Défi Big Data

Scripts Shell

Xavier Serpaggi

Mars 2018

Travail à rendre

Vous écrirez les réponses aux questions ci-dessous dans un fichier texte. Aucun autre format n'est attendu.

Ce fichier texte sera déposé sur Campus, même partiel, à la fin de la séance.

Pour la première étape, les réponses seront les commandes ou les scripts vous ayant permis de répondre aux questions. Comme toujours, il ne faut mettre que les commandes/scripts, pas leurs résultats.

Arrivée

Trucs & astuces
Quelques trucs et astuces supplémentaires pour l'utilisation du shell :

- Il est possible d'obtenir la liste de toutes les commandes que l'on a précédemment tapées avec `'history'`
- On peut relancer n'importe quelle commande en utilisant `'!foo'` où `foo` correspond au numéro dans l'historique.
- `'!!'` relance la dernière commande et `'!bar'` relance la dernière commande qui commence par `bar`.
- Control+A permet d'aller directement au début de la ligne de commande.
- Control+E permet d'aller directement à la fin de la ligne de commande.

Étape 1 : un tout petit peu plus sur les commandes

■ *Le shell est un véritable langage de programmation. Il possède donc des outils permettant de trouver les éventuelles erreurs que nous avons commises. Néanmoins, le programmeur a un devoir de rigueur et doit s'assurer que son script va bien se dérouler.*

Sur Campus vous trouverez une proposition de correction pour le script écrit lors de la dernière séance. Téléchargez-le, c'est avec lui que nous allons travailler aujourd'hui.

Il est possible de faire du *debug* dans le shell ou dans un script en positionnant un paramètre de Bash avec `'set -x'`. Pour arrêter le mode *debug* il suffit de faire `'set +x'`.

Enfin, il est possible de tester le code de retour de la dernière commande exécutée avec la variable spéciale `$?`. Une commande dont le déroulement s'est bien passé renvoie en général le code 0 (à vérifier dans sa page de manuel).

Exercices

1. Renommez le script téléchargé en `indexeur.sh`.
2. Avec `'grep'` cherchez la chaîne de caractères `TMP` dans `indexeur.sh`, puis affichez le code de retour avec `'echo'`.
3. Cherchez à présent la chaîne de caractères `zorglub` dans ce même fichier, puis vérifiez à nouveau le code de retour.
4. Sans rien faire d'autre, affichez encore le code de retour. À quelle commande correspond-t-il ?
5. Essayez de copier le fichier `indexeur.sh` dans le répertoire `/` (répertoire racine) en redirigeant la sortie standard et la sortie d'erreur vers le fichier spécial `/dev/null` [1] pour qu'aucun affichage ne soit produit, puis affichez le code de retour.

Étape 2 : scripts, arguments et robustesse

■ *Nous continuons et enrichissons le script obtenu lors de la dernière séance. L'objectif est de le rendre le plus robuste possible tout en le rendant plus souple dans ses possibilités d'exécution.*

Les scripts, tout comme les programmes écrits dans d'autres langages des programmation, peuvent prendre des arguments, mais aussi des options sur la ligne de commandes. C'est la commande `'getopts'`, interne à Bash, qui s'occupe de ça.

Il est possible de déclarer des fonctions pour des morceaux de code que l'on réutilise plusieurs fois. Ces fonctions peuvent avoir des arguments, comme les scripts.

Les commandes `'basename'` et `'dirname'` permettent d'extraire les différentes parties d'un chemin.

Les variables peuvent avoir des valeurs par défaut ou de substitution avec les écritures `${VAR=`*`default`*`}` et `${VAR-`*`substitution`*`}` respectivement.

Exercices

1. Dans le fichier `indexeur.sh`, complétez tous les commentaires vides (lignes 51 à 60) en expliquant à quoi correspond chaque étape. La ligne 53 est plus difficile à comprendre.
2. Modifiez le code de la fonction `die`. Elle doit prendre deux paramètres optionnels [2] : un message et un code de retour. La fonction devra afficher le message contenu dans le premier paramètre s'il existe ou `Erreur` sinon, puis devra mettre fin au script avec le code de retour passé en second paramètre s'il existe ou 1 sinon.
3. Modifiez le code de la fonction `usage` qui doit afficher un message d'aide puis terminer le programme avec le code de retour 1. Le message d'aide sera formaté comme ceux donnés par les commandes `'grep'`, `'cut'` ou `'sed'` quand on leur passe l'option `--help`.
4. Nous voulons à présent que notre script puisse prendre des options qui vont modifier son comportement :
   `-d` pour activer le mode debug ;
   `-h` pour afficher le message d'aide ;
   `-f` pour traiter l'argument comme un fichier local et donc, ne rien télécharger.
   En vous aidant de la page de manuel de Bash et du mémo de programmation Shell de Christophe Blaess [3], mettez en place ce comportement. De plus, toute autre option devra entrainer l'affichage du message d'aide.
5. Modifiez la fonction `usage` pour qu'elle affiche les options disponibles ainsi que leurs effets.

