# Scripts de base sous linux

*Un **shell script** est un fichier texte qui contient des commandes.*

*Ces commandes sont exécutées séquentiellement en fonction des instructions de contrôle.*

*L'exécution du **shell script** démarre par la première ligne du fichier et se termine lorsque la fin du fichier est atteinte.*

## I. Lancement d'un script shell :

*Un **script shell** peut être lancé de plusieurs manières :*

### 1. En le faisant traiter explicitement par un shell :

*Si son contenu comporte des instructions spécifiques **Bash** Shell:*

**bash chemin/nom-shell-script**

*Si son contenu relève d'un autre shell par eemple Bourne Shell*

**sh chemin/nom-shell-script**

*Par exemple :*

**bash ../mes_scripts/tri.script**

### 2. En l'appelant au niveau de la ligne de commandes

*Dans ce cas le fichier devra être **exécutable**. Pour rendre le fichier exécutable, on peut par exemple. Tapez :*

**chmod a+x chemin/nom-shell-script**

*Deux cas de figures, peuvent se présenter :*

- *Si la variable **PATH** est configurée pour rechercher dans le répertoire qui contient le script. On peut seulement, tapez :*

  **nom-shell-script**

  *Par exemple :*

  **tri.script**

- *Si la variable PATH n'est pas configurée pour rechercher dans le répertoire qui contient le programme. On doit spécifier le chemin du script même si on est dans le répertoire du script.*

  **chemin/nom-shell-script**

  *Par exemple :*

  **../mes_scripts/tri.script**

  **./tri.script**

## II. Variables :

*N'ont pas besoin d'être déclarées préalablement à leur utilisation.*

*Ne permettent de manipuler que des chaînes de caractères*

*Ont des noms qui commencent par une lettre ou le caractère _*

### a. *Valorisation d'une variable:*

*Se fait par affectation ou par saisie :*

*Par exemple :*

**NOM=="ceci est un exemple"**
**read ORDRE**

### b. *Utilisation du contenu d'une variable :*

*Pour accéder au contenu d'une variable, il faut préfixer le nom de celle-ci par le symbole **$**.*

*Une variable peut être utilisée n'importe où dans une ligne de commande.*

*Par exemples :*

**CDE="ls" ; OPTIONS="-lia" ; ARG="/usr /tmp"**
**$CDE $OPTIONS $ARG**

## III. GUILLEMETS, QUOTES :

*Sur une ligne de commande, on peut encadrer des mots par **"**, **'** ou **`**. Mais cela a un effet sur l'interprétation, par le shell, des caractères spéciaux.*

*Lorsqu'une chaîne de caractères est encadrée par des **'** (quotes), tous les caractères spéciaux ($, *, ?, \ ...) qui sont à l'intérieur perdent leur signification.*

*Lorsqu'une chaîne de caractères est encadrée par des " (guillemets ou double-quote), seuls les caractères spéciaux $, \ et ` qui sont à l'intérieur conservent signification; tous les autres la perdent.*

## IV. SUBSTITUTION DE COMMANDE :

*Consiste à conserver dans une variable le texte produit par une commande. Cela n'est envisageable que vis à vis de commandes qui affichent en temps normal un résultat à l'écran (**filtres**, **ls**, **pwd**, **logname**, **dirname**, ...).*

*La syntaxe de la substitution :*

**nom-var=`texte-de-la-commande`**

*Par exemple :*

**LISTE=`ls –ila /tmp`**

*On peut aussi l'utiliser pour imbriquer une commande dans une autre de manière à passer la sortie de la 1$^{ière}$ commande comme argument de la 2$^{ième}$. Par exemple :*

**find `pwd` -size +300k -user `logname` -print**

## V. PARAMETRES D'APPEL ou ARGUMENTS :

*Durant son exécution, un programme **shell** est capable d'utiliser des informations qui lui ont été fournies sur la ligne de commande lors de son lancement.*

*Exemples :*

**myscript info1 info2 info3 ... infon**
**tri.sh `ls /etc`**

*Les instructions du programme accèdent à ces informations grâce à des variables réservées : **$1**, **$2**, **$3**, ..., **$9** (le Bourne shell ne va pas plus loin), **$10**, **$11**, ...*

*Exemple :*

```
case "$2" in
start) if [ "$1" = "$alpha" ]
       then pwd
       else cd "$1"
       fi
  ;;
stop) kill -KILL $PID
       ;;
  *) echo "erreur d'utilisation";
  echo "veuillez recommencer
  ;;
esac
```

