



# UNIX

## Cours 8 : Programmation SHELL (bash) (Partie 2)

Filipe Vasconcelos<sup>1</sup>

<sup>1</sup>ESME, Lille, [filipe.vasconcelo@esme.fr](mailto:filipe.vasconcelo@esme.fr)

- ① État de sortie (exit status)
- ② Test
- ③ Structures conditionnelles
- ④ Structures itératives



## AAV4

Produire, tester et vérifier le résultat des scripts SHELL (en bash) pour réaliser des tâches algorithmiques simples à complexes



① État de sortie (exit status)

② Test

③ Structures conditionnelles

④ Structures itératives



# État de sortie (ou code de sortie)

La commande exit est utilisée pour terminer un script, c'est l'équivalent du return dans un programme en C. Un script ou un programme renvoie toujours un entier entre 0 et 255. Un script ou un programme ayant réussi sans erreur renvoie 0. L'état de sortie de la dernière commande exécutée est donné par \$?

Exemple (état de sortie 0) :

```
$ ls  
$ echo $?
```

```
0
```

Exemple (état de sortie « commande introuvable ») :

```
$ lss  
$ echo $?
```

# L'instruction return en C

```
$ ./return_en_C.sh
```

```
-----  
code :  
-----  
  
#include <stdio.h>  
int main(){  
    return 0;  
}  
-----
```

```
gcc returnC0.c (compilation)  
.a.out          (exécution)  
return $? = 0   ($? de l'exécution)
```

```
-----  
code :  
-----  
  
#include <stdio.h>  
int main(){  
    return 1;  
}  
-----
```

```
gcc returnC1.c (compilation)  
.a.out          (exécution)  
return $? = 1   ($? de l'exécution)
```

- NB : Le langage C est hors programme, mais le lien entre ce langage est les OS Unix, Linux ou l'interpréteur bash est très important !

① État de sortie (exit status)

② Test

③ Structures conditionnelles

④ Structures itératives



En bash il existe deux commandes équivalentes pour tester une condition : [] et test. (c.f `man [`). La valeur de l'état de retour (exit status \$?) indique si la condition est vraie 0 ou fausse 1. Commande [

```
[ 1 = 0 ]; echo $?
```

```
1
```

Commande test

```
test 0 = 0; echo $?
```

```
0
```



① État de sortie (exit status)

② Test

**③ Structures conditionnelles**

④ Structures itératives



Les structures conditionnelles sont des tests sur l'état de la sortie de la commande [] ou (test)

## if-then-fi

Structure si-alors

Exemple :

```
if [ condition ]
then
    echo "la condition est vraie"
fi
```

- le mot clé fi termine l'instruction
- pas d'espaces avant et après la condition
- La syntaxe if [ condition ];then est permise (c.à-d. sur une même ligne)



### if-then-else-fi

Structure si-alors-sinon

Exemple :

```
if [ condition ]
then
    echo "la condition est vraie"
else
    echo "la condition est fausse"
fi
```



### if-then-elif-then-else-fi

Structure si-alors-si-alors-sinon

Exemple :

```
if [ condition1 ]
then
    echo "la condition1 est vraie"
elif [ condition2 ]
then
    echo "la condition2 est vraie"
else
    echo "toutes les conditions sont fausses"
fi
```

# Syntaxe de la condition

## Différents types de conditions

- sur des chaînes de caractères ;
- sur des nombres ;
- sur des fichiers ;
- sur l'exécution d'une commande.

## Opérateurs booléens

- ET : && ou -a
- OU : || ou -o
- NON : ! NON



# Condition sur des chaînes de caractères

En bash toutes les variables sont considérées comme des chaînes de caractères. Il est donc très facile de tester ce que vaut une chaîne de caractères.

Condition	Signification	Syntaxe
\$chaine1 = \$chaine2	Vérifie si les deux chaînes sont identiques.	[ "\$var" = "lundi" ]
\$chaine1 == \$chaine2	Vérifie si les deux chaînes sont identiques.	[ "\$var" == "lundi" ]
\$chaine1 != \$chaine2	Vérifie si les deux chaînes sont différentes.	[ "\$var" != "lundi" ]
-z \$chaine	Vérifie si la chaîne est vide.	[ -z "\$var" ]
-n \$chaine	Vérifie si la chaîne est non vide.	[ -n "\$var" ]

Exemple :

```
var="jeudi"
if [ "$var" == "lundi" ]
then
    echo "Nous sommes lundi"
else
    echo "Nous ne sommes pas lundi"
fi
```

# Condition sur des nombres

Condition	Signification	Syntaxe
\$num1 -eq \$num2	Vérifie si les nombres sont égaux (equal). À ne pas confondre avec le « = » qui, lui, compare deux chaînes de caractères.	[ "\$num" -eq 5 ]
\$num1 -ne \$num2	Vérifie si les nombres sont différents (nonequal). Ne pas confondre avec « != »	[ "\$num" -ne 5 ]
\$num1 -lt \$num2	Vérifie si num1 est inférieur à num2 (lowerthan).	[ "\$num" -lt 5 ]
\$num1 -gt \$num2	Vérifie si num1 est supérieur à num2 (greaterthan).	[ "\$num" -gt 5 ]
\$num1 -le \$num2	Vérifie si num1 est inférieur ou égal à num2 (lowerorequal).	[ "\$num" -le 5 ]
\$num1 -ge \$num2	Vérifie si num1 est supérieur ou égal à num2 (greaterorequal).	[ "\$num" -ge 5 ]