1. `cp indexeur.sh / > /dev/null 2>&1`
2. Lisez la section `Parameter Expansion` de la page de manuel de Bash pour comprendre l'utilisation des valeurs par défaut et des valeurs de substitution des variables.
3. https://www.blaess.fr/christophe/developpements/aides-memoires/

6. Modifiez votre script pour mettre en œuvre toutes les fonctionnalités ci-dessus.
7. À présent que les fonctionnalités sont figées et qu'elles sont mises en œuvre, nous devons rendre le script robuste. C'est à dire qu'il devra être insensible ou presque à des comportements erronés, à de mauvais paramètres ou à des paramètres mal formés, ... Modifiez votre script pour qu'il se comporte correctement même dans les situations suivantes ('W' signifie qu'il n'y aura qu'un avertissement ou une mesure de contournement sans que le script ne s'arrête, 'E' signifie qu'il y aura un message d'erreur et que le script s'arrêtera) :
   — Le fichier n'a pas besoin d'être transcodé. À vérifier principalement dans le cas où le fichier est local (W).
   — Le transcodage du fichier ne se passe pas bien (E).
   — Lors de la manipulation des fichiers intermédiaires, les opérations ne se passent pas bien (E).
   — Les marqueurs `DEBUT DE FICHIER` et/ou `TABLE DES MATIÈRES` ne sont pas présents (W).
   — Le marqueur `TABLE DES MATIÈRES` n'est pas présent, mais `FIN DU FICHIER` est présent (W).

   Le script comporte déjà quelques lignes de détections d'erreurs grossières (il n'y a pas d'argument sur la ligne de commande, le protocole du fichier à télécharger n'est pas le bon). Inspirez-vous de ça, mais allez plus loin. Vous pouvez bien entendu modifier ces deux tests si vous avez de meilleures idées.

Étape 3 : un peu de calcul

■ *Dans cette dernière étape un peu plus compliquée, nous allons traiter du texte et aborder les capacités de calcul de l'interprète de commande ainsi que ses limites. Nous découvrirons comment pallier cela avec des outils annexes.*

Il est possible de faire des calculs simples avec la commande interne `'let'` ou avec la construction `'(( ... ))'` de Bash.

La commande `'uniq'` possède une option permettant de connaître le nombre d'occurrences d'un mot.

`'awk'` est une commande qui permet de faire des calculs et d'autres tâches complexes sur des fichiers textes. C'est en fait plus qu'une commande, c'est un langage de programmation complet, dérivé du C [4].

Il est possible de faire des transformations sur les lignes d'un fichier avec la commande `'sed'`, le mécanisme d'expressions régulières et les références arrières.

Exercices

Les questions préfixées de (*) sont d'autant plus difficiles qu'il y a un nombre d'étoiles important.

Pour chacune des questions, vous écrirez une commande (ou enchaînement de commandes) ou un petit script qui traite le fichier téléchargé et nettoyé par le script des étapes précédentes.

1. N'affichez que les mots dont la longueur est supérieure ou égale à une valeur donnée par la variable `$LMOT`.
2. (*) Donnez, en face de chaque mot, son nombre d'occurrences dans le fichier. Le format résultant devra être du CSV [5] (les valeurs seront séparées par des virgules). La sortie pourra alors être lue facilement par un tableur ou traitée par d'autres commandes.
3. Affichez, à la fin de l'index, la somme de toutes les occurrences. Vous adopterez le même format de sortie qu'à la question précédente.
4. (*) Donnez la liste des `$NMOTS` mots les plus courants dans le fichier, triés par ordre décroissant d'occurrences. La variable `$NMOTS` contient un nombre entier positif. Vous adopterez le même format de sortie qu'à la question précédente.

5. https://en.wikipedia.org/wiki/Comma-separated_values

```
*-------------+------*
| mot         | nb   |
|-------------|------|
| de          | 2869 |
| le          | 1887 |
| la          | 1609 |
| et          | 1596 |
| il          | 1420 |
| que         | 681  |
...
| abaissant   | 1    |
| abaissait   | 1    |
*-------------+------*
```

FIGURE 1 – format d'affichage demandé pour la question 6.

5. Donnez la longueur moyenne de tous les mots.
6. (**) Reprenez les questions 2 et 4 et modifiez le format d'affichage pour qu'il ressemble à celui présenté sur la figure 1.
7. (***) Écrivez un script qui regroupe toute ces fonctionnalités et qui permette de les déclencher au travers d'options.