

Linux shell  
bash (partie 2)

Formation continue  
Collège de Maisonneuve

2030, boul. Pie IX, bureau 430

Montréal (Québec) H1V 2C8

Rédacteurs  
Hakim Benameurlaine – Juin 2016  
Louis Savard – Février 2018

Table des matières

[Structures de contrôle 3](#_Toc508433616)

[Structures conditionnelles 3](#_Toc508433617)

[Conditionnelles imbriquées 5](#_Toc508433618)

[Choix multiples (case) 8](#_Toc508433619)

[structures itératives 10](#_Toc508433620)

[boucle for 10](#_Toc508433621)

[Boucle While 12](#_Toc508433622)

[Sortie et reprise de boucle 15](#_Toc508433623)

[Fonctions 16](#_Toc508433624)

[Calcul sur les entiers 17](#_Toc508433625)

## Structures de contrôle

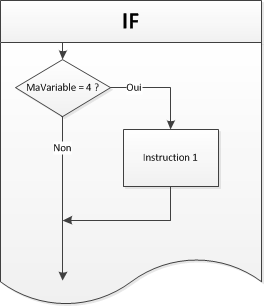
### Structures conditionnelles

La prise de décision est un élément fondamental de tout programme.

Les branchements conditionnels ou conditions sont un outil permettant d’indiquer à un script …

sI une condition est vrai, ALORS fais ceci, SINON fais cela.

Voici comment cela se présente …



**Déclaration if..then..fi**

**if [ expression conditionelle ]**

**then**

**instruction 1**

**instruction 2**

**…**

**fi**

**Exemple**

**#!/bin/bash**

**compte**=**100**

**if** [ **$compte** -eq **100** ]

**then**

**echo** "Le compte est 100"

**fi**

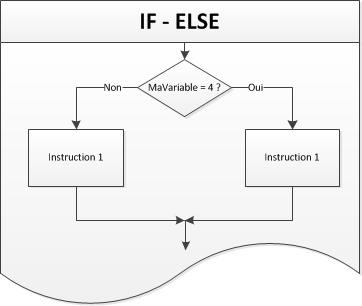
Cette instruction **if** est aussi appelée **simple if**.

* Si l'expression conditionnelle donnée est vraie,   
  elle entre et exécute les instructions contenues entre les mots clés then et fi.
* Si l'expression donnée renvoie zéro,   
  la liste d'instructions correspondante est exécutée.

Attention …

Si **then** est placé sur la 1ère ligne, il faut le séparer avec un ;

**if [ expression conditionelle ]; then instruction 1; fi**



**Déclaration if..then..else..fi**

**if [ expression conditionelle ]**

**then**

**instruction 1**

**instruction 2**

**...**

**else**

**instruction 3**

**instruction 4**

**...**

**fi**

**Remarques** …

* Si l'expression conditionnelle est vraie,   
  elle exécute les instructions 1 et 2.
* Si l'expression conditionnelle renvoie zéro,   
  elle saute à part et exécute les instructions 3 et 4.
* Après l'exécution de la partie if / else,   
  l'exécution reprend avec les instructions suivantes.

**Exemple**

**#!/bin/bash**

count=**99**

**if** [ **$compte** -eq **100** ]

**then**

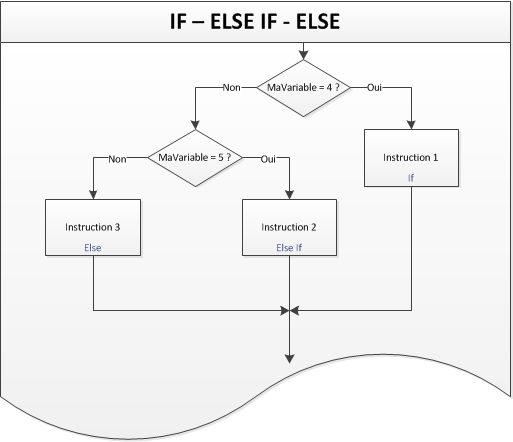
**echo** "Le compte est 100"

**else**

**echo** "Le compte est différent de 100"

**fi**

### Conditionnelles imbriquées



**Déclaration If..elif..else..fi**

**If [ expression conditionnelle 1 ]**

**then**

**instruction 1**

**instruction 2**

**…**

**elif [ expression conditionnelle 2 ]**

**then**

**instruction 3**

**instruction 4**

**…**

**else**

**déclaration 5**

**fi**

On utilise if ... elif .. if, si on veut sélectionner l'un des nombreux blocs de code à exécuter.

