Programmation Avancée

TP n°2: Terminal et fichiers

Simon Forest

21 janvier 2021

Exercice 1 : Commandes de base

Dans cet exercice, on se familiarise avec le terminal Linux (la plupart des commandes doivent aussi fonctionner sous MacOS, mais à vérifier !). Ouvrez un terminal et faites les questions suivantes.

1. La commande `pwd` permet de connaître le dossier courant. L'exécuter, en écrivant `pwd` puis en appuyant sur 'Entrée'. Quel est le chemin affiché ?
2. La commande `ls` permet de lister le contenu d'un dossier. L'exécuter. Quelle est l'information affichée ? Est-ce cohérent par rapport au dossier dans lequel se situe actuellement le terminal.
3. Comme la plupart des commandes `ls` peut prendre des options. Notamment `-l` pour afficher plus d'informations sur les fichiers listés et `-a` pour afficher les fichiers cachés. Exécuter la commande `ls` avec ces options en écrivant `ls -l -a` puis 'Entrée' dans le terminal. Qu'est-ce qui est affiché en plus ? Quelle est *a priori* l'interprétation à donner aux différentes colonnes ?
4. La commande `cd` permet de changer de répertoire (« cd » = « Change Directory »). Elle prend un argument, qui est le nom du dossier que l'on souhaite visiter. En utilisant `ls` pour voir les dossiers contenus dans le dossier courant, utiliser `cd` pour aller dans ces dossiers. Faire `pwd` et `ls` pour s'assurer que le terminal s'est bien déplacé vers le dossier voulu. Pour aller au dossier « parent », il faut exécuter `cd ..` (par convention sous Linux, `.` représente le dossier courant et `..` le dossier parent). On peut revenir au dossier « précédent » en exécutant `cd -`.
5. La commande `mkdir` permet de créer des dossiers. Elle prend en argument le nom du dossier à créer. L'utiliser pour créer un dossier s'appelant `mon_dossier` en exécutant la commande `mkdir mon_dossier`.
6. La commande `touch` permet de créer des fichiers. Elle prend en argument le nom du fichier à créer. L'utiliser pour créer deux fichiers `f.txt` et `g.txt` dans `mon_dossier`. On utilisera pour cela les deux méthodes suivantes :
   - se déplacer dans `mon_dossier` avec `cd`, puis créer le fichier avec `touch` ;
   - ou créer directement les fichiers `f.txt` et `g.txt` depuis le dossier qui contient `mon_dossier` en spécifiant un *chemin* à `touch` : `mon_dossier/f.txt` ou `mon_dossier/g.txt`.
7. La commande `nano` permet d'éditer un fichier en ligne de commande. L'utiliser pour éditer `f.txt` et `g.txt` pour y mettre respectivement une liste de fruits et de légumes (un par ligne).
8. La commande `grep` permet de faire des recherches dans des fichiers. Elle prend par défaut deux arguments dans cet ordre : la séquence de caractères que l'on cherche et le fichier dans lequel on souhaite chercher. Par exemple, exécuter `grep rais f.txt` (dans le bon dossier !) pour chercher toutes les lignes contenant « rais ». En particulier, si `f.txt` contient le mot « raisin », une ligne contenant ce mot devrait s'afficher. Pour afficher de plus la ligne où a été trouvé la requête, on peut rajouter l'option `-n` à la commande (essayer !).

9. La commande `cat` permet d'afficher le contenu d'un fichier. L'argument à lui donner est le nom du fichier à afficher. Exécuter `cat f.txt` pour afficher le contenu de `f.txt` créé plus haut.

10. La commande `cp` permet de copier un fichier ou dossier. Elle prend deux arguments qui sont le nom du fichier à copier et le nom de la copie. Exécuter `cp f.txt h.txt` pour copier `f.txt` dans un fichier `h.txt`. Utiliser `cat` sur `h.txt` pour vérifier que le contenu de `f.txt` a bien été copié.

11. La commande `man` permet de consulter la documentation associée à une commande ou fonction. Exécuter `man cp` pour avoir la documentation de la commande `cp`. Rechercher dans cette documentation le rôle de l'option `-r` pour `cp`.