*La liste complète des arguments d'un programme est rangée automatiquement dans la variable **$***

*Le nom du script en cours de traitement est conservé dans **$0***

*Le nombre de paramètres d'appel est connu grâce à la variable **$#***

*Le compte-rendu d'exécution d'un programme, de la dernière commande est stocké dans la variable **$?***

### 1. Réinitialisation de la liste des paramètres d'appel :

*Durant son exécution, un programme peut réactualiser les valeurs des paramètres d'appel. Ceci se fait grâce à la commande "**set**"*

```
set chaine1 [ chaine2 ... ]
```

*Exemples :*

```
set voici les nouveaux arguments
set $ALPHA TRI $BETA TDI
set `ls /etc`
```

*Cette commande met également à jour les valeurs de **$*** et $#*

### 2. Décalage des paramètres d'appel :

*Durant son exécution, un programme peut faire décaler les paramètres des dernières positions vers les premières. Ceci se fait grâce à la commande "**shift**"*

*Exemples :*

**shift** => pousse $2 dans $1, $3 dans $2, $4 dans $3, $5 dans $4, ...

**shift 3** => pousse $4 dans $1, $5 dans $2, $6 dans $3, ...

*Cette commande met également à jour les valeurs de **$*** et $#*

# VI. STRUCTURES DE CONTROLE :

## 1. La commande test :

*Cette commande permet de tester la nature d'un fichier, son existence, ses droits d'accès :*

**[ -condition chemin/fichier ]**

*La condition peut être l'une des valeurs suivantes :*

- ***-f :*** *est-ce un fichier ordinaire qui existe ?*
- ***-s*** *: est-ce un fichier non vide qui existe ?*
- ***-d*** *: est-ce un repertoire qui existe ?*
- ***-r*** *: est-ce un fichier accessible en lecture qui existe ?*
- ***-w*** *: est-ce un fichier accessible en écriture qui existe ?*
- ***-x*** *: est-ce fichier existant sur lequel le droit x est positionné ?*

*Exemples :*

**[ -f /var/tmp/poeme ]** ou bien **alpha=/var/tmp/poeme ; [ -f $alpha ]**
**[ -s ../action ]** ou bien **fic=../action ; [ -s $fic ]**
**[ -d /oracle ]** ou bien **REP=/oracle ; [ -d $REP ]**
**[ -r /var/tmp/poeme ]** ou bien **beta=/var/tmp/poeme ; [ -r $beta ]**
**[ -w /oracle/resul ]** ou bien **fic=/oracle/resul ; [ -w $fic ]**
**[ -x /oracle ]** ou bien **rep=/oracle ; [ -x $rep ]**

*La commande test permet également de formuler des conditions sur le contenu des variables*

**[ -condition $nom-variable ]**

*La condition peut être l'une des valeurs suivantes :*

- ***-z*** *: variable vide*
- ***-n*** *: variable non vide*

*Exemples*

**[ -z $A ]**
**[ -n $B ]**

*La commande test est utilisée pour comparer 2 variables l'une par rapport à l'autre, ou pour comparer une variable par rapport à une valeur constante. Sa syntaxe est :*

**[ $nom-variable1 -opérateur $nom-variable2 ]**
**[ constante -opérateur $nom-variable ]**
**[ $nom-variable -opérateur constante ]**

*Les opérateurs peuvent prendre les valeurs suivantes :*

- ***-eq** : égalité numérique*
- ***-ne** : difference numérique*
- ***-lt :** <*
- ***-gt** : >*
- ***-le** : <=*
- ***-ge** : >=*

*Exemples :*

```
[ $FICH = /var/tmp/poeme ]
[ $FICH != /oracle/resul ]
[ $# -eq 6 ]
[ 12 -ge $# ]
```

*Pour avoir la liste exhaustive des tests, tapez:* ***man test***

### 2. ***Instruction if :***

```
if commande-A
then
        liste-de-commandes-1
else
         liste-de-commandes-2
fi
#suite du shell script
```

*Si la commande A s'exécute avec succès, alors il faut exécuter la liste de commandes 1 et passer à la suite du shell script. Sinon il faut exécuter la liste de commandes 2 et poursuivre avec la suite du shell script.*

*Cela est équivalent à l'écriture suivante :*

```
Commande-A
if [ $? -eq 0 ]
then
        liste-de-commandes-1
else
        liste-de-commandes-2
fi
#suite du shell script
```

