Les scripts Bash

Partie 2 - Logique de scripting

Sommaire

Au menu :

Les structures conditionnelles

Les tests

Vrai ou faux ?

Pour bash, un test est :

- **vrai** s'il vaut **0**
- **faux** s'il vaut n'importe quelle autre valeur

Ainsi, le code de sortie (**status code**) d'une commande qui a réussi équivaut à vrai et celui d'une commande qui a échouée équivaut à faux

On peut aussi construire des tests avec des opérateurs particuliers :

- avec [**<le test>**]
- ou avec **test <le test>**

Codes de retour

Un exemple

Rappel :
$? permet de récupérer le code de sortie de la dernière commande

```
wilder@host:~$ mkdir newDir
wilder@host:~$ echo $?
0
wilder@host:~$ mkdir newDir
mkdir: impossible de créer le répertoire «newDir»: Le fichier existe
wilder@host:~$ echo $?
1
```

Comparaison de chaînes

Tester des chaînes de caractères

Supposons 2 chaînes s1 et s2

s1 = s2 : vrai si les chaînes sont identiques

s1 != s2 : vrai si les chaînes sont différentes

-z s1 : vrai si s1 est vide

-n s1 : vrai si s1 n'est pas vide

Note : Attention aux espaces !

```
wilder@host:~$ test 'identique' = 'identique'
wilder@host:~$ echo $?
0
wilder@host:~$ test 'identique' = 'différent'
wilder@host:~$ echo $?
1
wilder@host:~$ [ 'identique' = 'identique' ]
wilder@host:~$ echo $?
0
wilder@host:~$ [ 'identique' != 'différent' ]
wilder@host:~$ echo $?
0
wilder@host:~$ [ -z '' ]
wilder@host:~$ echo $?
0
```

Comparaison de nombres

Et avec des nombres ?

Supposons 2 nombres n1 et n2

n1 -eq n2 : vrai si les nombres sont égaux

n1 -ne n2 : 0 si les nombres sont différents

n1 -lt n2 : n1 < n2

n1 -le n2 : n1 <= n2

n1 -gt n2 : n1 > n2

n1 -ge n2 : n1 >= n2

```
wilder@host:~$ trois=3
wilder@host:~$ [ $trois -eq 3 ]
wilder@host:~$ echo $?
0
wilder@host:~$ [ 2 -ne $trois ]
wilder@host:~$ echo $?
0
wilder@host:~$ deux=2
wilder@host:~$ [ $deux -lt $trois ]
wilder@host:~$ echo $?
0
```

Opérateurs logiques

Et avec des combinaisons ?

Supposons c1 et c2 des conditions

! c1 : NON logique
(vrai si c1 est faux et vice versa)
c1 -a c2 : ET logique (vrai si c1 et c2 vrai)
c1 -o c2 : OU logique (faux si c1 et c2 faux)

```
wilder@host:~$ trois=3
wilder@host:~$ [ ! $trois -eq 3 ]
wilder@host:~$ echo $?
1
wilder@host:~$ [ 2 -lt $trois -a $trois
-lt 4 ]
wilder@host:~$ echo $?
0
```

Opérateurs sur les fichiers/dossiers

Et avec les fichiers ?

Supposons p un chemin

-e p : vrai si p existe

-s p : vrai si p existe et de taille > 0

-f p : vrai si p est un fichier

-d p : vrai si p est un dossier

-r p : vrai si je peux lire p

-w p : vrai si je peux écrire p

-x p : vrai si je peux exécuter p

```
wilder@host:~$ [ -e /tmp ]
wilder@host:~$ echo $?
0
wilder@host:~$ [ -f /etc/passwd ]
wilder@host:~$ echo $?
0
wilder@host:~$ [ -r /etc/passwd ]
wilder@host:~$ echo $?
0
wilder@host:~$ [ -w /etc/passwd ]
wilder@host:~$ echo $?
1
```

Si ... Sinon

Et si ?

Structure conditionnelle if

if condition
then
    instructions
elif conditions2 (optionnel)
then
    instructions
else (optionnel)
    instructions
fi

```
wilder@host:~$ if mkdir newDir
> then
>     echo "Création dossier succès"
> else
>     echo "Création dossier échec"
> fi

wilder@host:~$ if [ ! -e newDir ];then
mkdir newDir; else echo "newDir
existe déjà";fi
```

Case

Enumérer les cas

Structure conditionnelle case

case valeur in

valeur1) instructions;;

valeur2 | valeur3) instructions;;

...

***) instructions par défaut;;**

esac

```
wilder@host:~$ case $choice in
> 1)
>       echo "choix 1"
>       echo "Merci";;
> 2)
>       echo "choix 2"
>       echo "Bon choix";;
> esac
wilder@host:~$ case $choice in 1)
echo "choix 1";; 2) echo "choix2 ";;
esac
```

Les structures itératives

Structure itérative

Définition

En algorithmique, on appelle **structure itérative**, une construction d'un langage qui permet la répétition d'instructions

C'est à dire de portions de code dont l'exécution va être effectuée un nombre de fois donné ou tant qu'une **condition** est remplie.

Il est courant de les qualifier de **boucles**

Boucle for

Boucler sur une liste

Structure itérative for

for variable in liste
do
    instructions
done

Liste pouvant être :

- Une suite de mots
- Le résultat d'une substitution
- Le résultat d'une commande

```
wilder@host:~$ for word in "One"
"Two" "Three"
> do
>       echo $word
> done
wilder@host:~$ for number in $(seq 3
-1 0); do echo $number; done

#!/bin/bash
# My own ls
for path in *
do
        echo $path
done
```

Les arguments

À vous de jouer !

Créer un script qui affiche la liste de ses arguments :

- Un argument par ligne
- Numéroté de la forme :
  - 1 - Argument1
  - 2 - Argument2
  - ...

```
#!/bin/bash

# Echo the numbered list of the
script's arguments

number=1
for param in $*
do
        echo "$number - $param"
        number=$(( $number + 1 ))
done
exit 0
```

Boucle for

L'autre for

For (alternative)

for ((e1 ; e2 ; e3))
do
    instructions
done

e1, e2 et e3 sont des expressions arithmétiques

- e1 : effectuée une fois au début
- e2 : continue tant que e2 est vraie
- e3 : effectuée après chaque tour

```
wilder@host:~$ for (( i=1 ; i < 4 ; i++ ))
> do
>       echo $i
> done
1
2
3
```

Boucle while

Boucler sur une liste

Structure conditionnelle while

while <condition>
do
    instructions
done

```
wilder@host:~$ number=3
wilder@host:~$ while [ $number -ge 0
]
> do
>       echo $number
>       number=$(( $number - 1 ))
> done
3
2
1
0
```

Références

La doc officielle
Le wikibooks : Programmation Bash
Le Wiki Bash Hackers
Le Bash Guide de Greg
ExplainShell

Conclusion

Les tests de conditions
Les boucles