Introduction aux systèmes d'exploitation (IS1)
TP n° 12 (optionnel) : scripts

Le but de ce TP est d'écrire de petits programmes utilitaires – appelés couramment *scripts* – en `bash` ; `bash` est en effet plus qu'un langage permettant d'exécuter des commandes, et possède les caractéristiques d'un langage de programmation (rudimentaire, certes) : on peut définir des variables, des fonctions, écrire des instructions conditionnelles, des boucles...

Un script contient des séquences de commandes telles que l'on pourrait les taper dans un terminal. Les commandes successives sont séparées par des retours à la ligne (ou des points virgules).

Tout fichier de script commence par une ligne permettant d'identifier le programme qui doit être utilisé pour l'exécuter – dans le cas d'un script `bash`, une ligne équivalente à *`#!/usr/local/bin/bash`*.

Pour qu'un utilisateur puisse exécuter un script, il doit posséder les droits en **exécution**, mais aussi en **lecture** sur ce script (c'est un cas particulier où la lecture est nécessaire à l'exécution). Pour exécuter un script, on peut taper directement dans le terminal une référence valide du fichier, ou taper la commande `bash` suivie d'un chemin du fichier.

Exercice 1 – *premier script*

Écrire un script `enterre_coffre.sh` qui crée dans le répertoire courant un répertoire `Cachette`, puis dans ce répertoire un fichier `coffre` contenant le texte `tout plein de pieces d'or`. `Cachette` et `coffre` devront être protégés en lecture, écriture et exécution.

Les variables Elles possèdent un *identificateur*, qui peut être n'importe quelle suite de caractères commençant par une lettre ou '_' et ne contenant que des lettres, des chiffres ou '_'. Elles n'ont qu'un type possible, chaîne de caractères (mais l'évaluation numérique est possible lorsqu'elle a du sens).
La déclaration et l'affectation se font simultanément par :

`var=`*valeur* ***sans espace avant ni après le signe '='.***

On accède à la valeur d'une variable `var` par `$var`.

Exercice 2 – *variables du SHELL*

On rappelle l'existence de variables sont déjà utilisées par le shell : `SHELL`, `PWD`, `USER`, `HOME`... Modifier `enterre_coffre.sh` pour qu'il affiche en plus le texte suivant :
`Bonjour` *nom d'utilisateur*,
`je viens de cacher un trésor dans ton répertoire` *nom du répertoire courant*.

Les paramètres Il est possible de passer aux scripts des paramètres sur la ligne de commande. Pour manipuler ces paramètres à l'intérieur du script, il existe plusieurs variables spéciales, en particulier :

- `$#` est le nombre de paramètres passés au script,
- `$0` est le nom du programme en cours d'exécution,
- pour chaque entier `i` entre 1 et `$#`, `$i` contient le i^e paramètre,
- `$@` contient la liste de tous les paramètres. Attention à toujours protéger cette variable avec des guillemets (`"$@"`) (à cause des espaces).

Exercice 3 – *paramètres d'un script*

1. Écrire un script `fouille_cachette.sh` prenant un paramètre, qui sera interprété comme le nom d'un répertoire dont il faudra lister le contenu.

2. Le script précédent échoue si l'accès au répertoire donné en paramètre est protégé. Ajouter les instructions nécessaires pour qu'il puisse lister le contenu du répertoire même dans ce cas. Tester ce script sur le répertoire `Cachette`, ensuite restaurer les droits d'origine sur `Cachette`.

Les tests La syntaxe des instructions conditionnelles est la suivante :

`if` *test*
`then` *commande*
`else` *commande*
`fi`

Les retours à la ligne sont obligatoires, la partie `else` est facultative.
Une syntaxe possible pour les tests est `[[` *expression* `]]`, où *expression* peut faire appel à tout un panel d'opérateurs unaires ou binaires pour tester si un fichier existe (`[[ -e` *fic* `]]`), si c'est un répertoire (`[[ -d` *fic* `]]`), si deux chaînes sont égales (`==`), si deux conditions sont vraies simultanément (`&&`)... Attention, les espaces sont obligatoires.
Plusieurs références sur les conditionnelles en bash sont disponibles sur internet. Par exemple : `http://www.tldp.org/LDP/abs/html/tests.html`

Exercice 4 – *premiers tests*

1. Modifier `fouille_cachette.sh` pour qu'il vérifie qu'il a bien reçu (exactement) un paramètre. Dans le cas contraire, le script affichera juste un message précisant à l'utilisateur le bon usage du script.