* Cette instruction vérifie l'expression 1,   
  si elle est vraie exécute l'instruction 1,2.
* Si expression1 est fausse,   
  elle vérifie expression2,
* et si les deux expression sont fausses,   
  alors elle entre dans le bloc else et exécute les instructions dans le bloc else.

**Exemple**

#!/bin/bash

compte=**99**

**if** [ **$compte** -eq **100** ] ; **then**

**echo** "Le compte est 100"

**elif** **[ $compte** -gt **100** ] ; **then**

**echo** "Le compte est plus grand que 100"

**echo** " Le compte est plus petit que 100"

**fi**

Déclaration If..then..else..if..then..fi..fi..

**if [ expression conditionnelle 1 ]**

**then**

**instruction 1**

**instruction 2**

**…**

**else**

**if [ expression conditionnelle 2 ]**

**then**

**instruction 3**

**…**

**fi**

**fi**

Les déclarations if et else peuvent être imbriqués.

Le mot-clé fi indique la fin de l'instruction if interne.

Toute instruction if doit se terminer par le mot-clé fi.

**Exemple**

**#!/bin/bash**

count=**99**

**if** [ **$compte** -eq **100** ]

**then**

**echo** "Le compte est 100"

**else**

**if** **[ $compte** -gt **100** ]

**then**

**echo** "Le compte est plus grand que 100"

else

**echo** " Le compte est plus petit que 100"

**fi**

**fi**

**Exemples plus complexes**

Test de la présence d'une ligne commençant par polo dans **/etc/passwd** …

**#!/bin/bash**

grep "^polo" /etc/passwd > /dev/null

if [ $? –eq 0 ] ; then

echo "polo a déjà un compte"

fi

Si root a eu une bonne note, on le félicite …

**#!/bin/bash**

intra=84

if [ $intra -gt 80 ] ---> *test vrai, valeur retournée : 0 ;* then

echo "Très bien !"

fi

Avant d'exécuter un script, tester son existence. (extrait de $HOME/.bash\_profile) **…**

**#!/bin/bash**

if [ -f ~/.bashrc ] ; then

. ~/.bashrc

fi

Trouver le maximum de trois nombres a b c …

**#!/bin/bash**

a=$1;b=$2;c=$3

if [ $a -gt $b ]

then

if [ $a -gt $c ]

then

echo "1maximum=$a"

else

echo "2maximum=$c"

fi

elif [ $b -gt $c ]

then

echo "3maximum=$b"

else

echo "4maximum=$c"

fi

Supposons que le script exige la présence d'au moins un paramètre

**#! /bin/bash**

if [ $# -eq 0 ]

then

echo "paramètre absent"

elif [ $# -eq 1 ]

then

echo "donner le second paramètre:"

read parametre2

fi

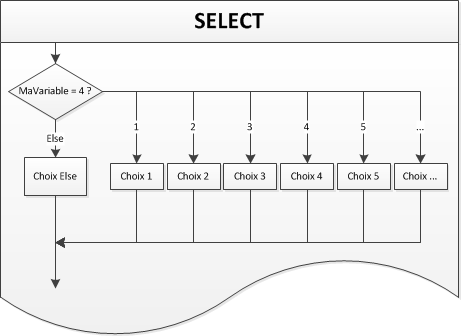
### Choix multiples (case)

La construction d’un script avec case permet de comparer des chaînes à des modèles qui peuvent contenir des caractères génériques.

Avec l’instruction case, bash vérifie une condition et contrôle avec le résultat de la verification le flux du script.

L'instruction case est une forme plus simple que l'instruction if..then..else.

L'instruction case n'est pas une boucle car elle n'exécute pas un bloc de code pour *n* nombre de fois.



case expression in

motif1)

instruction(s) ;;

motif2)

instruction(s) ;;

...

esac

**Exemples**

Ce script doit réagir différemment selon la valeur de **$USER**

**#! /bin/bash**

case $USER in

root) echo "mes respects M le $USER";;

usager1|usager2) echo "Bonjour $USER";;

usager3) echo "Réveille-toi $USER";;

esac

Le script attend une réponse *oui/non Yes/No* de l'utilisateur.

**#! /bin/bash**

read reponse

case $reponse in

[yYoO]\*) echo "réponse positive";;

[nN]\*) echo "réponse négative";;

\*) echo "réponse incorrecte";;

esac

Ce script va saluer l’usager en function de sa langue.

