INF2132 : Systèmes d'Exploitation

TP10 - Construction de Scripts Shell Basiques - Partie 3

Nom de l'enseignant :
Pr. Ilias Tougui

Nom de l'assistant :
Pr. Yasser Aderghal

Table des matières

Lisez attentivement cette page

I Objectifs du TP

* Apprendre l'utilisation des fonctions.
* Apprendre à utiliser les structures conditionnelles switch (ou case).
* Développer une méthodologie de résolution de problèmes.

II Prérequis

Avant de commencer ce TP, vous devez :

* Assister au CM : Programmation Shell sous Linux.
* Maîtriser les commandes de base Linux.
* Connaître les bases du shell bash.
* Avoir un Terminal Linux (Ubuntu ou autre distribution)
* Avoir un Éditeur de texte comme `nano`

III Les Fonctions dans les Scripts Shell (Bash)

Les fonctions sont des blocs de code de script auxquels vous attribuez un nom. Elles permettent de réutiliser ce bloc n'importe où dans votre code. Chaque fois que vous devez exécuter ce bloc, il vous suffit d'utiliser le nom de la fonction (ce qu'on appelle appeler la fonction).

Voici comment créer et utiliser les fonctions dans vos scripts shell.

III.1 Déclaration d'une Fonction (Format Classique)

Le format le plus courant utilise le nom de la fonction suivi de parenthèses :

```
nom_fonction() {
  commandes
}
```

— `nom_fonction` : Définit le nom unique attribué à la fonction. Chaque fonction doit avoir un nom unique.

— `commandes` : Représente une ou plusieurs commandes Bash qui constituent le corps de votre fonction. Lorsque vous appelez la fonction, Bash exécute les commandes dans l'ordre, comme dans un script normal.

III.2 Comment la Fonction Prend-elle des Arguments ?

Les arguments sont passés à la fonction directement après son nom lors de l'appel. À l'intérieur du corps de la fonction, ces arguments sont accessibles via des variables positionnelles spéciales ; exactement comme dans un script shell classique (voir annexe).

```
afficher_arguments() {
  echo "Le premier argument est : $1"
  echo "J'ai reçu $# arguments."
}

# Appel de la fonction
afficher_arguments "Bonjour" 42
```

III.3 Comment Déclarer des Variables ?

Par défaut, toutes les variables déclarées dans une fonction sont globales (elles sont accessibles et peuvent être modifiées partout dans le script).

Pour déclarer une variable qui est locale (accessible uniquement dans le corps de la fonction), utilisez le mot-clé `local` :

```
exemple_variables() {
  local variable_locale="Ceci est local à la fonction"
  variable_globale="Ceci est accessible partout"
  echo $variable_locale
}
```

1 Préparation de l'Environnement de Travail

Durée : 20 min, Note : 5 points

En utilisant les commandes de base de Linux et les concepts de scripting shell, suivez les consignes ci-dessous pour créer automatiquement l'environnement de travail structuré de ce TP.

