esprit
Se former autrement

Module : Système et Scripting

Année Universitaire
2024-2025

Planification & Informations

- **Chapitre 1** : Introduction & Commandes de base
- **Chapitre 2** : Langage de programmation Shell
- **Chapitre 3** : Conditions & Boucles en Shell
- **Chapitre 4** : Sous-programmes en Shell

Système & Scripting

Chapitre IV : Les sous-programmes et les arguments

Plan

1. Sous-Shells
2. Sous-programme sous forme de fonction
3. Les instructions exit et return.
4. Appel de fonctions
5. Gestion des menus avec Select
6. Récupération des arguments avec getopts

Sous-Shells

Définition :

- Un *sous-Shell* est un processus lancé par un shell (ou ***script Shell***)..
- Ces ***sous-Shells*** permettent au script de faire l'exécution en parallèle, donc d'exécuter différentes tâches simultanément

Syntaxe

(commande1; commande2; commande3; ... commandeN;)

Une liste de commandes placées entre *parenthèses* est exécutée sous forme de sous-Shells

Pourquoi Utiliser des Sous-Shells ?

1. Isolation de l'environnement

- Les sous-shells permettent d'exécuter des commandes dans un espace séparé sans affecter l'environnement du shell parent.
- Par exemple, les variables modifiées ou créées dans un sous-shell ne sont pas accessibles depuis le shell parent :

```
#!/bin/bash
var=parent
(var="sous-Shell"; echo $var)
echo $var
```

```
esprit@esprit:~/Desktop$ ./sousShell
sous-Shell
parent
```

```bash
#!/bin/bash

# Déclaration d'une variable dans le shell principal
variable_principale="Bonjour depuis le shell principal"
echo "Dans le shell principal : variable_principale = '$variable_principale'"

# Exécution dans un sous-shell
(
    # Déclaration d'une variable dans le sous-shell
    variable_sous_shell="Bonjour depuis le sous-shell"
    echo "Dans le sous-shell : variable_principale = '$variable_principale'"
    echo "Dans le sous-shell : variable_sous_shell = '$variable_sous_shell'"

    # Modification de variable_principale dans le sous-shell
    variable_principale="Modifiée dans le sous-shell"
    echo "Dans le sous-shell après modification : variable_principale = '$variable_principale'"
)

# Retour dans le shell principal, vérifier les valeurs des variables
echo "Retour dans le shell principal : variable_principale = '$variable_principale'"
echo "Retour dans le shell principal : variable_sous_shell = '${variable_sous_shell:-non définie}'"
```

```
esprit@esprit:~/Desktop$ ./script1
Dans le shell principal : variable_principale = 'Bonjour depuis le shell principal'
Dans le sous-shell : variable_principale = 'Bonjour depuis le shell principal'
Dans le sous-shell : variable_sous_shell = 'Bonjour depuis le sous-shell'
Dans le sous-shell après modification : variable_principale = 'Modifiée dans le sous-shell'
Retour dans le shell principal : variable_principale = 'Bonjour depuis le shell principal'
Retour dans le shell principal : variable_sous_shell = 'non définie'
```

Étendue des variables dans un sous-Shell

Les variables utilisées dans un sous Shell *ne sont pas* visibles en dehors du code du sous-shell. Elles ne sont pas utilisables par le **processus parent**, le Shell qui a lancé le sous-Shell. Elles sont en réalité des **variables locales**.

Exemple

```
#!/bin/bash
#shell principal
var1="toto"
var2="titi"

echo " nous somme dans le shell principal"
echo "var1=$var1"
echo "var2=$var2"

#Sous Shell
(
echo "nous somme à l'interieur  du script sous schell"
var1="bonjour"
var2="welcome"
echo "var1=$var1"
echo "var2=$var2"
)

echo " nous somme à l'extérieur  du script sous schell"
echo "var1=$var1"
echo "var2=$var2"

echo $?
```

```
[root@node1 ~]# ./test_sous_shell
 nous somme dans le shell principal
var1=toto
var2=titi
nous somme à l'interieur  du script sous schell
var1=bonjour
var2=welcome
 nous somme à l'extérieur  du script sous schell
var1=toto
var2=titi
0
```

Fonctions

Définition et Syntaxe:

- Bash supporte les fonctions
- Une fonction est un **bloc de code** qui implémente un ensemble d'opérations.

function nom_fonction {

commande...

