TD4 : Les scripts shell
V1.0.4

Document en ligne : www.mickael-martin-nevot.com

1 Généralités

Sans mention contraire, vous vous positionnerez dans votre (sous-) répertoire `tp4` durant l'ensemble de ce TD.

N'oubliez pas de consulter le manuel à chaque fois que cela est nécessaire. Vous pouvez aussi faire des recherches sur le Web en prenant soin de vérifier que les informations trouvées soient correctes.

2 Premier script

2.1 Définition d'un script *shell*

Vous connaissez un certain nombre de commandes **de base** Linux (et leur utilisation) mais il est également possible de créer des commandes **utilisateur** offrant de nouvelles fonctionnalités. Parmi ces commandes, il y a les scripts (*shell*).

Un script est un **fichier texte** (qui doit donc être modifié avec un éditeur de texte) qui peut être interprété par le *shell* une fois lancé. Un script est composé d'une suite d'instructions élémentaires, dont éventuellement des commandes Linux, exécutées de manière séquentielle (il s'agit des bases de la programmation procédurale, ou « traditionnelle »).

Idéalement, un fichier script doit être **exécutable** (c.-à-d. avoir les droits en exécution) : ainsi, il pourra être « lancé » avec une commande Linux.

L'utilisation d'une commande permettant d'exécuter un script est logiquement identique à celle d'une commande de base Linux. Cependant, comme par défaut un script n'est pas dans un répertoire défini dans la variable d'environnement `PATH`, il faut préfixer les chemins relatifs par `./`. Par exemple :

- `./bonjour` ;
- `./tp3/bonjour` ;
- `~/initiation-informatique/tp4/bonjour`.

Plus souvent encore que pour l'utilisation des commandes Linux, la syntaxe d'écriture d'un script peut légèrement varier d'un *shell* à un autre, même si la majorité des éléments vus dans le cadre de cet enseignement est standard.

2.2 Exercice

Ouvrez un éditeur de texte (en arrière-plan) puis **tapez** les lignes suivantes (la première ligne réfère toujours à un *shell* : il ne doit absolument rien y avoir avant) :

```
#!/bin/bash
echo "Salut $USER!"
echo "Ceci est mon premier script!"
```

Sauvegardez ce fichier avec le nom `hello`, puis ajoutez-lui le droit en exécution (pour vous).

3 Arguments

3.1 Paramètre et argument

3.1.1 Paramètre

Un **paramètre** (ou paramètre formel ou encore argument muet) est la **variable** utilisée dans un script, ou plus généralement dans la définition d'un programme, d'une procédure ou d'une fonction.

3.1.2 Argument

Un **argument** (ou paramètre effectif) est la **valeur** (ou variable) fournie dans une ligne de commande ou lors de l'appel d'un programme, d'une procédure ou d'une fonction.

3.1.3 Paramètre positionnel

Un **paramètre positionnel** est une variable dont le nom est un numéro.

Sous Linux, les arguments des lignes de commande sont automatiquement enregistrés dans des variables (ou paramètres positionnels) :

- `0` (valeur : `$0`) : nom de la commande ;
- `1`, `2`, etc. (valeur : `$1`, `$2`, etc.) : argument 1, argument 2, etc.

Il existe aussi certaines variables auxiliaires aux paramètres positionnels (également appelées variables prépositionnées) :

- `#` (valeur : `$#`) : nombre d'arguments ;
- `*` (valeur : `$*`) : l'ensemble des arguments ;
- etc.

3.2 Mise en pratique

Créez un script qui affiche son nom, ses quatre premiers arguments et son nombre d'arguments, puis testez-le.

Créez un script qui affiche le contenu d'un répertoire dont le chemin (relatif ou absolu) est passé en argument, puis testez-le.

Créez un script qui crée, dans le répertoire courant, un nouveau répertoire dont le nom est passé en argument, ainsi qu'un fichier dont le nom est également passé en argument à l'intérieur de ce nouveau répertoire ; puis, qui interdit à tous les utilisateurs l'accès à ce fichier en écriture et en exécution ; testez-le.

