

# Chapitre 3: Les scripts Shell

Chérifa Boucetta



# Contenu

---

1. Les fichiers de démarrage du shell
2. Notion des variables variables
3. La programmation shell

# Les fichiers de démarrage du shell

---

- Il y a plusieurs types de fichiers de configuration:
  - ceux qui sont lus au moment de la connexion (*login*)
  - ceux qui sont lus à chaque lancement d'un shell.
- Les fichiers lus au moment de la connexion au système servent généralement à décrire l'environnement de l'utilisateur.
  - ***/etc/profile***, commun à tous les utilisateurs;
  - ***~/.bash\_profile*** ou éventuellement ***~/.bash\_login*** ou ***~/.profile***, spécifiques à chaque utilisateur.
- Les fichiers lus à chaque lancement du shell pour s'assurer que l'environnement correct sont :
  - ***/etc/bashrc*** commun à tous les utilisateurs;
  - ***~/.bashrc*** spécifique à chaque utilisateur.
  - ***~/.bash\_profile***

# Le fichier `/etc/profile`

---

- Il spécifie les paramètres de base:
  - exemple les paramètres d'historique de **bash**
  - Pour des raisons de sécurité, il désactive la conservation d'un fichier d'historique permanent pour l'utilisateur root.
- Il paramètre aussi une invite utilisateur par défaut.
- Il définit les Variables d'environnement globales et programmes de démarrage.
- Il appelle des scripts du répertoire ***/etc/profile.d*** pour fournir la plupart de l'initialisation.

# Les fichiers de démarrage du shell

---

- Le répertoire `/etc/profile.d`, contient les scripts d'initialisation individuels tels que:
  - **`/etc/profile.d/dircolors.sh`**: pour contrôler les couleurs des noms de fichiers dans la liste du contenu d'un répertoire.

Exemple:

```
user@ubnt:~$ ls --color
bash_logout  cache          profile        rep3
bashrc       named.conf.default-zones rep1          snap
builds      named.conf.options rep2          test
```

- **`~/.bash_profile`**
  - Ce fichier contient essentiellement la définition des variables d'environnement pour un utilisateur particulier.

# Le fichier bashrc

---

- Les fichiers de la famille « bashrc » sont lus chaque fois qu'un shell est lancé
- **Le fichier /etc/bashrc**
  - C'est un fichier de configuration permettant de personnaliser l'environnement Shell, définir des alias (des raccourcis pour certaines commandes), de créer des fonctions....
- **N.B:** toutes les nouvelles sessions de bash, puisqu'elles sont des processus fils, héritent des variables définies dans les fichiers « profile » lors de la connexion.
- **Le fichier ~/.bashrc**
  - permet pour un utilisateur particulier, typiquement :
    - La définition des alias personnels;
    - La définition des paramètres de fonctionnement et des fonctions ;
    - L'initialisation des variables.

# Les fichiers de démarrage du shell

---

- **/etc/bash\_logout**

- s'il est présent, il est exécuté à la fin de la session bash de tous les utilisateurs.

- **~/bash\_logout**

- s'il est présent, il est exécuté à la fin de la session bash spécifique à un utilisateur.
  - Il est lu par le shell quand un utilisateur se déconnecte du système.

- **/etc/profile.d/umask.sh**

- Ce script assure le paramétrage de la valeur **umask** qui est important pour la sécurité

# Contenu

---

1. Les fichiers de démarrage du shell
2. **Notion des variables variables**
3. La programmation shell



# Les variables

---

- Trois types : **utilisateur**, **système** et **spéciales**.
- Le système UNIX définit pour chaque processus une liste de variables d'environnement, qui permettent de définir certains paramètres.
- La commande **env** permet d'afficher l'ensemble des variables d'environnement pour le shell actif

# Les variables

---

- Parmi ces variables :
  - **HOME** : contient le chemin absolu du répertoire de connexion de l'utilisateur
  - **LOGNAME** : contient le nom de connexion de l'utilisateur
  - **PATH** : contient la liste des répertoires contenant des exécutable séparés par ' : '
    - Ces répertoires seront parcourus par ordre à la recherche d'une commande externe
  - **SHELL** : contient le chemin d'accès absolu des fichiers programmes du shell
  - **PWD**: Répertoire Courant

# Les variables

---

- Exemples:

```
user@ubnt:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
user@ubnt:~$
```

```
user@ubnt:~/rep1$ echo $PWD
/home/user/rep1
user@ubnt:~/rep1$ echo $SHELL
/bin/bash
```

```
user@ubnt:~$ echo $HOME
/home/user
```