**#! /bin/bash**

echo “Quelle est votre langue ?”

read langue

case $langue in

francais) echo Bonjour ;;

anglais) echo Hello ;;

espagnol) echo Hola ;;

esac

Ce script renvoit le type de paramètre passé au script.

**#! /bin/bash**

param=$1

case $param in

0|1|2|3|4|5|6|7|8|9 ) echo $param est un chiffre ;;

[0-9]\*) echo $param commence par un chiffre ;;

[a-zA-Z]\*) echo $param commence par une lettre;;

\*) echo $param commence par un caractère inconnu ;;

esac

Un vrai exemple, extrait du script smb (*/etc/rc.d/init.d/smb*)

**#! /bin/bash**

# smb attend un paramètre, récupéré dans la variable $1

case "$1" in

start)

echo -n "Starting SMB services: "

deamon smbd -D

echo

echo -n "Starting NMB services: "

deamon nmbd -D

;;

stop)

echo -n "Shutting SMB services: "

killproc smbd

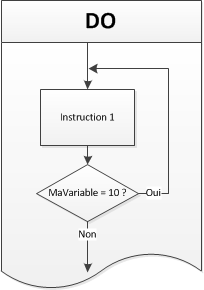
;;

esac

## structures itératives

### boucle for

La boucle **for** permet de parcourir une liste de valeurs et de boucler autant de fois qu'il y a de valeurs.



Concrètement, la forme d'unforest la suivante :

POUR variable PRENANT valeur1 valeur2 valeur3

FAIRE

------> effectuer\_une\_action

VALEUR\_SUIVANTE

Dans cet exemple, la variable va prendre successivement les valeurs valeur1, valeur2, valeur3.

La boucle va donc être exécutée trois fois et la variable vaudra à chaque fois une nouvelle valeur de la liste.

for variable in [liste]

do

commandes *(utilisant $variable)*

done

La liste peut être explicite …

**#! /bin/bash**

for nom in root usager1 usager2

do

echo "changer mot de passe de $nom"

passwd $nom

done

La liste peut être calculée à partir d'une expression …

**#! /bin/bash**

# Recopier les fichiers personnels de root dans /tmp

for ficher in /root/\*

do

cp $fichier /tmp

done

La liste peut être à partir de la variable système **$\*** …

**#! /bin/bash**

# pour construire une liste de fichiers dans $\*

cd /root

set \*

echo $\*

for nom in $\*

do

echo $nom

done

**Exemple**

for nom in u1 u2 u3

do

useradd $nom

done

for i in /home/u[1-3] ; do ls -la $i; done

for i in /home/\*/\* ; do echo $i; done

for i in /dev/tty[1-7]

do

setleds -D +num < $i

echo $i

done

for x in /home/\*

do

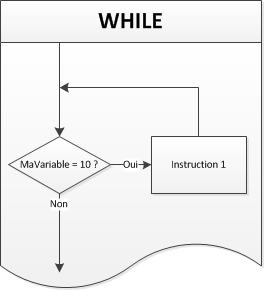
echo $x

done

### Boucle While

La boucle while exécute un bloc d'instructions tant qu'une certaine condition est satisfaite ; lorsque cette condition devient fausse la boucle se termine.

Cette boucle permet donc de faire un nombre indéterminé de tours de boucle, voire infini si la condition ne devient jamais fausse.



Le principe est de faire un code qui ressemble à ceci :

TANT QUE test

FAIRE

------> effectuer\_une\_action

RECOMMENCER

Avec bash, on l'écrit comme ceci …

while [ test ]

do

echo 'Action en boucle'

done

Il est aussi possible, comme pour le if, d'assembler les deux premières lignes en une, à condition de mettre un point-virgule

while [ test ]; do

echo 'Action en boucle'

done

En résumé …

|  |  |
| --- | --- |
| while liste-commandes  do  commandes  done | La répétition se poursuit TANT QUE la dernière commande de la liste est vraie |
| until liste-commandes  do  commandes  done | La répétition se poursuit JUSQU'A CE QUE la dernière commande de la liste devienne vraie |

**Exemple**

Le script scriptboucles.sh qui suit va demander à l'utilisateur de dire **oui** et répéter cette action tant qu’il n'a pas fait ce que l'on voulait …

**#!/bin/bash**

Read -p “Entrez votre réponse” reponse

while [ -z $reponse ] || [ $reponse != 'oui' ]

do

read -p 'Dites oui : ' reponse

done

Comme il s'agit d'un OU (||), tant que l'un des deux tests est vrai, on recommence la boucle.