2. Modifier ensuite `fouille_cachette.sh` pour qu'il vérifie que ce paramètre est bien la référence d'un répertoire. Dans le cas contraire, le script affichera aussi un message adéquat (de préférence différent du précédent).

3. Ajouter les instructions nécessaires pour que le script teste si le répertoire est accessible avant, éventuellement, d'en modifier les droits.

4. Modifier enfin le script pour qu'il teste la présence d'un fichier `coffre` dans le répertoire. S'il en contient un, il devra en afficher le contenu. Sinon, un message annoncera l'échec de la fouille.

5. Faire en sorte que le script restaure lui-même les droits qu'il a modifiés.

Les boucles On peut définir un boucle de type « pour tout » de la manière suivante :

`for` *var* `in` *liste de valeurs*
`do` *commande*
`done`

La liste de valeurs peut être explicite, ou obtenue par expansion d'une chaîne comme `*`, ou contenue dans une variable, ou obtenue comme affichage résultant d'une commande ; dans ce dernier cas, la syntaxe est `$(`*cmd*`)` ou ‘*cmd*‘ (symbole *backquote*).

On dispose aussi d'une boucle « tant que » :

`while` *test*
`do` *commande*
`done`

Exercice 5 – *première boucle*

1. Écrire un script `compte_repertoires.sh` qui liste (uniquement) les sous-répertoires du répertoire courant à l'aide d'une boucle (même si vous savez le faire autrement).

2. Modifier ensuite ce script pour compter les sous-répertoires trouvés. Pour cela, on pourra utiliser l'évaluation numérique d'une expression avec `$((`*expression*`))`. En particulier, la commande `i=$((i+1))` incrémente la variable `i`. Pour plus de précisions, voir par exemple `http://www.tldp.org/LDP/abs/html/dblparens.html`.

Exercice 6 – *une meilleure cachette*

Modifier `compte_repertoires.sh` pour obtenir un script `cache_coffre.sh` qui se comporte comme `enterre_coffre.sh`, mais en commençant par tester si le répertoire courant possède au moins un sous-répertoire ; dans ce cas, il crée le sous-répertoire `Cachette` dans le premier sous-répertoire trouvé.

Exercice 7 – *un peu de hasard*

La fonction interne de `bash` `RANDOM` renvoie, à chaque appel, un entier pseudo-aléatoire (dans l'intervalle $[0, 32767]$). L'opération `%` (modulo) permet de se ramener à un autre intervalle si nécessaire.

À l'aide de cette fonction, modifier `cache_coffre.sh` pour que le sous-répertoire dans lequel `Cachette` est créé soit choisi au hasard parmi les sous-répertoires du répertoire courant. Pour cela, vous pouvez par exemple utiliser `RANDOM` pour tirer un nombre `k` aléatoire inférieur au nombre de sous-répertoires du répertoire courant, puis déterminer le k^e sous-répertoire parcouru par une boucle.

Exercice 8 – *toujours plus loin*

Modifier `cache_coffre.sh` pour qu'il crée `Cachette` le plus profondément possible dans l'arborescence (en choisissant toujours au hasard le sous-répertoire dans lequel il poursuit l'exploration). Il crée donc `Cachette` dans un répertoire qui n'a pas de sous-répertoires, pas forcément le plus profond dans l'absolu.

Pour cela, il est possible de boucler tant que le répertoire courant possède des sous-répertoires – et dans ce cas, en choisir un au hasard et s'y déplacer. Quand on arrive à un cul-de-sac, il est temps de créer `Cachette`.

Exercice 9 – *et maintenant, à la chasse au trésor*

Écrire un script `cherche_coffre.sh` qui, étant donné un paramètre, fouille toute la sous-arborescence du répertoire dont la référence est passée en paramètre à la recherche d'un fichier de nom `coffre`. Le script ne devra naturellement pas se laisser rebuter par d'éventuels répertoires à accès restreint. Pour chaque fichier `coffre` trouvé, le script affichera sa référence absolue ainsi que son contenu.

Pour cela, il faut écrire un script ***récursif***, c'est-à-dire qui s'invoque lui-même (avec un autre paramètre).

Note : le but de cet exercice est de vous faire simuler une version de base de la commande `find`, il est donc naturellement interdit d'utiliser cette commande pour votre solution...