# Les variables

---

- La commande **set** sans paramètre permet d'afficher la liste des variables définies dans le shell.
- L'affectation d'une variable s'effectue simplement avec l'opérateur égal sous la forme **nom=contenu** sans espace de part et d'autre du caractère égal.
  - Le contenu d'une variable peut être un entier, une chaîne de caractères, le contenu d'une variable, le résultat d'une commande ou toute combinaison de ces éléments.
- Pour supprimer une variable déjà déclarée, on utilise la commande **unset** dont la syntaxe est la suivante :

**unset nom\_variable**

# Les variables

---

- **Exemple:**

```
$ a=salle
```

```
$ b=informatique
```

```
$ echo $a
```

```
salle
```

```
$ set
```

```
HOME=/home/user
```

```
....
```

```
LOGNAME=user
```

```
PATH=/usr/bin/:/home/toto/bin:.
```

```
SHELL=/bin/bash
```

```
a=salle
```

```
b=informatique
```

# Les variables

---

- Pour qu'une variable soit visible de tous les shells (donc de toutes les commandes lancées depuis le shell courant), il faut l'exporter par la commande **export**.
  - **export MANPATH="/usr/share/docs"**
- Lors du démarrage d'une session shell, la plupart des variables d'environnement sont initialisées et exportées à partir des différents fichiers de configuration, tels que les fichiers `bashrc` et `profile`.
- La commande **env** permet de démarrer un shell avec des variables d'environnement déjà positionnées à des valeurs données.
- Ces variables d'environnement ont une durée de vie égale à celle du shell démarré.

# Les variables spéciales

---

- **Les variables spéciales:**

- **\$?** : valeur de retour de la dernière commande exécutée
  - **\$\$** : numéro du processus (PID) du shell actif
  - **#!** : numéro du processus (PID) du dernier processus lancé en arrière plan
  - **\$0** : nom de procédure de commande
  - **\$1....\$9** : valeur de n<sup>ième</sup> paramètre
  - **\$#** : nombre de paramètre transmis à l'appel
  - **\$\*** : liste de tous les paramètres \$1.....\$9
- 
- L'opérateur **shift** décale les paramètres.

# Les variables: Exemple 1

---

- **Afficher la liste des variables:**

```
user@ubnt:~$ vim test_var
```

```
for var in "$@"
```

```
do
```

```
    echo $var;
```

```
done
```

- **Résultat d'exécution:**

```
user@ubnt:~/rep1$ ./test_var 1 2 3 4 5
1 2 3 4 5
user@ubnt:~/rep1$ ./test_var arg1 arg2 arg3
arg1 arg2 arg3
user@ubnt:~/rep1$ ./test_var arg1 arg2 arg3 4 78 TEST
arg1 arg2 arg3 4 78 TEST
user@ubnt:~/rep1$ █
```



# Les variables: Exemple 2

- **Afficher le nombre de variables**

```
user@ubnt:~$ vim variable
```

```
echo "Vous avez lancé $0, il y a $# paramètres"
```

```
echo "Le paramètre 1 est $1"
```

- **Résultat d'exécution:**

```
user@ubnt:~/rep1$ ./variable
Vous avez lancé ./variable, il y a 0 paramètres
Le paramètre 1 est
user@ubnt:~/rep1$ ./variable 2 4 Tata
Vous avez lancé ./variable, il y a 3 paramètres
Le paramètre 1 est 2
```

# Contenu

---

1. Les fichiers de démarrage du shell
2. Notion des variables variables
3. **La programmation shell**

# Automatisation des tâches - scripts

---

- Utilisation interactive ou "**script**"
  - fichier texte contenant une suite de commandes
    - forme de "programme exécutable "
  - Utilise le langage de programmation de l'interpréteur
  - exécutable par le shell :
    - En saisissant son nom dans une console ;
    - Via GUI (double clic sur le fichier). (sous réserve qu'il soit accessible depuis le dossier courant...)