Cette dernière pourrait se traduire par : « Tant que la réponse est vide ou que la réponse est différente deoui ».

**Autres exemples**

**#!/bin/bash**

echo -e "Entrez un nom de fichier"

read fichier

while [ -z $fichier ] #chaîne vide

do

echo -e "Entrée invalide"

read fichier

done

echo -e" Entrez un nom de fichier"

read fichier

[ -z $fichier ]

do

echo -e "Invalide"

done

**#!/bin/bash**

# Pour dire bonjour toutes les secondes (arrêt par CTRL-C)

while true

do

echo "Bonjour $USER"

sleep 1

done

Dans le prochain exemple, il faut noter que la redirection de l'entrée de la commande *while .. do .. done* est placée à la fin …

**#!/bin/bash**

#Lecture des lignes d'un fichier pour traitement

fichier=/etc/passwd

while read ligne

do

echo $ligne

done < $fichier

### Sortie et reprise de boucle

**break** placé dans le corps d'une boucle, provoque une sortie définitive de cette boucle.

**continue** permet de sauter les instructions du corps de la boucle (qui suivent continue) et de "continuer" à l'itération suivante.

Pour les boucles for, while et until, continue provoque donc la réévaluation immédiate du test de la boucle.

**Exemples**

Boucle de lecture au clavier arrêtée par la saisie de *stop*

**#! /bin/bash**

text=""

while true

do

read ligne

if [ $ligne = stop ]

then

break

else

text="$text \n$ligne"

fi

done

echo -e text=$text

Lecture des lignes d'un fichier …

**#! /bin/bash**

fichier="/etc/passwd"

grep "^usager" $fichier | while true

do

read ligne

if [ "$ligne" = "" ]

then

break

fi

echo $ligne

done

## Fonctions

Une fonction est un ensemble d’instructions, permettant d'effectuer plusieurs tâches avec des paramètres d'entrée différents. Son utilisation permet de rendre un programme plus lisible et structuré. Ainsi, il facilite le développement des scripts.

En plus, il est possible d'appeler une fonction autant de fois que souhaité dans un script.

* Un script n'est pas limité dans le nombre de fonction.   
  Par contre, elles ne doivent pas avoir le même nom.
* Il est possible de réutiliser les variables globales initialisées dans un script à l'intérieur ou à l'extérieur de la fonction.
* Il est possible de déclarer des variables locales à une fonction.
* Une bonne fonction est un élément qui traite d'un élément spécifique et récurrent dans un script.  
  Les fonctions sont un moyen de centraliser les informations destinées à être répétées, afin que l'on puisse s'y référer à chaque endroit où cela est nécessaire.

L'utilisation des fonctions se fait en deux moments …

* Il faut d'abord définir la fonction.  
  Il s’agit de décrire quelle série de commandes il faudra exécuter lorsque l'on appellera la fonction ;
* Il faut ensuite appeler la fonction à chaque moment désiré.

**Syntaxes** …

function nom-fct {

bloc d'instructions

}

ou

nom-fct() {

bloc d'instructions

}

**Exemples**

function test\_fonction

{

echo Fonction test

## 

## Calcul sur les entiers

Trois méthodes permettent d'effectuer des calculs …

* la première utilise la syntaxe spéciale $((operation)) ;
* la seconde utilise la commande let ;
* la troisième utilise la commande bc, qui accepte aussi les nombres décimaux.   
  Saisir **bc** seul sur la ligne de commande permet de passer en mode interactif.

Voici comment on peut incrémenter une variable avec chacune des méthodes …

$ a=1

$ a=$(($a + 1))

$ echo $a  
2

$ a=1

$ let "a=$a + 1"

$ echo $a

2

$ a=1

$ a=$(echo "$a+1" |bc )

$ echo $a

2

Note …  
Il est important de ne pas confondre la syntaxe $((expression arithmétique)) avec la substitution de commande $(commande).

**Exemples**

echo $((30+2)) 32

echo $((30+2\*10/4)) 35

echo $(( (30+2) \* (10-7) /4 )) 24

Calcul de la factorielle (de 1 à 10)

declare -i k

k=1

p=1

while [ $k -le 10 ]

do

echo "$k!=" $((p=$p\*$k))

k= $k+1