}

Ou

nom_fonction () {

commande...

}

- **Les fonctions sont appelées, *lancées*, simplement en invoquant leur noms**.

Fonctions

Définition et Syntaxe:

- Les fonctions peuvent récupérer des arguments qui leur sont passés et renvoyer un code de sortie au script pour utilisation ultérieure.

nom_fonction $arg1 $arg2

- La fonction se réfère aux arguments passés par leur position (comme s'ils étaient des **paramètres positionnels**), c'est-à-dire $1, $2 et ainsi de suite.

Fonctions

Exemple

```
#!/bin/bash

# Fonction pour créer un fichier
create_file() {
     if [ -d  $1 ]; then
        touch "$1/$2"
        echo "Le fichier '$2' a été créé dans le répertoire '$1'."
    else
        echo "Erreur : Le répertoire '$1' n'existe pas."
    fi
}
# Fonction pour lister les fichiers dans un répertoire
list_files() {
        if [ -d $1 ]; then
        echo "Liste des fichiers dans '$1' :"
        ls -l "$1"
    else
        echo "Erreur : Le répertoire '$1' n'existe pas."
    fi
}
create_file $1 $2
list_files $1
```

```
t@esprit:~/Desktop$ ./script-function rep file1
chier 'file1' a été créé dans le répertoire 'rep'.
 des fichiers dans 'rep' :
 0
w-r-- 1 esprit esprit 0 11:10 6  ديسمبر file1
t@esprit:~/Desktop$ ./script-function rep1 file1
r : Le répertoire 'rep1' n'existe pas.
r : Le répertoire 'rep1' n'existe pas.
t@esprit:~/Desktop$ ./script-function rep file2
chier 'file2' a été créé dans le répertoire 'rep'.
 des fichiers dans 'rep' :
 0
w-r-- 1 esprit esprit 0 11:10 6  ديسمبر file1
w-r-- 1 esprit esprit 0 11:11 6  ديسمبر file2
```

Sortie et retour d'une fonction

- **Code de sortie:** Les fonctions renvoient une valeur, appelée un *code (ou état) de sortie*. Le code de sortie peut être explicitement spécifié par une instruction **return**, sinon, il s'agit du code de sortie de la dernière commande de la fonction (0 en cas de succès et une valeur non nulle sinon).

Terminaisons

- un script : **exit n** (n = code de retour indiquant succès ou échec).
- une fonction : **return n** (n = code de retour indiquant succès ou échec).
- une boucle **: break**

Exemple :

```
#!/bin/bash
E_PARAM_ERR=250          # Si moins de deux paramètres passés à la fonction.
EGAL=251                 # Code de retour si les deux paramètres sont égaux.
max ()                   # Envoie le plus important des deux entiers.
{
if [ -z "$2" –o –z "$1" ]
then
  return $E_PARAM_ERR
fi

if [ "$1" -eq "$2" ]
then
  return $EGAL
else
  if [ "$1" -gt "$2" ]
  then
    return $1
  else
    return $2
  fi
fi
}
```

Exemple (suite)

```
Max 33 34  #appel de la fonction.
return_val=$?

if [ "$return_val" -eq $E_PARAM_ERR ]
then
      echo "Vous devez donner deux arguments à la fonction."
elif [ "$return_val" -eq $EGAL ]
  then
    echo "Les deux nombres sont identiques."
else
    echo "Le plus grand des deux nombres est $return_val."
fi

  exit 0
```

Gestion de menu avec select

La commande interne **select** est une structure de contrôle de type boucle qui permet d'afficher de manière cyclique un menu.

La liste des items sera affichée à l'écran à chaque tour de boucle.

Les items sont indicés automatiquement.

La variable ***var*** sera initialisée avec l'item correspondant au choix de l'utilisateur.

select var in item1 item2 item3 ..... itemn

do

commandes

done

La commande Select

- Cette commande utilise également deux variables réservées .
- La variable **PS3** représente le prompt utilisé pour la saisie du choix de l'utilisateur.

