

Systeme d'exploitation

Scripting Bash

(6)

Les fonctions

Syntaxe courte pour les conditions

- Autres syntaxes pour la commande **test** :
 - **[...]** : tests simples
 - **[[...]]** : conditions/commandes composées

Attention : **espaces autour des doubles crochets !**

- Exemples
 - if **[-f nom]** # vérifie que nom est un fichier (idem test -d,-l,-x ...)

 - if **[\$# == 3]** # support du symbole == en plus de -eq

 - if **[[\$# == 2 || \$# == 3]]** # vérifie nombre d'arguments : 2 ou 3

Les fonctions

- A définir à chaque fois qu'une suite d'instructions est utilisée plusieurs fois
 - augmente la lisibilité et la ré-utilisabilité
 - permet de paramétrer l'exécution via la transmission d'arguments
- Déclaration

```
function maFonction() {  
    # le mot clé function n'est pas obligatoire => ()  
    # les parenthèses ne sont pas obligatoires => function  
    # le mot clé function doit être le premier de la ligne  
}
```
- Appel de la fonction, comme un script : `maFonction arguments`
- Accès aux arguments, comme un script **variables positionnelles** : `$1 $2 $3 ..`
Attention: `$0` désigne toujours le nom du script (pas celui de la fonction)

Exemple N°1

Déclaration

```
function afficher_taille { # affiche la taille des fichiers fournis en paramètre
    echo "--$1--"
    echo "$( ls -sh $1 | cut -d' ' -f1) octets,"
    echo "$( wc -c $1 | cut -d' ' -f1) "caractères,"
    echo "$( wc -l $1 | cut -d' ' -f1) lignes"
}
```

Utilisation

```
for fichier in *.txt *.sh
do
    afficher_taille $fichier
done
```

Affiche la taille
des fichiers txt et sh

```
--texte.txt--
4,0Koctets,
9 caractères,
1 lignes.
--essai.sh--
4,0Koctets,
292 caractères,
16 lignes.
```

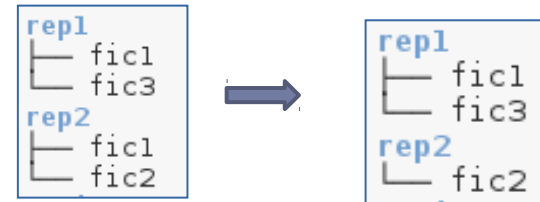
Exemple N°2

- Fonction qui supprime le plus vieux des 2 fichiers passés en paramètres

```
function effacer_vieux () {
    if [ "$1" -nt "$2" ]
    then rm -f "$2"
    else rm -f "$1"
    fi
}
```

Rechercher les fichiers présents dans 2 répertoires et effacer la version la plus vieille

./essai.sh rep1 rep2



```
for fichier in $(ls "$1") # pour les fichiers de $1
do
    if [ -f "$2/$fichier" ] # teste s'il est présent aussi dans $2
    then
        effacer_vieux "$1/$fichier" "$2/$fichier" # supprime le doublon
    fi
done
```

1^{er} argument
=> **\$1** pour la fonction

2^{ème} argument
=> **\$2** pour la fonction

Retour de fonctions

- Une fonction peut retourner explicitement une valeur
sinon c'est le code de retour de la dernière commande qui est retourné
- mot-clé **return N°**
 - sort de la fonction et retourne une valeur **entière**, lisible ensuite via un **\$?**
 - Attention : code de retour \Leftrightarrow valeur comprise entre **0 et 255**
 - **return sans paramètre** permet de sortir de la fonction (~ *break*)
retourne le compte-rendu de la dernière commande
- Pour transmettre d'autres types de valeurs, ou plusieurs valeurs, il faut :
 - soit utiliser des **variables**
 - soit écrire sur le **flux standard** qui sera ensuite redirigé (echo ...)

Exemple : saisie contrôlée de l'heure

```
function controle { # Saisie d'une valeur comprise entre 2 paramètres
    while read -p "Entrer une valeur comprise entre $1 et $2 :" reponse
    do
        if [[ $reponse -ge $1 && $reponse -le $2 ]];then return $reponse ;fi
        echo "erreur : valeur non conforme"
    done
}
```

résultat par valeur de retour

```
saisie_heure(){ # Saisie de l'heure appelant la fonction ci-dessus
    controle 0 23
    heure=$?
    minute=$(controle 0 59)
}
```

résultats par modification de variables


```
saisie_heure
date --set "$heure:$minute" 2>/dev/null
```

la modification de la date par set n'est permise, que pour root

Retour par redirection du flux standard

```
function controle {
    while read -p "entrez une valeur comprise entre $1 et $2 :" reponse ; do
        if [[ $reponse -ge $1 && $reponse -le $2 ]] ; then return $reponse ; fi
        echo "erreur : valeur non conforme"
    done
}

saisie_heure(){
    controle 0 23
    echo -n $?
    controle 0 59
    echo $?
}
```



capture du flux de sortie dans un fichier temporaire

```
saisie_heure >tmp
read heure minute <tmp
date --set "$heure:$minute" 2>/dev/null
```

capture dans une variable :

```
tmp=$(saisie_heure)
set $tmp
date --set "$1:$2" 2>/dev/null
```


Variables locales

Par défaut les variables sont déclarées pour l'interpréteur => globales au script
 Pour forcer une variable à être **locale** à la fonction : **local** *maVariable*

Script **soupe.sh** :

```
var1="poireau"
var2=$1
echo " *:$* - 1:$var1 - 2:$var2"
```

```
function mixer {
local var1="carotte"
echo " *:$* - 1:$var1 - 2:$var2"
var2="navet"
}
```

```
mixer "$2" "$var2"
echo " *:$* - 1:$var1 - 2:$var2"
```

Exécution : ./soupe.sh choux patates

*:choux patates - 1:poireau - 2:choux

*:patates choux - 1:carotte - 2:choux

*:choux patates - 1:poireau - 2:navet