TP : Shell Scripts

1 Remarque générale

Lorsque vous cherchez des informations sur Internet, n'oubliez pas que langage de shell script que nous avons vu correspond à bash. La plupart des ressources disponibles se rapporte à ce langage néanmoins il est possible que vous tombiez de temps en temps sur un autre langage de shell script.

2 Mise en jambe

Objectifs : savoir écrire et exécuter un shells script minimal. savoir utiliser des variables et exécutez des commandes.

Cours : slides 61 à 71.

Hello World : écrivez un script qui affiche le message "Hello World". Exécutez le et vérifiez que le résultat est conforme.

Paramètre : écrivez un script qui prend un paramètre et affiche la valeur de se paramètre à l'écran. Exemple la commande `./monscript toto` doit afficher `parametre : toto`. Exécutez le et vérifiez que le résultat est conforme.

Somme de deux nombres : écrivez un script qui calcule la somme de deux nombres passés en paramètres et l'affiche à l'écran.

3 Avec des si

Objectifs : savoir utiliser la structure conditionnelle et la commande `test` en shell script.

Cours : slides 72 à 78

Max : Ecrivez un script qui prend deux nombre en paramètre et affiche le plus grand des deux.

Utilisation correcte : Reprenez le script de l'exercice *Paramètre* ci-dessus. Que se passe-t-il si vous lancer le script sans indiqué de paramètre ? Ajoutez des instructions afin que le script affiche un message d'erreur si l'utilisateur ne passe pas 1 et 1 seul paramètre au script.

Nombre ou pas nombre : écrivez un petit script qui retourne la valeur 0 si le premier paramètre reçu représente un nombre entier et 1 sinon (utilisez grep). Astuce, parfois on ne s'intéresse pas à la sortie d'une commande mais uniquement à son code de retour. On peut alors rediriger la sortie de cette commande vers le fichier `/dev/null` qui agit comme un trou noir (tout ce qu'on met dedans est perdu à jamais).

Nombre ou pas nombre 2 : écrivez un petit script qui demande un nombre a l'utilisateur et vérifie si les caractères donnés par l'utilisateur correspondent bien à un nombre.

4 Boucles

Objectifs : savoir utiliser la structure conditionnelle et la commande `test` en shell script.

Cours : slides 79 à 86

Liste des paramètres : écrivez un script qui affiche tous les paramètres qu'il a reçu.

Somme de n nombres : écrivez un script qui calcule la somme de tous les nombres passés en paramètres.

Nombre impairs : écrivez un script qui affiche la liste des n premiers nombres impaires (à partir de 1), n étant un paramètre optionnel qui sera fixé à 10 si il n'est pas précisé par l'utilisateur.

Table de multiplication : écrivez un script nommé `table` permettant d'afficher des tables de multiplications. `table 5 10` aura pour résultat l'affichage :

```
0 x 5 = 0
1 x 5 = 5
2 x 5 = 10
3 x 5 = 15
4 x 5 = 20
5 x 5 = 25
6 x 5 = 30
7 x 5 = 35
8 x 5 = 40
9 x 5 = 45
10 x 5 = 50
```

Fonction application : écrivez un script `applique.sh` qui prend en paramètre le nom d'une commande et qui appelle successivement cette commande avec chacun des fichiers contenu dans le répertoire courant. (Par exemple si le répertoire courant contient 2 fichiers `fich1.txt` et `fich2.txt`, l'exécution de `./applique.sh wc` génèrera les appelles : `wc fich1.txt` et `wc fich2.txt`.)

Qui sont les plus grands ? : écrivez un script qui affiche la liste des utilisateurs dont l'UID est plus grand qu'un nombre donné en paramètre (cette information se trouve dans le fichier `/etc/passwd`)

Pointage et gestion des droits : (optionnel) écrivez un script `pointer.sh` qui ajoute une ligne contenant : le login de la personne exécutant le script, la date et l'heure, à la fin du fichier `pointage.sh`. Imaginons que ce script doit être appelé par les employés d'une entreprise lorsqu'ils arrivent ou quittent leur poste. Un problème se pose : comment faire pour que le script `pointer.sh` puisse modifier le fichier `pointage.sh` sans que les utilisateurs n'en aient le droit ? Ceci est a priori impossible car lorsqu'un utilisateur lance un script, celui-ci s'exécute avec les droits de l'utilisateur qui le lance. Il existe néanmoins une astuce pour parvenir à cette fin : cherchez des informations sur le mot `Setuid` sur Internet pour trouver la solution. Vérifier avec un de vos collègues que cette solution fonctionne bien.