- La variable **REPLY** qui contient l'indice de l'item sélectionné.
- La variable **var** contient le libellé du choix .

Exemple

```
#!/bin/bash

PS3="Votre choix :"

select item in "- Sauvegarde -" "- Restauration -" "- Fin -"
  do
     echo "Vous avez choisi l'item $REPLY : $item"
      case $REPLY in
           1)
               # Appel de la fonction sauve
                echo "Lancement de la sauvegarde"
               ;;
           2)
                # Appel de la fonction restaure
                echo "Lancement de la sauvegarde"
               ;;
           3)
               echo "Fin du script"
               exit 0
              ;;
           *)
               echo "Choix incorrect"
               ;;
       esac
  done
```

```
[root@storage ~]# ./script
1) - Sauvegarde -
2) - Restauration -
3) - Fin -
Votre choix : 1
Vous avez choisi l'item 1 : - Sauvegarde -
Lancement de la sauvegarde
Votre choix : 2
Vous avez choisi l'item 2 : - Restauration -
Lancement de la restauration
Votre choix : 3
Vous avez choisi l'item 3 : - Fin -
Fin du script
```

Récupération des arguments avec getopts

- ***Syntaxe***

  getopts listeOptionsAttendues option

- La commande interne **getopts** permet à un script d'analyser les options passées en argument.
- Pour vérifier la validité de chacune des options, il faut appeler **getopts** à partir d'une boucle.

Récupération des arguments avec getopts (suite)

- Pour **getopts**, une option est composée d'un caractère précédé du signe "-".
- L'appel à la commande **getopts** récupère l'option suivante et retourne un code vrai tant qu'il reste des options à analyser.

Exemple

```
#!/bin/bash

    while getopts "ab:e:" option
    do
        echo "getopts a trouvé l'option $option"
        case $option in
            a)
                echo "Exécution des commandes de l'option a"
                echo "Indice de la prochaine option à traiter : $OPTIND"
                ;;
            b)
                echo "Exécution des commandes de l'option b"
                echo "Liste des arguments à traiter : $OPTARG"
                echo "Indice de la prochaine option à traiter : $OPTIND"
                ;;
            e)
                echo "Exécution des commandes de l'option e"
                echo "Liste des arguments à traiter : $OPTARG"
                echo "Indice de la prochaine option à traiter : $OPTIND"
                ;;
        esac
    done
    echo "Analyse des options terminée"
    exit 0
```

Exécution du script avec getops

$ ./test_getopts -a -b toto -e tata titi

La liste des options utilisables avec ce script sont définies à la ligne (getopts "**ab:e:**" option). Il s'agit des options -a, -b et -e.

Le caractère ":" inscrit après les options " b" et "e" (**getopts "ab:e:" option**) indique que ces options doivent être suivies obligatoirement d'un argument.

La variable "option" (**getopts "ab:e:" option**) permet de récupérer la valeur de l'option en cours de traitement par la boucle **while**.

La variable réservée **"$OPTIND**" contient l'indice de la prochaine option à traiter.

La variable réservée **"$OPTARG**" contient l'argument associé à l'option

Exécution du script avec getops

```
[root@node1 ~]# ./test_getopts -a -b toto -e tata,titi
getopts a trouvé l'option a
Exécution des commandes de l'option a
Indice de la prochaine option à traiter : 2
getopts a trouvé l'option b
Exécution des commandes de l'option d
Liste des arguments à traiter : toto
Indice de la prochaine option à traiter : 4
getopts a trouvé l'option e
Exécution des commandes de l'option e
Liste des arguments à traiter : tata,titi
Indice de la prochaine option à traiter : 6
Analyse des options terminée
```

```
[root@node1 ~]# ./test_getopts -a -b
getopts a trouvé l'option a
Exécution des commandes de l'option a
Indice de la prochaine option à traiter : 2
./test_getopts : l'option nécessite un argument -- b
getopts a trouvé l'option ?
Analyse des options terminée
```