Pratique, Installation et Utilisation des machines
TP 9

02 décembre 2016

Exo 1 - Initiation aux scripts bash : petit script de recherche

Pour pouvoir tester votre futur script Shell, créez dans le répertoire courant les fichiers (vides) "file 1", "file 2", "text" et "memo", puis créez les répertoires "rep 1", "rep 2", "tmp" et "log".

Exo 1.1 - Récupérer des arguments sur la ligne de commande

- Créez un nouveau fichier `recherche.sh`, ouvrez le avec un éditeur de code (vi ou emacs) et commencez par indiquer sur la première ligne la mention `#! /bin/bash` pour indiquer qu'il s'agit d'un script Shell Bash.
- Ensuite utilisez 3 fois la commandes 'echo' pour afficher les messages "La commande se nomme ...", "Vous l'avez lancée avec ... arguments", "Ces arguments sont ...".
  Pour rappel, la variable `$0` *contient le nom du script, la variable* `$#` *contient le nombre d'arguments récupérés sur la ligne de commande, et la variable* `$*` *contient ses arguments.*
- Enregistrez le script, fermez votre éditeur et lancer le script depuis le Shell. Essayez de le lancer avec différents arguments pour vérifier son fonctionnement.

Exo 1.2 - Faire un script robuste et respecter les conventions

Un utilisateur étourdit pourrait lancer votre script sans lui donner d'arguments...

- Ouvrez à nouveau votre programme dans vi ou emacs et insérez, avant les commandes d'affichage, un test qui vérifiera que l'utilisateur à donné au moins 1 argument à la commande.
  Pour rappel, la variable `$#` *contient le nombre d'arguments et l'opérateur* `-lt` *permet de vérifier si un nombre est inférieur à une valeur. Un test se fait avec :* `if [ ... ]; then ... fi`
- Si le test passe, affichez un message pour avertir l'utilisateur de la bonne manière d'utiliser votre commande avec un message du type : "usage : nomduscript keyword"
  Pour rappel, la variable `$0` *contient le nom du script.*
- Keyword est un mot à remplacer par la valeur appropriée. Il doit donc être souligné pour respecter les conventions du manuel. Pour souligner le mot, encadrez le avec les instructions `\033[4m` et `\033[0m`
- Une fois ce message affiché, ajoutez l'instruction `exit 1` pour arrêter le programme en signalant qu'un problème est survenu.

Exo 1.3 - La boucle de recherche

- Ajouter à votre script une boucle `for` qui examinera un a un les fichiers du répertoire courant.
  Pour rappel, une boucle `for` *s'écrit avec la syntaxe suivante :* `for var in ... ; do ... done`
  Pour obtenir la liste des fichiers du répertoire courant vous pouvez utiliser le joker `*`.
- Dans la boucle, ajouter un test qui vérifiera à chaque tour si le fichier courant correspond au premier argument passé à la commande par l'utilisateur.
  Pour rappel, il faut utiliser `$` *pour récupérer la valeur d'une variable et* `$1` *permet de récupérer la valeur du premier argument. Pour comparer deux chaînes on peut utiliser l'opérateur* ==.

- Dans le test affichez un message du type "trouvé !" pour signaler que le fichier recherché à été identifié.
- Vérifiez que votre programme fonctionne bien, même quand les noms comportent des espaces.

Exo 1.4 - Identifier le type d'un fichier

Nous allons ajouter un test pour vérifier et afficher le type du fichier trouvé.

- En plus de tester si le fichier correspond à la saisie de l'utilisateur, ajoutez un test pour vérifier le type du fichier et afficher un message du type "trouvé ! c'est un fichier !" ou "trouvé ! c'est un répertoire !"
  Pour rappel, `-f` *permet de tester si il s'agit d'un fichier ordinaire et* `-d` *d'un répertoire. Pour faire un double test on peut utiliser la syntaxe :* `[ ... ] && [ ... ]`

Exo 1.5 - Petit message supplémentaire

Histoire d'embellir un peu notre programme, nous allons afficher un message "5 4 3 2 1 go !" avant de lancer la recherche.

- Ajoutez une autre boucle for pour afficher le compte à rebours. Pour obtenir une liste de nombre de 1 à 10 par exemple vous pouvez utiliser la syntaxe suivante : `{1..10}`. Pour une liste décroissante il suffit d'inverser les limites.
- Après la boucle ajoutez une commande pour afficher le message "go !" et conclure le compte à rebours.

