TP0 : découverte de l'environnement

Algorithmique et projet de programmation — M1 mathématiques

13 septembre 2016

Les énoncés (et quelques corrections) des TPs seront mis au fur et à mesure sur ma page web, à l'adresse :

`http://www.normalesup.org/~cagne/teaching.html`

Les exercices du contrôle continu et le projet de programmation final seront à envoyer à l'adresse email `pierre.cagne@normalesup.org`.

1 UNIX et la console

Les machines de l'université utilisent un système d'exploitation UNIX-like tel que GNU/Linux, BSD, Solaris, etc. connu pour sa puissante console. Il s'agit d'un programme permettant à l'utilisateur de réaliser toutes les tâches administratives sur une machine grâce à de simples commandes entrées au clavier. En particulier, pour les TPs de ce cours, on utilisera la console pour compiler nos programmes C. Toute commande possède un manuel accesible en tapant `man [command name]`. Par exemple, pour savoir comment utiliser la commande `ls`, on peut entrer `man ls`.

Exercice 1. Consulter les manuels des commandes `ls`, `mkdir` et `wget`.

Exercice 2. En **ligne de commande**, créer un dossier `td0` et y télécharger le fichier `http://www.normalesup.org/~cagne/teaching/20162017/algo_proj_m1/tp0.zip`. Extraire le fichier téléchargé à l'aide de la commande `unzip`.

Remarque. Se référer aux pages `man` doit devenir un rélfexe quand vous n'êtes pas sûr de l'utilisation d'une commande. C'est l'esprit RTFM...

2 Le langage C

Le langage `C` est un langage de programmation, au même titre que `Python`, `Java`, `Scilab`, `R`, etc. que vous avez peut-être déjà utilisé. C'est un "vieux" langage (il date de 1972), ce qui lui fournit à la fois des avantages (large communauté, très sûr, optimisé pour la plupart des compilateurs, etc.) et des inconvénients (syntaxe lourde, absence des paradigmes "nouveaux", gestion de la mémoire manuelle, etc.). C'est un langage *compilé*, c'est-à-dire qu'il fonctionne approximativement de la manière suivante :

- on écrit un fichier texte avec un éditeur tel `emacs`, avec une syntaxe bien définie (que l'on va apprendre tout au long du semestre), dont l'extension doit être `.c`, disons `main.c` pour l'exemple,

- on appelle un compilateur, usuellement via la ligne de commande, sur le fichier `main.c` nouvellement créé ; son rôle est de *traduire* le code source écrit en `C` en code machine compréhensible directement par le processeur ; on utilisera dans ce cours le compilateur *open source* `gcc`, un des plus utilisé dans le monde du logiciel libre,
- on obtient alors en sortie un fichier exécutable, disons `main.out`, qu'on peut alors appeler grâce à la ligne de commande (presque) comme n'importe quel autre programme.

Plus spéficiquement, pour compiler un fichier source appelé `main.c` en un programme exécutable `main.out`, on se place dans le dossier contenant le source et on exécute la commande suivante :

```
gcc -Wall -o main.out main.c
```

On voit que `gcc` prend ici, outre le source `main.c`, deux autres arguments commençant par des tirets. Ce sont des *flags* qui agissent sur le comportement de `gcc`. Le premier, `-Wall`, signifie *warnings all* et permet d'afficher toutes sortes d'avertissements de la part du compilateur : vous êtes libre de ne pas utiliser ce flag, mais vous risquez alors d'avoir plus de mal à comprendre les erruers de compilations qui pouraient surgir. Le deuxième flag est `-o`, signifiant *out*, et prend lui-même en paramètre un nom de fichier, ici `main.out` : il sert à indiquer le nom du fichier exécutable créé par le compilateur à partir du source `main.c` : on peut également ommettre ce flag, mais alors l'exécutable obtient par défaut le nom très explicite de `a.out`... Ce sera probablement les deux flags que vous utiliserez le plus dans ce cours, mais n'hésitez pas à aller faire un tour sur le manuel (`man gcc`) pour vous faire une idée du nombre phénoménale de paramètres possibles que peut prendre `gcc` !

On peut maintenant exécuter notre programme simplement en rentrant son chemin d'accès. Attention, si on l'exécute depuis le dossier le contenant (et c'est le cas le plus fréquent), il faut entrer `./main.out` et non pas seulement `main.out`.[1]

Remarque. L'extension en `.out` n'a rien d'obligatoire. En fait, la norme est plutôt de ne pas utiliser d'extension du tout : pour ouvrir Mozilla Firefox par exemple, on utilise en effet la commande `firefox` et non pas `firefox.out` !

Exercice 3. Compiler le fichier `hello.c` (présent parmi les fichiers extraits précédemment) et exécuter le programme obtenu.

Exercice 4. Ouvrir le fichier `hello.c` depuis la ligne de commande. Par exemple, si on utilise l'éditeur `emacs`, on exécute la commande :

```
emacs hello.c &
```

(Le symbole `&` donne l'ordre au terminal de nous rendre la main avant la fin du processus lancé. En effet, on voudrait pouvoir réutiliser la ligne de commande sans avoir à fermer notre éditeur de texte !)

Essayer de comprendre la syntaxe du document.

Exercice 5. Modifier le fichier `hello.c` et le recompiler pour voir l'effet de la modification sur l'exécution du programme.

Exercice 6. Faire de même avec les autres fichiers extraits de `tp0.zip`.

1. Si l'on omet `./`, le terminal va automatiquement chercher le programme appelé, ici `main.out`, dans des dossiers spécifiques, listés dans une variable `$PATH`. Vous pouvez vous rendre compte que le dossier courant n'y est pas en tapant simplement `echo $PATH`.