# Introduction aux scripts

---

- Rappels :
  - **Algorithme** → suite de règles pour produire un résultat
  - **Programme** → réalisation de l'algorithme par une suite d'instructions adressées à une machine
  - **Processus** → exécution du programme

code	Interprétation
<code>#!/bin/bash</code>	<i>Shebang</i>
<code># un titre : un essai</code>	<i>Commentaire</i>
<code>echo -n "Bonjour"</code>	<i>Une instruction par ligne</i>
<code>whoami</code>	

# Les scripts shell

- Un script shell est une liste d'instructions contenues dans un fichier.

```
#!/bin/bash  
# Un petit script mon_script  
echo 'Il fait beau'
```

- Pour pouvoir exécuter ces instructions, deux conditions doivent être remplies :
  - La première ligne doit contenir **#!/bin/bash** (pour un shell script utilisant bash) ;
  - Le fichier doit être exécutable et lisible
    - `chmod u+x mon_script.sh`
    - `./mon_script.sh`
- Si toutes ces conditions ne sont pas remplies, il est toujours possible de forcer l'exécution du script avec la commande **bash**: `user@ubnt:~$ bash mon_script.sh`

# Les scripts shell

- L'utilisateur peut créer des fichiers contenant des lignes de commandes qui seront interprétées par un shell, des variables, des structures de contrôles, des structures répétitives...etc.
- La commande **read** lit la saisie de l'utilisateur à partir du canal d'entrée standard et stocke ces données dans des variables du shell

## \$cat testread.sh

```
#!/bin/bash
echo "saisir le contenu des
variables a et b "
read a b
echo " a = $a"
echo " b = $b "
```



## \$testread.sh

```
saisir le contenu des variables a et b
15
salut
a = 15
b = salut
```

# Les scripts shell

- Remarques: Il existe trois types de quotes :

- **Les simples quotes** ' '

- Avec de simples quotes la variable n'est pas analysée et son contenu est affiché tel quel

```
user@ubuntu-srv:~$ b='Master 2021'
user@ubuntu-srv:~$ echo 'le contenu de b est: $b'
le contenu de b est: $b
```

- **Les doubles quotes** " "

- les doubles quotes demandent à bash d'analyser le contenu du message. S'il trouve des symboles spéciaux (comme des variables), il les interprète.

```
user@ubuntu-srv:~$ a="Matster 2021"
user@ubuntu-srv:~$ echo "le contenu de a est: $a"
le contenu de a est: Matster 2021
```

- **Les back quotes** ` `

- les back quotes demandent à bash d'**exécuter** ce qui se trouve à l'intérieur.
  - La commande pwd a été exécutée et son contenu inséré dans la variable message !

```
user@ubuntu-srv:~$ d=`date`
user@ubuntu-srv:~$ echo "nous sommes le $d"
nous sommes le jeu. 07 oct. 2021 16:50:54 UTC
```

# Le test If

---

- **Le test if :**

```
if condition1
then
liste_commande1
elif condition2
then
liste_commande2
else
liste_commande3
fi
```

- **Remarque:** En shell, la valeur zéro est associée à la valeur booléenne vraie et toute autre valeur correspond à la valeur faux.



# La commande test

---

- **la commande test:** effectuer des tests de comparaison, en retournant un code de sortie égal à zéro lorsque la comparaison est vraie, égal à 1 si faux et une autre valeur sinon.
  - La syntaxe est la suivante : **test [expression]**
- **Options courantes :**
  - **ch1 = ch2** : chaînes de caractères ch1 et ch2 identiques (il faut qu'il y ait espace)
  - **ch1 != ch2** : chaînes de caractères ch1 et ch2 différentes
  - **nb1 -eq nb2** : nombres nb1 et nb2 égaux
  - **nb1 -ne nb2** : nombres nb1 et nb2 différents
  - **nb1 -gt nb2** :  $nb1 > nb2$
  - **nb1 -ge nb2** :  $nb1 \geq nb2$
  - **nb1 -lt nb2** :  $nb1 < nb2$
  - **nb1 -le nb2** :  $nb1 \leq nb2$

# Conditions

---

- Tester les fichiers

- `-d fichier` → fichier existe et est un répertoire
- `-e fichier` → fichier existe
- `-f fichier` → fichier est régulier (ni répertoire ni fichier spécial)
- `-r fichier` → vous avez la permission de lire fichier
- `-w fichier` → vous avez la permission d'écrire dans fichier
- `-x fichier` → vous avez la permission d'exécuter fichier
- `-s fichier` → fichier existe et n'est pas vide
- `fichier1 -nt fichier2` → fichier1 est plus récent que fichier2 (*date de la dernière modification*)

