeur de la liste des paramètres
	$**i** désigne la valeur du paramètre n°i

Exemple 1 :

```
$ cat > list.sh
# !/bin/bash
echo "contenu du répertoire $1"
ls $1
echo "contenu du répertoire $2"
ls $2
CTRLD +D
```

L'exécution de la commande suivante :

$ bash list /home /isamm

Donnera le contenu du répertoire **home (**donné en premier paramètre et désigné par $1**)** suivi du contenu du répertoire **isamm (**donné en deuxième paramètre et désigné par $2**)**

Exemple 2 : programme shell copie

```
$ cat > copie.sh
#!/bin/bash
# Programme de copie depuis le premier argument vers le second
argument echo "Nom du programme : $0"
echo "Nb d'arguments : $#"
echo "Les arguments sont : $*"
echo "Source : $1"
echo "Destination : $2"
cp $1 $2
CTRL + D
```

III. Opérations sur les variables

III.1. La commande test :

La commande test est utilisée dans les conditions. Elle permet d'évaluer une expression, elle retourne 0 si la condition est vrai et 1 sinon.

Syntaxe 1: test expression

Syntaxe 2: [expression]

III.1.1 Expression sur les fichiers :

- -e vrai si le fichier existe — ex : [-e $var]
- -s vrai si le fichier existe et de taille différente de 0. — ex : [-s $var]
- -r vrai si le fichier existe et il est accessible en lecture. — ex : [-r $var]
- -w vrai si le fichier existe et il est accessible en écriture. — ex : [-w $var]

- -x vrai si le fichier existe et il est accessible en exécution. ex : [-x $var]
- -f vrai si le fichier existe et il est régulier. ex : [-f $var]
- -d vrai si le fichier existe et il est un répertoire. ex : [-d $var]

III.1.2 Chaîne de caractère :

- = vrai si elles sont égales. ex : ["$var1" = "$var2"]
- != vrai si elles sont différentes. ex : ["$var1" != "$var2"]
- -z vrai si elle est vide. ex : [-z "$var1"]
- -n vrai si elle n'est pas vide. ex : [-n "$var1"]

Remarque : Il est nécessaire d'entourer les variables de guillemets dans les comparaisons

III.1.3 Nombres et comparaisons numériques:

- -eq vrai si égale ex : [$nbr1 -eq $nbr2]
- -ne vrai si n'est pas égale ex : [$nbr1 -ne $nbr2]
- -gt vrai si supérieur ex : [$nbr1 -gt $nbr2]
- -lt vrai si inférieur ex : [$nbr1 -lt $nbr2]
- -ge vrai si supérieur ou égale ex : [$nbr1 -ge $nbr2]
- -le vrai si inférieur ou égale ex : [$nbr1 -le $nbr2]

Remarque : Toutes les variables sont considérées comme des chaînes, leur valeur est convertie en nombre pour les opérateurs de conversion numérique.

III.1.4 Les opérateurs logiques:

- !négation
- -a l'opérateur et
- -o l'opérateur ou
- (exp) parenthèse

Exemple : Pour tester si rep1 est un répertoire et s'il est accessible en écriture : [-d "rep1" -a -w "rep1"]

Attention : on prendra soin de séparer les différents opérateurs et symboles par des **espaces**

III.2. La commande expr :

Permet d'évaluer une expression arithmétique.

- expr n1 + n2 addition
- expr n1 – n2 soustraction
- expr n1 / n2 division
- expr n1 % n2 modulo
- expr n1 * n2 multiplication
- expr n1 (\< , \> , =, != , \>= ,\<=) n2 permet de faire la comparaison

Exemple : var=`expr $var + 1`

IV. Expression de contrôle if

La commande interne **if** implante le choix alternatif.

Syntaxe :

if ***suite_de_commandes_test***

then ***suite_de_commandes1***

[elif ***suite_de_commandes***

then ***suite_de_commandes*** **] ...**

[else ***suite_de_commandes***

] fi

Le fonctionnement est le suivant : ***suite_de_commandes_test*** est exécutée ; si son code de retour est égal à **0 (vrai),** alors ***suite_de_commandes1*** est exécutée sinon c'est la branche **elif** ou la branche **else** qui est exécutée, si elle existe.

La structure de contrôle doit comporter autant de mots-clés **fi** que de **if** (une branche **elif** ne doit pas se terminer par un **fi**).

