Université d'AIN TEMOUCHENT – Algérie

Faculté des Sciences et de la Technologie

Département de Mathématiques et d'Informatique

Matière: **Système d'exploitation avancé**

Chapitre : Les conditions	**TP n° : 04 (1/3)**

1. Notion de fonction

Pour écrire une fonction ou une procédure en script Shell, vous pouvez utiliser la syntaxe suivante :

```
#!/bin/bash
ma_fonction() {
  # Corps de la fonction
  echo "Ceci est ma fonction"
  # Vous pouvez ajouter d'autres instructions ici
}
```

Voici une explication de ce code :

1) `ma_fonction` est le nom de la fonction que vous créez. Vous pouvez lui donner le nom de votre choix.
2) Les parenthèses () contiennent les paramètres de la fonction, mais dans cet exemple, la fonction n'accepte aucun paramètre.
3) Les accolades {} définissent le corps de la fonction. C'est l'endroit où vous écrivez le code que la fonction exécutera. Dans cet exemple, nous utilisons `echo` pour afficher du texte.

Pour appeler cette fonction, vous pouvez simplement écrire son nom suivi de parenthèses :

```
ma_fonction
```

Voici un exemple plus complet d'une fonction Bash qui accepte des paramètres :

```
#!/bin/bash
ma_fonction_avec_parametres() {
  echo "Le premier paramètre est : $1"
  echo "Le deuxième paramètre est : $2"
  # Vous pouvez ajouter d'autres instructions ici
}
# Appeler la fonction en passant des arguments
ma_fonction_avec_parametres "Valeur1" "Valeur2"
```

Dans cet exemple, la fonction `ma_fonction_avec_parametres` accepte deux paramètres, que vous pouvez passer lorsque vous l'appelez.

2. Variable local vs variable global:

Le mot clé `local` est utilisé dans les scripts Shell, en particulier dans le langage de script Bash, pour déclarer des variables locales. Une variable locale est une variable dont la portée est limitée à l'intérieur de la fonction ou du bloc où elle est définie. Cela signifie que la variable n'est accessible que dans cet espace spécifique et ne peut pas être utilisée en dehors de celui-ci.

Voici un exemple d'utilisation du mot clé local dans un script Bash pour déclarer des variables locales:

```
#!/bin/bash
ma_fonction() {
local variable_locale="Ceci est une variable locale"
echo "Dans la fonction : $variable_locale"
}
variable_locale="Ceci est une variable globale"
ma_fonction
echo "En dehors de la fonction : $variable_locale"
```

Voici un exemple d'une fonction qui calcule la formule suivante x×y+z

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

```
calculer() {
local x="$1"  # Premier paramètre
local y="$2"  # Deuxième paramètre
local z="$3"  # Troisième paramètre

  # Effectuer le calcul
local resultat=$((x /* y + z+m))

  # Renvoyer le résultat
  echo "$resultat"
}

# Appeler la fonction avec des valeurs spécifiques pour x, y et z
m=10
a=5
b=3
c=7

resultat=$(calculer "$a" "$b" "$c")
echo "Le résultat de $a /* $b + $c + $m est : $resultat"
```

3. Si

La syntaxe dans bash est:

Code : Bash

```
if [ test ]
then
echo "C'est vrai"
fi
```

Remarques:

1) *Le mot "`fi`" (`if` à l'envers !) marque la fin du bloc `if`.*
2) *Les espaces à l'intérieur des crochets sont obligatoires. On écrit `[ test ]` et non `[test]`.*

Une autre façon d'écrire le `if`: en plaçant le `then` sur la même ligne précédé par un point-virgule après les crochets :

Code : Bash

```
if [ test ]; then
echo "C'est vrai"
fi
```

Créez un nouveau script:

Code : Bash

```
#!/bin/bash
nom="Karim"
if [ $nom = "Karim" ]
then
echo "Salut Karim !"
fi
```

Ce script affichera :

Code : Console

```
Salut Karim !
```

Notez aussi que vous pouvez tester 2 variables à la fois dans le `if` :

Code : Bash

```
#!/bin/bash
nom1="Karim"
nom2="Yacine"
if [ $nom1 = $nom2 ]
```

```
then
echo "Salut les jumeaux !"
fi
```

Comme ici $nom1 est différent de $nom2, le contenu du `if` ne sera pas exécuté. Le script n'affichera donc rien.

4. Sinon

La syntaxe est comme suit :

Code: Bash

```
if [ test ]
then
echo "C'est vrai"
else
echo "C'est faux"
fi
```

Ajoutons un `else` au précédant script :

Code : Bash

```
#!/bin/bash
if [ $1 = "Karim" ]
then
echo "Salut Karim !"
else
echo "Je ne te connais pas !"
fi
```

De quoi vous rappelle le symbole `$1` ?

Testez le script:

Code : Console

```
$ ./conditions.sh Karim
Salut Karim !
```

Changez de paramètre :

Code : Console

```
$ ./conditions.sh Ali
Je ne te connais pas !
```

Remarque :

Notez que ce script échoue si on n'ajoute pas de paramètre. Il faudrait d'abord vérifier dans un `if` s'il y a au moins un paramètre. Nous apprendrons à faire cela ultérieurement.

5. Sinon si

Il existe aussi le mot-clé "`elif`", abréviation de "`else if`", qui signifie "sinon si". Sa forme ressemble à cela :

Code : Bash

```
if [ test ]
then
echo "Le premier test a été vérifié"
elif [ autre_test ]
then
echo "Le second test a été vérifié"
elif [ encore_autre_test ]
then
echo "Le troisième test a été vérifié"
else
echo "Aucun des tests précédents n'a été vérifié"
fi
```

On peut mettre autant de "`elif`" qu'on veut. En revanche, on ne peut mettre qu'un seul "`else`", qui sera exécuté à la fin si aucune des conditions précédentes n'est vérifiée.

On peut reprendre notre script précédent et l'adapter pour utiliser des `elif` :

Code : Bash

```
#!/bin/bash
if [ $1 = "Karim" ]
then
echo "Salut Karim !"
elif [ $1 = "Zakaria" ]
then
echo "Bien le bonjour Zakaria"
elif [ $1 = "Ali" ]
then
echo "Hé Ali, ça va ?"
else
echo "Je ne te connais pas !"
fi
```

Exercice 1 :

Créer un script '`calculatrice`' dans lequel deux nombres opérandes et un signe opérateur (`+, -, ×, /`) devront être donnés en paramètres, ou saisis. Le script doit réaliser l'opération souhaitée.

Exemple :

```
$ calculatrice 7 + 4
Le résultat est : 11
```

NB. Chaque opération doit être réalisé dans une fonction à part.