# Exemple test

- **Exemple: Vérifier le type du paramètre passé en argument:**
- user@ubnt:~\$ cat script.sh

```
#!/bin/bash
if [ -d $1 ] # les espaces sont obligatoires après [ et avant ]
    then echo " $1 est un répertoire"
elif [ -h $1 ]
    then echo "$1 est un lien symbolique"
elif [ -f $1 ]
    then echo "$1 est un fichier ordinaire "
else echo "tester une autre option de test"
fi
```

## Exécution:

```
user@ubuntu-srv:~$ ./script1.sh test
test est un fichier ordianire
user@ubuntu-srv:~$ ./script1.sh Documents/
Documents/ est un répertoire
```

# Le test If: exemple

---

- `$cat affich_rep # Afficher le contenu du répertoire donné en #paramètre. S'il n y a pas de paramètre, un message est affiché`

```
#!/bin/bash
if test $# -eq 1 #on peut écrire aussi if test [ $# -eq 1 ]
then
    if test -d $1
    then
        echo "le contenu de $1 est `ls -R $1` "
    else
        echo "$1 n'est pas un repertoire"
    fi
else
    echo "syntaxe: $0 repertoire"
fi
```

# La boucle For

---

- Pour la boucle **for**, il ne s'agit pas de fixer une valeur de départ et une valeur de fin contrôlant le nombre d'itérations mais d'une répétition d'un traitement pour des valeurs différentes d'une variable.
- La syntaxe est la suivante :

```
for var in valeur1 valeur2 ... valeur n
do
    liste_commandes
done
```

- La liste de commandes sera exécutée autant de fois que ce contient la liste en nombre d'éléments.

# Les scripts shell

## Exemple 1: \$cat for.sh

```
#!/bin/bash
mot1=bonjour
mot2=hello
for var in $mot1 $mot2 $1 salut
do
    echo $var
done
```

Exécution:

```
user@ubnt:~$for.sh bonsoir
Bonjour
Hello
Bonsoir
Salut
```

# Les scripts shell

---

## Exemple 2: \$ cat concatenation.sh

#concatenation de tous les fichiers passés en argument dans le premier fichier

```
#!/bin/bash
f=$1
shift
for i in $*
do
    echo $i >> $f
    cat $i >> $f
done
```

# La boucle while

---

- Cette structure permet de boucler sur une séquence de commandes tant que la condition est vraie. La boucle est interrompue si la valeur de retour est différente de zéro.

La syntaxe est la suivante :

**while** condition

**do**

    liste\_commandes

**done**

- On peut sortir de la boucle avec la commande **break**.



# Les scripts shell

---

- Exemple: `$cat testwhile`

```
#!/bin/bash
while [ $1 != " fin " ];do
echo $1
shift
Done
```

L'exécution donne :

```
user@ubnt:~$ testwhile 1 2 3 fin 4 5 6
1
2
3
```

# Le calcul

---

- Tester les nombres entiers

- `$a -eq $b` → **\$a** égal **\$b**
- `$a -ne $b` → **\$a** différent de **\$b**
- `$a -lt $b` → **\$a** inférieur strictement à **\$b**
- `$a -gt $b` → **\$a** supérieur strictement à **\$b**
- `$a -le $b` → **\$a** inférieur ou égal à **\$b**
- `$a -ge $b` → **\$a** supérieur ou égal à **\$b**
- `! condition` → **condition** est fausse (« ! » signifie **non**)

# La commande **expr**

---

- La commande **expr** permet d'effectuer les quatre opérations arithmétiques de base, avec les opérateurs suivants :
  - **+** pour l'addition ;
  - **-** pour la soustraction ;
  - **\\*** pour la multiplication ;
  - **/** pour la division.

# Exemple 1

---

- Exemple: afficher la table de multiplication d'un chiffre passé en paramètre

```
#!/bin/bash
for i in 1 2 3 4 5 6 7 8 9 ;do
expr $i \* $1
done
```

# Exemple 2

---

- **Exemple: Compter jusqu'à 100**

- On peut également écrire `expr <expression>` sous la forme `$((<expression>))`

```
#!/bin/bash
i=0
while [ $i -ne 100 ] ;do
i=`expr $i + 1` #ou i=$(( $i + 1 ))
echo $i
done
```

# Exercice

Ecrire le script *MyCps* qui permet de copier la sous arborescence d'un répertoire dans un autre. Ce script admet deux paramètres : le premier est le répertoire qui contient la sous arborescence à copier et la deuxième dans laquelle on veut faire la copie