*La partie «* **else liste-de-commandes-2** *» est facultative.*

*On peut également écrire :*

```
if commande-A ; then liste-de-commandes-1 ; else liste-de-commandes-2 ; fi
#suite du shell script
```

*Exemples :*

```
if [ $# -eq 0 ]
then
   echo « Aucun argument reçu ! »
fi

echo « où aller ? »
read chemin
if cd $chemin 2 > /dev/null
then echo « déplacement vers nouvel endroit réussi ! »
       ls -l | more
       sleep 5
else echo « échec tentative de déplacement »
fi
```

### 3. ***Instruction while :***

```
while exécution-réussie-de-commande-A
do
       liste-de-commandes-1
done
#suite du shell script
```

*Tant que la commande A s'exécute avec succès, la liste de commandes 1 est traitée. Lorsqu'enfin la commande A échouera dans son exécution, la suite du shell script pourra être traitée.*

*Autre écriture possible :*

```
while exécution-réussie-de-commande-A ; do liste-de-commandes ; done
#suite du shell script
```

*Exemples :*

```
echo « votre choix ? »
read choix
while [ $choix != 'S' -a $choix != 's' ]
do
       echo « position dans l'arborescence »
       pwd
       sleep 5
```

```
        echo « contenu du répertoire »
        ls -l | more
        sleep 3
        echo « votre choix ? »
        read choix
done


echo « où aller ? »
read chemin
while  cd  $chemin  2 > /dev/null
then echo « déplacement vers nouvel endroit réussi ! »
       ls  -l  |  more
       sleep  5
       echo « où aller ? »
       read chemin
done
```

### 4. ***Instruction case :***

```
case  $nom-variable  in
        valeurA) liste-commandes-A ;;
        valeurB) liste-commandes-B ;;
……
        valeurN) liste-commandes-N ;;
        *) liste-commandes-autres-cas ;;
esac
#suite-shell-script
```

*Quand la variable contient la valeur A alors la liste de commandes A est exécutée et la suite du shell script se déroule.*

*Quand la variable contient la valeur B alors la liste de commandes B est exécutée et la suite du shell script se déroule.*

*Quand la variable ne contient aucune des valeurs proposées, alors la liste des commandes « autres cas » est exécutée avant de passer à la suite du shell script.*

*Exemples :*

```
case  $LOGNAME  in
  root ) PS1="# " ;;
  tux | ali )  PS1="Salam$LOGNAME$ "  ;;
  * ) PS1="\h:\w$" ;;
esac
export  PS1
```

```
case  $#  in
  0) echo "aucun parametre"
     echo "Syntaxe :  $0 <nom d'utilisateur>" ;;
  1) echo "1 paramêtre passé au programme : $1" ;;
  2) echo "2 paramêtres passés au programme : $1 et $2" ;;
  *) echo "TROP DE PARAMETRES !" ;;
esac
```

### 5. ***Instruction for :***

```
for  nom-variable  in  valeur1  valeur2  …  valeur N
do
        liste-de-commandes
done
#suite du shell script
```

*La variable mentionnée derrière le mot clé « for » prend tour à tour chacune des valeurs énumérées derrière le mot clé « in ». A chaque fois qu'elle change de valeur, il y a exécution de la liste de commandes. La suite du shell script est exécutée une fois que toutes les valeurs ont été prises.*

*Exemples :*

```
somme=0
for  i  in  1 2 3 4 5 6 7 8 9 10
do
   somme=`expr   $somme  +  $i`
done
echo "Somme des 10 premiers nombres : $somme"


for fichier in *.sh
do
   cat  $fichier
done
```

## 6. *UTILISATION DE FONCTIONS :*

```
Nom-fonction() {
Liste-de-commandes
}
```

*Une fonction doit avoir été définie (explicitée) avant de pouvoir être utilisée.*
*L'appel de la fonction se fait de la façon suivante :* **nom-fonction**
*Il est possible de lui passer des arguments :*
**nom-fonction argument1 argument2 … argumentN**

*Exemple :*

```
sshd_start() {
 if [ -r /etc/ssh/ssh_host_key ]
 then
   /usr/bin/ssh-keygen -t rsa
 fi
 if [ -f /etc/ssh/ssh_host_dsa_key ]
 then
   /usr/bin/ssh-keygen -t dsa
 fi
 /usr/sbin/sshd
}
if  [  $1 = start  ]
then
        sshd_start
fi
```