Exemple 1:

```
$ echo "coucou" >toto
$ chmod 200 toto
$ ls -l toto
- -w- --- --- 1 user1 isamm 7 déc 17 17:21
toto $ cat > ex1.sh
#!/bin/bash
echo "test et affichage du droit de lecture sur le premierargument"
if [ -r $1 ]
then cat$1
else
echo "$1 n'est pas accessible en lecture "
echo "attribution du droit de lecture etaffichage"
chmod u+r $1
cat$1
fi
CTRL +D
$ bash ex1.sh toto     #exécution
test et affichage du droit de lecture sur le premier argument
```

```
toto n'est pas accessible en lecture
attribution du droit de lecture et
affichage coucou
```

V. Choix multiple case

Syntaxe :

case ***mot*** **in**

[***modèle*** **[| *modèle*] ...)** ***suite_de_commandes*** **;;] ...**

esac

Le shell évalue la valeur de ***mot*** puis compare séquentiellement cette valeur à chaque modèle. Dès qu'un modèle correspond à la valeur de ***mot***, la suite de commandes associée est exécutée, terminant l'exécution de la commande **case**.
Les mots **case** et **esac** sont des mots-clé ce **;** ***suite_de_commandes*** doit se terminer par deux caractères **point-virgule** collés.
Un ***modèle*** peut être construit à l'aide des caractères et expressions génériques de **bash**. Dans ce contexte, le symbole | signifie OU.
Pour indiquer le cas par défaut, on utilise le modèle *. Ce modèle doit être placé à la fin de la structure de contrôle **case**.
Le code de retour de la commande composée **case** est celui de la dernière commande exécutée de *suite_de_commandes*.

Exemple 1 :

```
#!/bin/bash
# programme shell oui affichant OUI si l'utilisateur a saisi le caractère o ou O
read –p "Entrez votre réponse : " rep
case $rep in
o|O ) echo "OUI" ;;
*) echo "Indefini"
esac
```

Remarque: il n'est pas obligatoire de terminer par **;;** la dernière *suite_de_commandes*

Exemple 2 :

```
#!/bin/bash
case $# in
0) echo "$0 est sans arguments"  ;;
1) echo "$0 possède 1argument"  ;;
2) echo "$0 possède 2 arguments"  ;;
*) echo "$0 possède plus que 2 arguments"  ;;
```

```
esac
```

VI. Itération while

La commande interne **while** correspond à l'itération ***tant que*** présente dans de nombreux langages de programmation.

Syntaxe :

while ***suite_cmd1***

do

suite_cmd2

done

suite_cmd2 est exécutée autant de fois que le code de retour de *suite_cmd1* est égal à zéro (c.à.d. vrai).
L'originalité de cette structure de contrôle est que le test ne porte pas sur une condition booléenne (vraie ou fausse) mais sur le code de retour issu de l'exécution d'une suite de commandes.
En tant que mots-clé, **while**, **do** et **done** doivent être les premiers mots d'une commande.

Exemple : table de multiplication

```
$ cat > mult.sh
#! /bin/bash
# Table de multiplication d'un chiffre donné en argument
echo "Table de multiplication de$1"
i=0
while [ i –le 10]
do
echo "$1 x $i = `expr ($1 * $i)`"
i=`expr ($i + 1)` # i++
done
CTRL + D
$ bash mult.sh 5 # Exécution
Table de multiplication de 5
5 x 0 = 0
5 x 1 = 5
5 x 2 = 10
…
5 x 10 = 50
```

VII. Itération for

L'itération **for** possède plusieurs syntaxes dont les plus générales sont :

VII.1. Première forme pour itération for:

Syntaxe :	**Exemple :**
for var in liste_mots do liste de commandes done	for i in un deux trois do echo $i done **L'exécution donnera :** un deux trois

La variable *var* prend successivement la valeur de chaque mot de *liste_mots*.

VII.2. Deuxième forme pour itération for:

Syntaxe :	**Exemple :**
for var in $* do liste de commandes done	for i in $* do echo $i done **L'exécution de:** $bash fich le systeme unix **Donnera:** le systeme unix

Lorsque cette syntaxe est utilisée, la variable *var* prend successivement la valeur de chaque paramètre de position initialisé.

VII.3. Troisième forme pour itération for:

Syntaxe :	**Exemple :**
for var in * do liste de commandes done	for i in * do echo $i done **L'exécution permettra l'affichage du contenu du répertoire courant**

La variable *var* prend successivement la valeur du nom de chaque entrée (fichier/répertoire) dans le répertoire courant.

VII.4. Quatrième forme pour itération for:

Syntaxe :	**Exemple :**
for var in `cmd` do	for i in 'ls /home' do echo $i done

liste de commandes done	**L'exécution permettra l'affichage du contenu du répertoire home**

La variable *var* prend successivement la valeur du nom de chaque entrée produite par le résultat de l'exécution de la commande cmd.

VII.5. Remarques :

- **Evaluation d'une expression**

Res = $((*expression*))

Res = $[*expression*]

Res = `expr *expression*`

Exemple :

Ci-dessous les lignes qui permettent d'incrémenter la valeur d'une variable nommée nb de 1 sachant que sa valeur initiale est égale à 0.

nb=0 **nb=$(($nb +1))**	**nb=0** **nb=$[$nb +1]**	**nb=0** **nb=`expr $nb + 1`**

- **Exécution d'une commande**

Si la ligne de programme à écrire est une commande à exécuter elle sera notée de la manière suivante : `commande`

Exemple :

$ echo ls => Affichera la chaine ls sur la sortie standard.

$ echo `ls` => Affichera le résultat de l'exécution de la commande ls : le contenu du répertoire courant sur la sortie standard.