# Exercice2

---

- Ecrire le script shell qui permet de renommer tous les fichiers du répertoire passé en argument et d'extension .cpp en .cpp.old

# Exercice2

---

- **Ecrire le script shell qui permet de renommer tous les fichiers du répertoire passé en argument et d'extension .py en .py.old**

```
#!/bin/bash
if [ $# != 1 ]
then
    echo "syntaxe: $0 repertoire"
else
    if test -d $1
    then
        for x in $1/*.cpp ; do
            if test -e $x
            then
                echo "$x -> $x.old"
                mv "$x" "$x.old"
            fi
        done
    else
        echo "$1 doit etre un repertoire"
    fi
fi
```



# La boucle pour

---

- Une autre alternative pour la boucle **for**:

**for(( **EXPR1**; **EXPR2**; **EXPR3**)); do**

**statements**

**done**

- EXPR1: initialisation; EXPR2: loop-test ou condition; EXPR3: indice d'incrément

```
#!/bin/bash
echo "Enter a number: "; read x
let sum=0
for (( i=1 ; $i<=$x ; i=$i+1 )) ; do
let "sum = $sum + $i"
done
echo "La somme des $X premiers chiffres est : $sum"
```

# Les tableaux

---

- Tableau de variables: création de tableau var

- Syntaxe:

**varname=(list of words)**

- Exemples:

```
var=('valeur1' 'valeur2' 'valeur3')  
# ou  
var('valeur1' 'valeur2' 'valeur3')  
# ou  
var=[0]'valeur1' [1]'valeur2' [2]'valeur3')
```

# Les tableaux

---

```
# Affiche toutes les entrées du tableau
${var[@]}
valeur1 valeur2 valeur3
# Affiche toutes les entrées du tableau aussi
${var[*]}
valeur1 valeur2 valeur3
# Affiche la valeur de l'indice 0 (premier)
${var[0]}
valeur1
# Affiche le nombre d'indices
${#var[@]}
3
#Affiche tous les indices
${!var[@]}
0 1 2
```

# Les fonctions

---

- Une fonction est un bloc d'instructions que l'on peut appeler ailleurs dans le script.
- Pour déclarer une fonction, on utilise la syntaxe suivante :

```
maFonction()  
{  
  hello world  
}
```

- La déclaration d'une fonction doit toujours se situer avant son appel. On mettra donc les fonctions en début de script.

# Les fonctions:

---

- La fonction est appelée
  - **fonction [paramètres]**
  - Les paramètres d'appel dans les variables \$1 à \$9 et \$#.
  - La valeur de \$0 reste inchangée (nom du script)
- Déclarer des variables locales
  - **local** avant la déclaration de la variable
  - modification de la variable locale ne modifie pas une variable globale portant le même nom

# Les fonctions: Exemple 1

---

```
#!/bin/bash
read -p "donner un entier ?" n

multiplication() {
for i in 1 2 3 4 5 6 7 8 9
do

    echo "$i*$n= `expr $i \* $n`"

done

}

multiplication

exit
```

# Les fonctions: Exemple 2

---

- Variable locale/variable globale

```
MaFonction()  
{  
  local maVariable="Hello World"  
  echo $maVariable  
}  
maVariable="Bonjour"  
echo $maVariable  
MaFonction  
echo $maVariable
```



```
user@NS1-at-rtlab:~/scripts$ ./fonction2.sh  
Bonjour  
Hello World  
Bonjour
```

# Retour d'une fonction

- Exit status

- Si la fonction ne contient pas return
  - exit status= résultat de l'exécution de la dernière commande
- sinon
  - return valeur ou exitnnn(nnn=0->255)

- Différence

- Return: terminer l'exécution d'une fonction
- Exit: terminer l'exécution du script shell

- Exemple:

```
#!/bin/bash
retfunc() {
echo "this is retfunc()"
return 1
}
exitfunc() {
echo "this is exitfunc()"
exit 1
}
retfunc
echo "We are still here"
exitfunc
echo "We will never see this"
```

```
$ ./test.sh
this is retfunc()
We are still here
this is exitfunc()
```



# Les fonctions: Exemple 3

---

```
#!/bin/bash
dir=/home/user/scripts
backupdir=~ /backup/

backup_one_file()
{
cp $1 $backupdir
echo $1 has been backed up
}

for file in $dir/*
do
[ -s $file ] && backup_one_file $file
done
```

# Fin du Chapitre 3