4 Structure conditionnelle (dans les scripts)

L'instruction `if` permet d'exécuter des instructions si une condition est vraie. Sa syntaxe rigoureuse est la suivante (`instruction`, `instruction1` et `instruction2` étant des suites d'instructions quelconques) :

```
if condition
then
    instruction1
fi
```

ou la suivante (afin d'effectuer une action que la condition soit vraie ou fausse) :

```
if condition
then
    instruction1
else
    instruction2
fi
```

Il existe plusieurs opérateurs permettant de vérifier des conditions (sur les fichiers), en voici quelques-uns (attention, les espaces entre la condition et les crochets sont obligatoires) :

- `-e` : vrai si le fichier existe ; exemple : `if [[ -e ./toto ]] …` ;
- `-f` : vrai si le fichier est « ordinaire » ; exemple : `if [[ -f ./tata ]] …` ;
- `-d` : vrai si le fichier est un répertoire ; exemple : `if [[ -d /bin ]] …` ;
- `-r` : vrai si le fichier est accessible en lecture ; exemple : `if [[ -r ./hello ]] …`.

Écrivez et testez le script suivant qui, si le premier argument est égal à zéro, affiche le deuxième argument, sinon, affiche le troisième argument ; testez-le (pour bien comprendre le fonctionnement de `if`) :

```
#!/bin/bash
if (( $1 == 0 ))
then
    echo $2
else
    echo $3
fi
```

Créez un script qui, si l'argument passé est un fichier existant, affiche son contenu, sinon, affiche le message d'erreur *« file does not exist »* ; testez-le.

5 Boucles (dans les scripts)

La boucle `for` permet d'exécuter une suite d'instructions avec une variable prenant à chaque nouvelle itération la valeur suivante d'une suite de valeurs.

Pour l'une de ses formes, le nom de la variable doit être écrit entre le `for` et le `in` alors que la suite de valeurs (séparée par des espaces) doit être placée entre le `in` et le `do`. Les instructions à répéter sont placées entre le `do` et le `done`. Par exemple (l'instruction `echo x = $x` étant donc exécuté une fois pour chaque valeur possible de `x`) :

```
#!/bin/bash
for x in un deux trois quatre
do
    echo x = $x
done
```

Ce script affiche donc à l'écran :

```
x = un
x = deux
x = trois
x = quatre
```

Écrivez et testez le script suivant qui affiche tous les fichiers du répertoire courant :

```
#!/bin/bash
for fichier in ./*
do
    echo "$fichier"
done
```

Écrivez et testez le script suivant qui affiche tous les arguments de la ligne de commande :

```
#!/bin/bash
for par in $*
do
    echo $par
done
```

6 Mise en pratique

Dans un script, il est possible de remplacer l'exécution d'une commande par son résultat (utile en particulier pour l'affichage) en l'entourant du caractère ` (à ne pas confondre avec le caractère '). Par exemple : `echo work directory: `ls``.

Écrivez un script qui, pour chaque fichier du répertoire courant, indique si c'est un répertoire ou non, puis testez-le.

7 Premier script de Sypher

Dans un répertoire d'accès public (que vous pouvez créer le cas échéant) de votre répertoire personnel, réalisez un script `ne_pas_executer.jpg` qui vous envoie un *e-mail* (et/ou écrire dans un fichier journal que vous dédiez à cet usage si la configuration du système ne le permet pas) précisant l'utilisateur ayant (honteusement) exécuté votre script.

Donnez-lui les droits d'exécution pour tous.

Essayez d'exécuter ou d'afficher le fichier équivalent d'un de vos camarades et constatez le résultat.

8 Pour aller plus loin

Écrivez un script qui reproduit le comportement de `ls -R` en utilisant uniquement les commandes `ls`, `cd` et `echo`.