

Structures de contrôle et fonctions en shell

Système et environnement de programmation

Université Grenoble Alpes

Plan

- 1 Tests
- 2 Structures conditionnelles
- 3 Itérations
- 4 Fonctions

Plan

- 1 Tests
- 2 Structures conditionnelles
- 3 Itérations
- 4 Fonctions

Code de retour

`$?` = code retour de la dernière commande exécutée

Convention : 0 = commande terminée sans erreurs

Exemples

```
cp toto.txt tutu.txt
```

```
echo $?
```

⇒ affiche 0

```
cp
```

```
echo $?
```

⇒ affiche 1

Plusieurs valeurs positives différentes peuvent être utilisées pour discriminer le type d'erreur (cf. manuel de chaque commande)

Code de retour

Autres usages

- pour certaines commandes, comme `grep`, le code de retour permet de discriminer le résultat

Exemple

```
grep motif fichier
```

`$?` vaut 0 si et seulement si

`motif` apparaît dans `fichier` au moins une fois

- `exit val` termine un script avec `val` comme code de retour
Ce code sera visible via `$?` dans le shell appelant

La commande test

test a pour code de retour 0 si et seulement si l'expression qui lui est donnée en arguments est vraie

Syntaxe : `test expression ou [expression]`

Très utilisée en conjonction avec les structures conditionnelles et les boucles while, surtout sous la seconde forme (compacte)

test, expressions sur les fichiers et répertoires

Syntaxe : option fichier

(attention aux espaces)

option	signification
-e	fich existe
-s	fich n'est pas vide
-f	fich est un fichier
-d	fich est un répertoire
-r	fich a le droit r (lecture)
-w	fich a le droit w (écriture)
-x	fich a le droit x (exécution)

Exemple

```
# code de retour nul si toto.txt existe
test -e toto.txt
```

test, expressions sur les entiers

Syntaxe : n_1 option n_2

(attention aux espaces)

option	signification
-eq	$n_1 = n_2$
-ne	$n_1 \neq n_2$
-lt	$n_1 < n_2$
-gt	$n_1 > n_2$
-le	$n_1 \leq n_2$
-ge	$n_1 \geq n_2$

Exemple

```
# code de retour nul si la variable x représente  
# un entier strictement inférieur à 42  
[ $x -lt 42 ]
```


test, expressions sur les chaines

Syntaxe : option chaine ou chaine₁ option chaine₂
(attention aux espaces)

option	signification
-z	chaine est vide
-n	chaine n'est pas vide
=	les 2 chaines sont identiques
!=	les 2 chaines sont différentes

Exemple

```
# code de retour nul si la variable x contient  
# la chaine "INF203"  
[ $x = "INF203" ]
```

test, opérateurs booléens sur les expressions

Expression	Interprétation logique
<code>expr₁ -a expr₂</code>	ET
<code>expr₁ -o expr₂</code>	OU
<code>! expr</code>	NON

Plus parenthésage : `\(\)`

Exemple

```
# code de retour nul si la variable dir représente :  
# - un répertoire accessible en exécution  
[ -d $dir -a -x $dir ]  
# - un répertoire accessible en lecture ou en execution  
[ -d $dir -a \( -r $dir -o -x $dir \) ]
```

Plan

- 1 Tests
- 2 Structures conditionnelles
- 3 Itérations
- 4 Fonctions

Structure conditionnelle simple

Syntaxe

```
if <condition>
then
    # à exécuter si la condition est vraie
    <suite de commandes 1>
else
    # à exécuter sinon
    <suite de commandes 2>
fi
```

La partie `else` est facultative.

La condition est déterminée par le code de retour (`$?`) d'une commande, souvent on se contente de la commande `test`

Structures conditionnelles imbriquées

Sucre syntaxique pour imbriquer des `if` dans la partie `else` :

```
if <condition 1>
then
    <suite de commandes 1>
elif <condition 2>
then
    <suite de commandes 2>
else
    <suite de commandes 3>
fi
```

Exemples (1)

```
#!/bin/bash
mod=$(expr $1 % 2)
if [ $mod -eq 0 ]
then
    echo $1 est pair
else
    echo $1 est impair
fi
```

Exemples (2)

```
#!/bin/bash
```

```
# attention aux espaces apres '[' et avant ']'
```

```
if [ $1 -gt $2 ]
```

```
then
```

```
    echo $1 est plus grand que $2
```

```
else
```

```
    echo $1 n'est pas plus grand que $2
```

```
fi
```

Exemples (3)

```
#!/bin/bash
```

```
# La condition peut être n'importe quelle commande
```

```
if cd ~/INF203/TP1
```

```
then
```

```
    echo Je peux me rendre dans mon répertoire de TP1
```

```
else
```

```
    echo Problème : mon répertoire de TP1 n'existe pas
```

```
    echo                ou je n'ai pas le droit de m'y rendre
```

```
fi
```


Exemples (4)

```
#!/bin/bash
if [ -d $1 ]
then
    echo "$1 est un répertoire"
elif [ -f $1 ]
then
    echo "$1 est un fichier "
    if [ -x $1 ]
    then
        echo "executable"
    fi
else
    echo "$1 n'existe pas !"
fi
```

Remarque sur le typage (avec test)

```
#!/bin/bash
# le type d'une variable est déduit du contexte
# sa valeur est convertie dans le bon type au besoin
V1=042
if [ $V1 -eq 42 ] # V1 est vue comme un entier
    then echo '$V1 vaut bien 42'
fi
if [ $V1 != 42 ] # V1 est vue comme une chaîne
    then echo "mais \"$V1\" n'est pas 42"
fi
```

Plan

- 1 Tests
- 2 Structures conditionnelles
- 3 Itérations**
- 4 Fonctions

Boucle while

Syntaxe

```
while <condition>
do
    <suite de commandes>
done
```

Exemple

```
#!/bin/bash
echo "donnez un nom de fichier"
read fich
while [ ! -f $fich ]
do
    echo "$fich n'existe pas, donnez un autre nom"
    read fich
done
```

Un while sans test

```
#!/bin/bash
num=1
while read ligne
do
    echo "$num : $ligne"
    num=$((expr $num + 1))
done
```

Boucle for

Syntaxe

```
for <variable> in <liste>
do
    <suite de commandes>
done
```

Exemples (1)

```
#!/bin/bash
```

```
cd $HOME/INF203
for i in 1 2 3
do
    ls -l TP$i
done
```

```
for fich in *.o
do
    rm $fich
done
```

Exemples (2)

```
#!/bin/bash
```

```
if [ ! -d Executables ]  
then  
    mkdir Executables  
fi
```

```
for fich in *  
do  
    if [ -f $fich -a -x $fich ]  
    then  
        mv $fich Executables  
    fi  
done
```


Plan

- 1 Tests
- 2 Structures conditionnelles
- 3 Itérations
- 4 Fonctions**

Definition d'une fonction

Syntaxe

```
<nom_fonction>() {  
    <suite de commandes>  
}
```

Exemples (1)

```
#!/bin/bash
```

```
accumule() {  
    # $i est le ième paramètre de la fonction  
    somme=$(expr $somme + $1)  
}
```

```
somme=0  
for i in 1 2 3 4 5  
do  
    accumule $i  
done
```

```
echo "somme = $somme"
```

Exemples (2)

```
#!/bin/bash
```

```
lire() {  
    somme=0  
    while read ligne  
    do  
        echo "ligne : " $ligne  
        somme=$(expr $somme + $ligne)  
        echo "somme " $somme  
    done  
}
```

```
lire < fichier_d_entiers
```