5 Un peu plus complexe

Objectifs : perfectionnement et réutilisation de l'ensemble des notions vues.

Comptage : écrivez un script qui prend en paramètre un nom de fichier et qui affiche ce fichier ligne par ligne, en ajoutant devant chaque ligne : le numéro de la ligne et le nombre de mots qu'elle contient.

ls récursif : écrivez un script qui affiche la liste de tous les fichiers contenus dans le répertoire courant, ses sous-répertoires, les sous-sous-répertoires, etc.

Archivage : écrivez un script qui réalise les actions suivantes :
- Il créé le dossier `~/archive/YYYY-MM-DD/` ou `YYYY-MM-DD-hh:mm` est replacé par la date et l'heure actuelle (si le dossier `~/archive/` n'existe pas, il est créé dans la foulée).
- Il déplace tous les fichiers du répertoire courant (ainsi que ses sous-dossiers) dans ce nouveau dossier.

Ajoutez ensuite une option pour demander au script de mettre l'ensemble des fichiers à déplacer dans une archive compressée `.tgz` (utilitaire `tar`).

Un script de pro : (optionnel) ouvrez le script système `/etc/init.d/killprocs` qui est appelé lors de l'extinction du système. Essayez de comprendre comment il fonctionne.

6 Problème (presque) concret : automatisation

Objectif : savoir combiner les compétences acquises pou résoudre un problème concret, utiliser les fonctions.

Cours : slides 87 à 89.

Cet exercice constitue un petit problème classique : comment automatiser une tâche répétitive tout en gérant proprement les exceptions.

Dans notre problème, nous avons un ensemble de données qui sont stockées dans le répertoire `~perretb/public_html/I3FM/Unix/subwork/`. Ces données sont stockées dans un peu plus de 1200 fichiers indépendants. Nous souhaitons appliquer le programme qui a été développé par l'équipe sur chacun de ces fichiers. Ce programme appelé `doSomething` est disponible ici : `~perretb/public_html/I3FM/Unix/doSomething`.

`doSomething` n'est pas encore parfaitement fiable (c'est la beta 0.92) et il arrive qu'il soit incapable de faire ce qu'il est sensé faire sur certaines données. Néanmoins les ingénieurs ont trouvé un compromis, `doSomething` peut fonctionner selon 3 niveaux d'optimisation. Dans le niveau le plus haut il est très rapide mais plante souvent, dans le niveau le plus bas il est plutôt lent mais plante rarement. La stratégie retenue est alors la suivante : pour chaque fichier on essaye de le traiter avec le niveau 1 (optimisation maximale). Si cela ne fonctionne pas on réessaye avec le niveau 2. Si cela ne fonctionne toujours pas, on passe au niveau 3. Finalement, si le niveau 3 ne fonctionne pas non plus, il faut enregistrer le nom du fichier fautif pour que l'équipe de débogage se penche dessus.

Concrètement, le programme `doSomething` s'utilise de la façon suivante. Il prend obligatoirement 2 paramètres :
- le premier indique le niveau d'optimisation : -o1 (optimisation élevée) -o2 (optimisation moyenne) -o3 (optimisation faible)
- le nom du fichier à traiter.

Si le traitement se déroule correctement, le résultat est écrit sur la sortie standard et le code de retour 0 (succès) est renvoyé. Si une erreur survient (erreur dans la façon d'utiliser le programme ou impossibilité de traiter le fichier indiqué au niveau d'optimisation voulu), le descriptif de l'erreur est écrit sur la sortie d'erreur et le code de retour 1 (échec) est renvoyé.

Comme il est impensable de devoir gérer cette procédure manuellement sur les 1223 fichiers à traiter, vous allez devoir écrire un script qui automatise le processus et enregistre les résultats au fur et à mesure.

1) Commencez par écrire une fonction qui prend trois arguments :
– Le nom du fichier à traiter
– Le nom du fichier de résultats
– Le nom du fichier de log des erreurs
Cette fonction applique la stratégie de traitement décrite sur le fichier d'entrée. Le résultat (si une des niveaux d'optimisation fonctionne) doit être inscrit à la suite du fichier de résultats. Si le niveau d'optimisation 3 échoue, le message d'erreur produit par le programme doit être enregistré dans le log d'erreur (on ne s'intéresse pas au message d'erreur des niveaux 1 et 2).