Exemple d'exécution du script final :

```
$ ./recherche.sh
usage : ./recherche.sh keyword
$ echo $?        # pour tester la valeur de retour de mon script
1
$ ./recherche.sh "rep 2"
La commande se nomme ./recherche.sh
Vous l'avez lancée avec 1 argument(s)
Ces arguments sont : rep 2
10 9 8 7 6 5 4 3 2 1 go !
trouvé ! c'est un répertoire !
fin de la recherche
```

Exo 2 - Initiation aux scripts bash : un petit jeu

Nous allons maintenant créer un jeu du type "devine un nombre entre 1 et 100".

- Créez un nouveau script nommé `jeu.sh` et placez `#! /bin/bash` sur la première ligne.

Exo 2.1 - Tirer un nombre au hasard

Le Shell bash fournit une variable spéciale nommée `$RANDOM` (seulement valable sous bash ou ksh) qui permet de récupérer un nombre aléatoire entre 0 et 32767. Chaque fois que l'on récupère le contenu de `RANDOM` avec `$RANDOM`, le nombre obtenu change.
Pour obtenir un nombre entre 1 et 100 nous allons utiliser le modulo pour obtenir un nombre entre 0 et 99 puis lui ajouter 1 : `$RANDOM % 100 + 1`
Pour que le Shell comprenne qu'il doit faire le calcul il faut passer cette expression à la commande 'expr' : `expr $RANDOM % 100 + 1`
Pour récupérer le résultat de cette commande on peut utiliser une de ces deux syntaxe :
`$(expr $RANDOM % 100 + 1)` ou `` `expr $RANDOM % 100 + 1` ``

- Dans votre script, ajouter la commande pour tirer un nombre au hasard et placez le résultat dans une variable `NOMBRE`.
- Pour vérifier le bon fonctionnement de votre programme, vous pouvez ajouter momentanément un affichage du nombre avec `echo $NOMBRE'`.

Exo 2.2 - Récupérer une saisie de l'utilisateur

- La commande 'read' permet de récupérer une saisie de l'utilisateur. Cherchez dans le manuel comment utiliser cette commande et comment ajouter un message du type "devine un nombre entre 1 et 100 : " pour inviter l'utilisateur à saisir un nombre.

Exo 2.3 - Boucle principale

Maintenant il va falloir comparer le nombre tiré au hasard avec le nombre proposé par l'utilisateur. Nous allons utiliser une boucle while et deux tests. Le code doit avoir cette structure :

```
while NOMBRE != SAISIE ; do
    if SAISIE < NOMBRE ; then
        affiche "plus grand" et demande un autre nombre
    elif SAISIE > NOMBRE ; then
        affiche "plus petit" et demande un autre nombre
    fi
done
```

Pour rappel les tests sont à placer entre crochets. Un symbole `!` permet de nier un test. Pour comparer des nombres on peut utiliser `-lt` (plus petit que), `-gt` (plus grand que), `-eq` (égal à).

- Ajoutez la boucle principale à votre programme et testez le.
- N'oubliez pas que l'on sort de la boucle principale si on a bien deviné le nombre. Ajoutez l'affichage d'un message pour féliciter le joueur !

Exo 2.4 - Utiliser un alias

Pour aider un peu votre joueur vous pourriez lui fournir des astuces !

- Créez un fichier caché nommé ".astuce" et placez dedans juste ce texte indiquant l'astuce pour ce jeu : "En proposant à chaque fois un nombre au milieu de l'intervalle donné, tu divises chaque fois la difficulté par 2 !"
- Maintenant, on voudrait pouvoir utiliser la commande 'astuce' pour afficher ce message. Au début de votre script `jeu.sh`, ajoutez la commande suivante qui permet de créer un alias : `alias astuce="cat .astuce"`
- Ajoutez un message affiché avec `echo` pour prévenir votre joueur qu'il peut utiliser la commande `astuce`.
- Lancer votre script avec `source jeu.sh` pour que l'alias soit bien créé dans le contexte de votre console. Ensuite, entre deux parties de votre jeu, vous pourrez taper la commande `astuce` et vous verrez le message (car c'est la commande `cat .astuce` qui est déclenchée par l'alias).

Exo 3 - Scripts Bash en pratique : modifier son environnement

Avant de modifier votre fichier `.bashrc` pensez à faire une sauvegarde du fichier originel.

- Faites une copie de votre fichier `.bashrc` en `.bashrc_bak` dans votre répertoire personnel
- Modifier votre `.bashrc` afin que le prompt contienne `[username]@[repertoire courant]:[date]` avec au moins deux couleurs.
- Modifier votre `.bashrc` afin que la variable `PATH` contienne le répertoire dans lequel se trouve `jeu.sh`

- Créer un alias dans votre `.bashrc` afin de lancer votre jeu ainsi :
  `$ devinette`