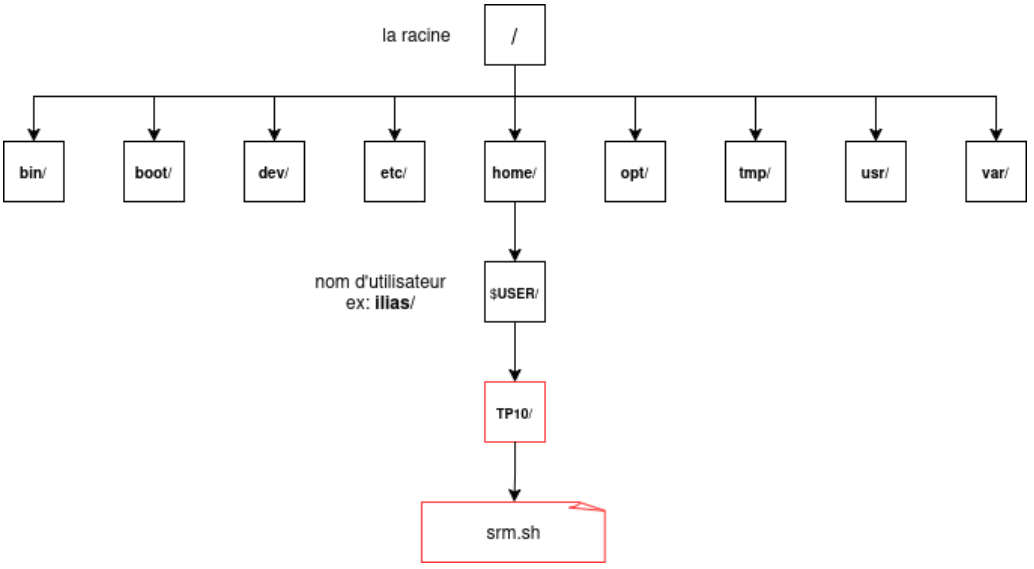

FIGURE 1 – Structure de l'environnement de travail.

1. Déplacez-vous dans la racine.
2. Déplacez-vous ensuite dans votre répertoire personnel.
3. Créez le répertoire de travail `TP10` et accédez-y
4. Téléchargez le fichier srm.sh depuis Connect (ou Outlook) et placez-le dans TP10.
5. Utilisez la commande : less -N srm.sh. Vérifiez que le fichier est créé et examinez rapidement son contenu.
6. Maintenant, nous voulons créer plusieurs répertoires (dossiers) et fichiers pour une simulation. Au lieu de les créer manuellement, nous allons utiliser cette **Expansion de Brace ({x..y}, {x,y})** dans les commandes suivantes. Exécutez-les l'une après l'autre.
   - `mkdir dir{1..3}`
   - `mkdir -p dir1/{dev,test,prod}`
   - `touch dir{1..3}/file{a..d}`
7. Nous sommes actuellement dans le répertoire (`TP10`). Maintenant, exécutez la commande `tree .` et qu'observez-vous ? (Si vous obtenez une erreur, installez la command avec `sudo apt update && sudo apt install tree`).
8. Essayez de créer des sous-répertoires et des fichiers avec des noms de votre choix en utilisant l'Expansion de Brace.
9. **Appelez le professeur pour vérifier votre environnement.**

2 Comprendre les fonctions en Shell avec exemples

Durée : 50 min, Note : 5 points

On va mettre ces connaissances à profit pour programmer une nouvelle commande très utile pour les utilisateurs : pouvoir supprimer un fichier comme avec la commande `rm`, mais avec la possibilité de le récupérer dans une corbeille. Le principe est de déplacer le fichier à supprimer dans un répertoire `~/.corbeille` au lieu de le supprimer définitivement. Cette corbeille est stockée dans le repertoire personelle de l'utilisateur, donc facilement accessible de partout. Vider la corbeille revient simplement à supprimer ce dossier.

La commande srm.sh sera le script complet qui automatisera toutes ces opérations pour nous.

1. Rappel : Vous devez être dans le répertoire `TP10`
2. Lisez le bloc de code suivant et, sur une feuille, expliquez chaque ligne. (voir la définition de la fonction à la Section III, Voir CM)

```
CORBEILLE="$HOME/.corbeille"

creer_corbeille() {
    # Vérifier si le dossier corbeille existe, sinon le créer
    if [ ! -d "$CORBEILLE" ]; then
        mkdir -p "$CORBEILLE"
    fi
}
```