2) Écrivez un seconde fonction qui prend les mêmes arguments que la fonction précédente et qui détermine si le fichier a déjà été traité. Elle doit retourner :
– 0 si le fichier n'apparait ni dans le fichier de résultat ni dans le fichier de log d'erreur
– 1 dans le cas contraire
3) Écrivez le script basé sur ces deux fonctions qui automatise le traitement de tous les fichiers contenu dans le répertoire dont il reçoit le nom en paramètre. (Faites une boucle sur l'ensemble des fichier du répertoire, déterminez alors si une action est à réaliser grâce à la fonction 2) et si c'est le cas utilisez la fonction 1))

7 Jeux du pendu

Objectif : s'amuser en shell script.

Le pendu est un jeu consistant à trouver un mot en devinant quelles sont les lettres qui le composent. Au début du jeu, les caractères du mot à trouver sont cachés et le joueur ne peut voir que le nombre total de caractères qui le compose. A chaque étape du jeu, le joueur propose un caractère. Si ce caractère apparait dans le mot, toutes les lettres correspondant à ce caractère sont découvertes (montrées au joueur). Si la lettre n'apparait pas dans le mot, le joueur perd une vie. Le joueur doit découvrir toutes les lettres du mot avant de perdre toutes ses vies.

Le but de cet exercice est de programmer le jeux du pendu en shell script. Ce type de jeu est bien adapté à ce langage car il s'agit essentiellement de manipuler des chaines de caractères.

Pour commencer, récupérez le fichier `~perretb/public_html/I3FM/Unix/liste.txt` qui contient une liste de mots.

Pour coder le jeux du pendu nous allons suivre le schéma suivant. Le programme va maintenir une liste des caractères cachés. Au début cette liste contient toutes les lettres de l'alphabet (`caches="abcd...wxyz"`).
A partir de cette liste et du mot a trouver, nous allons définir diverses fonctions :
– une fonction qui remplace les lettres cachées du mot à découvrir par un autre caractère (par exemple un tiret -)
– une fonction qui teste si le mot contient un caractère donné (pour déterminer si le joueur perd une vie)
– une fonction qui retire un caractère de la liste des caractères cachés
– une fonction qui teste si le mot ne contient aucune des lettres cachées (dans ce cas le joueur gagne)
– et finalement une fonction qui tire un mot au hasard dans la liste des mots.
On va écrire ces différentes étapes, l'une après l'autre. Testez chaque fonction après l'avoir écrite et assurez vous qu'elle fonctionne correctement dans des cas variés avant de passer à la fonction suivante.

1) Écrivez une fonction `motAuHasard` qui affiche un mot tiré au hasard dans le fichier `liste.txt` (la variable RANDOM permet de générer des nombres aléatoires. Par exemple `$(( RANDOM % nombdredelignes ))` calcule un nombre au hasard compris entre 0 et `nombrelignes`

2) Écrivez une fonction `afficheMot` qui prend deux arguments : le mot à trouver et la liste des lettres cachées. Cette fonction remplace dans le mot à trouver toutes les lettres de la liste des lettres cachées par des tirets et affiche le résultat.

3) Écrivez un fonction `decouvre` qui prend deux arguments : la liste des caractères cachés et un caractère. Cette fonction supprime le caractère de la liste des caractères cachés et affiche la nouvelle liste.

4) Écrivez une fonction `testPresence` qui prend deux paramètres : le mot à découvrir et un caractère. Cette fonction retourne 0 si le caractère est présent dans le mot à découvrir et 1 sinon.

5) Écrivez une fonction `testGagne` qui prend deux arguments : le mot à trouver et la liste des lettres cachées. Cette fonction retourne 0 si le mot ne contient aucun caractère caché et 1 sinon.

6) Vous pouvez maintenant écrire la partie principale du jeu en combinant les fonctions précédentes. Si vous êtes perdus, vous pouvez suivre le pseudo code donné sur la page suivante mais essayez d'abord de trouver par vous-même.

```
vie := 7
caches := "abcdefghijklmnopqrstuvwxyz"
lemot := motAuHasard()
tant que ( vie > 0 )
faire
  afficheMot()
  demander un caractère c à l'utilisateur
  caches := decouvre(caches,c)
  si testPresence(mot,c) == 0
    si testGagne(mot,caches)
      Afficher("Gagné!")
      Quitter
    fin si
  sinon
    vie := vie - 1
  fin si
fin tant que
Afficher("perdu!")
```