Exemple :

```
num=5
if [ "$num" -eq 5 ]
then
    echo "le nombre est égal à 5"
else
    echo "le nombre est différent de 5"
fi
```

# Condition sur des fichiers

Condition	Signification	Syntaxe
-e nomFichier	Vérifie si le fichier existe.	[ -e "\$file" ]
-d nomFichier	Vérifie si le fichier est un "répertoire".	[ -d "\$file" ]
-f nomFichier	Vérifie si le fichier est un fichier (pas un repertoire).	[ -f "\$file" ]
-L nomFichier	Vérifie si le fichier est un lien symbolique (raccourci).	[ -L "\$file" ]
-r nomFichier	Vérifie si le fichier est lisible (r).	[ -r "\$file" ]
-w nomFichier	Vérifie si le fichier est modifiable (w).	[ -w "\$file" ]
-x nomFichier	Vérifie si le fichier est exécutable (x).	[ -x "\$file" ]
\$fichier1 -nt \$fichier2	Vérifie si fichier1est plus récent que fichier2 (newerthan).	[ "\$file1" -nt "\$file2" ]
\$fichier1 -ot \$fichier2	Vérifie si fichier1est plus vieux que fichier2(olderthan).	[ "\$file1" -ot "\$file2" ]

Exemple :

```
file="test.txt"
if [ -e $file ]
then
    echo "le fichier $file existe"
else
    echo "le fichier $file n'existe pas"
fi
```

# Utilisation des booléens

Condition	Signification	Syntaxe
<code>!condition</code>	vrai si condition est fausse	<code>[! ("\$var"="abc")]</code>
<code>cond1 &amp;&amp; cond2</code>	vrai si cond1 ET cond2 sont vraies	<code>[-e \$f] &amp;&amp; [-w \$f]</code>
<code>cond1    cond2</code>	vrai si cond1 OU cond2 est vrai	<code>[\$n -ge 2]    [ \$n -le 4 ]</code>

Exemple :

```
if [ -e $file ] && [ -w $file ]
then
    echo "le fichier $file existe et est modifiable"
fi
if [ $num -ge 2 ] || [ $num -le 6 ]
then
    echo " 2 <= n <=6"
fi
```



# Condition sur le résultat d'une commande

- ❑ Pour tester une commande on utilisera la variable \$?.
- ❑ Si la commande réussit (\$? est égal à 0) alors la condition est remplie, l'action correspondante est exécutée.
- ❑ Toutes les autres valeurs de retour sont considérées comme équivalente à false (erreur d'exécution).

Exemple :

```
grep $USER /etc/passwd
if [ $? -eq 0 ]
then
    echo "Vous êtes mentionné dans /etc/passwd"
else
    echo "Vous n'êtes pas mentionné dans /etc/passwd"
fi
```

Ici, grep retourne 0 si le motif est trouvé dans le fichier, sinon il retourne 1.

- ① État de sortie (exit status)
- ② Test
- ③ Structures conditionnelles
- ④ Structures itératives**



Les structures itératives permettent de répéter autant de fois que nécessaire une partie du code (boucle). Il y a trois façons d'implémenter des boucles

- for
- while
- until



Dans une boucle **for**, des instructions sont exécutés n fois .

## « pour-faire »

```
for var in "valeur1" "valeur2" "valeur2"  
do  
    echo "la variable vaut $var"  
done
```

Définir une liste à partir du résultat d'une commande :

```
for file in $(ls)  
do  
    echo "fichier $file trouvé"  
    <instruction sur file>  
done
```



en utilisant la commande seq

```
for i in $(seq 1 10)
do
    echo ${i}
done
```

Pour modifier le pas (c.f man seq) Exemple (de 2 en 2) :

```
for i in $(seq 1 2 10)
do
    echo ${i}
done
```



Une syntaxe proche C peut être utilisée :

```
for ((i=1;i<=5;i++))  
do  
    echo ${i}  
done
```



# Boucle while

Dans la boucle **while**, des instructions sont exécutées tant que la condition est vraie.  
Exemple : Boucle infinie

```
while true
do
    echo "je suis une boucle infinie"
done
```

Utilisation classique : lecture de fichier ligne par ligne

```
n=1
while read line
do
    echo "$n : $line"
    n=$((n+1))
done < fichier.txt
```



Exemple : Intéragir avec l'utilisateur

```
while [ "$choix" != "q" ]
do
    echo "Tapez q pour quitter la boucle"
    read choix
done
```



## Boucle until

Dans la boucle **until**, des instructions sont exécutées jusqu'à ce qu'une condition soit vraie.  
Exemple :

```
until false
do
    echo "je suis une boucle infinie"
done
```