Exercice 2 : Programmes avec arguments

On a vu avec l'exercice précédent que les programmes peuvent être paramétrés par des arguments donnés par un terminal. Nous allons voir comment faire de même dans les programmes que l'on crée.

1. Créer un fichier `main.c` avec une fonction `main` qui, au lieu de prendre 0 arguments comme d'habitude, en prend deux : `int main(int argc, char** argv)`. Mettre `return 0;` comme unique contenu de la fonction et vérifier que le fichier compile encore.

2. Utiliser `printf` pour afficher la valeur de `argc`. Quelle est la valeur affichée ?

3. Maintenant, exécuter le programme depuis un terminal en ajoutant des arguments : si votre programme s'appelle `prog`, exécuter par exemple `./prog a b c`. Quelle est la valeur de `argc` en fonction du nombre d'arguments écrits ?

En fait, `argv` peut se voir comme un tableau de `char*`, c'est-à-dire un tableau de chaînes de caractères. Les différents éléments du tableau stockent les arguments passés au programme. La variable `argc` précise alors la taille de ce tableau.

4. En utilisant une boucle avec `printf`, afficher les chaînes de caractères contenues dans `argv`. Exécuter le programme avec et sans arguments. Quelle est la chaîne stockée dans `argv[0]` ? Quelles sont les chaînes stockées dans `argv[i]` pour `i` > 0 ?

Exercice 3 : Une version basique de `cp`

Utilisez ce que vous avez appris dans l'exercice précédent pour écrire votre propre implémentation de la commande `cp`. Plus précisément, votre programme devra prendre en arguments deux noms de fichiers et copier le contenu du premier dans le deuxième. Il faudra utiliser les fonctions vues au premier semestre pour gérer les fichiers : `fopen`, `fgetc`, `fputc` et `fclose`. D'ailleurs, dans quels modes faut-il ouvrir les deux fichiers pour faire ce que l'on veut faire ?
Vous pourrez utiliser la commande `diff`, qui prend en arguments deux noms de fichiers existants et affiche leurs différences, pour vérifier que votre programme copie *exactement* le contenu du premier fichier dans le deuxième.

Exercice 4 : Analyse d'un fichier texte

Faire un programme qui, étant donné un fichier donné en argument, affiche
- son nombre de lignes,
- le nombre d'occurrences du caractère `e`,
- sa taille,
- s'il s'agit d'un fichier texte 7-bit (n'utilisant que des caractères de numéro $\leqslant 127$).

Faire des tests à l'aide d'un fichier `fichier.txt`. Vous pouvez vérifier si votre programme est correct avec les commandes `wc fichier.txt` (qui donne la taille et nombre de lignes), `tr -cd e < fichier.txt | wc -c` (pour nombre de `'e'`) et `file fichier.txt` pour le type de fichier.

Exercice 5 : reconnaissance des principaux formats d'image

Bien que les fichiers d'image ne soient pas du texte, ils ont tous dans leur entête une "signature" qui permet de les reconnaître. Ainsi

- Format `GIF` : les six premiers octets sont `'G','I','F','8','9','a'`. Le 89a vient de ce que la variante utilisée actuellement du format a été normalisée en 1989.
- Format `JPEG` : les octets de numéro 6 à 10 inclus du fichier forment la chaîne `"JFIF"` (signifiant « JPEG File Interchange Format », où JPEG veut dire « Joint Photographic Experts Group »)
- Format `PNG` : les octets de numéro 1 à 5 inclus sont `'P', 'N', 'G', '\r', '\n'`.
- Le format NetPBM, plus rare, est un format non compressé d'images. Il correspond aux suffixes `.ppm` `.pgm` et `.pbm`. Dans ce format la première ligne est forcément composée de deux caractères, le premier étant `'P'` et le second un chiffre entre `'1'` et `'6'`.

Faire un programme qui reconnaît si un fichier est une image de l'un de ces quatre formats. Le tester sur des images de ces types que l'on pourra trouver sur Internet.