3. Créez un fichier nommé **myDel.sh**, ajoutez la ligne de **shebang** en première ligne, puis ajoutez le bloc de code ci-dessus.
4. Définissez les permissions du fichier de façon à ce qu'il soit exécutable uniquement par l'utilisateur. (bien sûr en utilisant chmod).
5. Exécutez votre programme. S'il fonctionne, vérifiez le résultat ; sinon, proposez une solution.
6. Lisez le bloc de code suivant et, sur une feuille, expliquez chaque ligne. (À noter que `2>` signifie que la sortie d'erreur de la commande doit être redirigée, et que `/dev/null` est un fichier spécial dans lequel tout ce qui est écrit disparaît ; il est surnommé le "Black hole" de Linux.)

```
lister_corbeille() {
    # Vérifier si la corbeille est vide
    if [ ! "$(ls $CORBEILLE 2>/dev/null)" ]; then
        echo "La corbeille est vide !"
        return
```

```
    fi

    echo "Fichiers dans la corbeille:"
    echo "-------------------------"
    # Parcourir chaque fichier
    for element in "$CORBEILLE"/*; do
        echo "  $(basename "$element")"
    done
    echo "-------------------------"
}
```

7. Exécutez maintenant la commande ./srm.sh dir1/filea. Vérifiez que filea a été supprimé et que le fichier a été déplacé dans la corbeille.
8. Exécutez maintenant la commande ./srm.sh dir1/dev. Vérifiez que le répertoire a été supprimé et qu'il a été déplacé dans la corbeille.
9. Exécutez maintenant la commande ./srm.sh –help
10. Remarquez que le programmeur de ce script ont prévu l'option `-r` pour pouvoir supprimer un répertoire avec elle. Cependant, nous avons réussi à supprimer le répertoire `dir1/dev` sans utiliser cette option.

    Ouvrez le script srm.sh et essayez d'identifier et corriger ce bug.
11. Maintenant, essayez de supprimer un répertoire de votre choix avec la commande srm.sh sans spécifier l'option `-r`. Est-ce que cela fonctionne ?
12. **Appelez le professeur pour vérifier votre solution.**

Optionnel :

— Exécutez maintenant la commande ./srm.sh -r dir2/filea. Vérifiez que filea a été supprimé et que le fichier a été déplacé dans la corbeille.

— Remarquez que maintenant, lorsque vous utilisez l'option `-r`, vous pouvez supprimer à la fois un fichier et un répertoire. Cependant, le programmeur de ce code souhaite que cette option ne serve qu'à supprimer les répertoires.

Ouvrez le script srm.sh et essayez d'identifier et de corriger ce bug.

— Maintenant, essayez de supprimer un fichier de votre choix avec l'option `-r` . Est-ce que cela fonctionne ?

— Essayez de transformer ce script en programme global. Placez ce programme dans le répertoire `/usr/local/bin`.

— Utilisez la commande `source ~/.bashrc` pour recharger votre Bash. (Quelle est la différence entre la commande `source` et la commande `bash` ?)

— Maintenant, vous pouvez utiliser votre commande depuis n'importe quel emplacement sur votre système.

3 Comprendre les switch cases dans shell script

Durée : 50 min, Note : 5 points

Dans cette partie du TP, nous voulons comprendre comment utiliser le `switch case` dans un script shell.

1. Rappel : Vous devez être dans le répertoire TP10
2. Créez un fichier nommé `switchCase1.sh`, ajoutez le bloc de code ci-dessous. (Utilisez l'éditeur de votre choix).

```
#!/bin/bash
fichier=$1
case $fichier in
*.txt)
    echo "Fichier texte détecté"
    cat $fichier;;
*.sh)
    echo "Script shell détecté"
    bash $fichier;;
*.pdf)
    echo "Document PDF détecté"
    evince $fichier &;;
*)
    echo "Type de fichier inconnu";;
esac
```

3. Définissez les permissions du fichier de façon à ce qu'il soit exécutable uniquement par l'utilisateur.
4. Exécutez ce programme avec : `./switchCase1.sh switchCase1.sh`
5. Sur une feuille, essayez d'expliquer ce que fait le programme, ainsi que la signification et le rôle des mots-clés `echo`, `case ... in *.txt) *.pdf)`
6. L'astérisque `*` ou joker comme on l'appelle sous Linux :

   selon vous, que fait-il exactement dans ce cas ?
7. Faites des essais avec la commande suivante : `ls -l *.sh`.

   Le joker se comporte-t-il de la même manière que dans le switch case ?
8. Créez un fichier nommé `switchCase2.sh`, ajoutez le bloc de code ci-dessous. (Utilisez l'éditeur de votre choix).

```
#!/bin/bash
echo "Choisissez une option:"
echo "1) Lister les fichiers"
echo "2) Afficher la date"
echo "3) Quitter"
read -p "Votre choix: " choix
```

```
case $choix in
1)
    ls -l;;
2)
    date;;
3)
    echo "Au revoir!"
    exit 0;;
*)
    echo "Option invalide";;
esac
```

9. Définissez les permissions du fichier de façon à ce qu'il soit exécutable uniquement par l'utilisateur (propriétaire), puis exécutez et interagissez ce programme.
10. Si vous remarquez bien, dans ce programme l'option 3 ne fait pas vraiment grand-chose, à part afficher un message et retourner le statut 0. Or, ici, on aimerait que le programme soit interactif : tant que l'utilisateur ne choisit pas l'option 3, le programme doit continuer à tourner.

    Proposez une idée sur la manière d'ajouter cette fonctionnalité, et intégrez-la ensuite dans le script `switchCase2.sh` (Voir CM)
11. Sur une feuille, écrivez une fonction nommée `chercher_fichier` qui doit respecter les consignes suivantes :
    - Inclure cette ligne : `read -p "Écrire le nom du fichier : " file`.
    - Vous devez inclure un test pour vérifier si ce que l'utilisateur a saisi est un fichier ou non (cela nécessitera éventuellement une instruction `if-else`).
    - Afficher si le fichier existe dans le répertoire courant ou non.
    - **Vérifiez votre fonction avec votre professeur.**
12. Ensuite, dans votre script et entre le **shebang** et le reste de votre code shell, incluez votre fonction et enregistrez le fichier.
13. Est-ce que votre fonction fonctionnerait ? Pourquoi ?
14. Ajoutez un nouveau cas dans vos `switch cases` pour inclure votre fonction. Donnez-lui par exemple l'option 4. (N'oubliez pas le `;;` pour marquer la fin de votre cas.)
15. Pour aider l'utilisateur à utiliser votre programme,

    ajoutez `echo "4) chercher un fichier"` pour afficher ce que fait l'option 4.
16. Interagissez avec votre nouveau programme.
17. **Appelez le professeur pour vérifier votre solution.**

Annexe : Référence Rapide Bash

Variables Spéciales

Variable	Description
`$0`	Nom du script
`$1, $2, ...`	Arguments positionnels (1er, 2ème, etc.)
`$#`	Nombre d'arguments
`$@`	Tous les arguments (séparés)
`$*`	Tous les arguments (une seule chaîne)
`$?`	Code de retour de la dernière commande (0 = succès)

Opérateurs de Comparaison

Pour les Nombres

Opérateur	Signification	Exemple
`-eq`	égal à	`[ $a -eq 5 ]`
`-ne`	différent de	`[ $a -ne 5 ]`
`-lt`	inférieur à	`[ $a -lt 10 ]`
...	...	...

Pour les Chaînes

Opérateur	Signification	Exemple
`=`	égal à	`[ "$a" = "texte" ]`
`!=`	différent de	`[ "$a" != "texte" ]`
`-z`	chaîne de caractères vide	`[ -z "$a" ]`
`-n`	chaîne de caractères non vide	`[ -n "$a" ]`

Tests sur les Fichiers

Test	Description
`-f fichier`	Fichier régulier existe
`-d fichier`	Répertoire existe
`-r fichier`	Fichier lisible
`-w fichier`	Fichier modifiable
`-x fichier`	Fichier exécutable

Arithmétique

Syntaxe	Exemple
`$(( ))`	`resultat=$((a + b))` (entiers uniquement)
`bc`	`echo "scale=2; 10/3" | bc` (avec décimales)