MM031 - TP1

Maxime Chupin : chupin@ann.jussieu.fr

19 Janvier 2016

1 Manipulation du système

1.1 Utilisation basique d'un terminal

Un terminal est une fenêtre d'interface vous permettant de piloter votre système à l'aide de commandes à taper au clavier. Une interface en ligne de commande est une interface *homme-machine* dans laquelle la communication entre l'utilisateur et l'ordinateur s'effectue en mode texte :

→ l'utilisateur tape une ligne de commande, c'est-à-dire du texte au clavier pour demander à l'ordinateur d'effectuer une opération ;

→ l'ordinateur affiche du texte correspondant au résultat de l'exécution des commandes tapées ou à des questions qu'un logiciel pose à l'utilisateur.

Dans ce TP, nous ne parlerons que des terminaux des systèmes d'exploitations Unix (dont dérive Linux). Dans la suite du cours, libre à vous d'utiliser un autre système d'exploitation (Windows, Mac OSX, etc.). Cependant, je vous encourage vivement à utiliser Linux.

Pour une aide mémoire sommaire mais suffisante des commandes Unix les plus utiles, on pourra se référer au document de *Génération Linux* qui se trouve à l'adresse suivante : `http://www.generation-linux.fr/dl/Les_commandes_linux.pdf` et que je mets aussi sur mon site professionnel : `https://ljll.math.upmc.fr/~chupin`.

Après avoir ouvert un terminal, effectuez les taches suivantes :

1. Tapez la commande `pwd` puis pressez `Entrée` pour vous *localiser*.
2. À l'aide de la commande `ls`, visualisez le contenu du répertoire courant.
3. À l'aide de la commande `cd` *<nom du répertoire>*, déplacez vous dans les répertoires.
4. Créez un répertoire `MM0031` à l'aide de la commande `mkdir` *<nom du répertoire>* puis un répertoire `TP1` à l'intérieur de celui-ci. Vous travaillerez dans ce répertoire durant le TP.

Remarque : pour toutes ces fonctions une documentation est disponible en tapant la commande `man` *<nom de la fonction>*.

1.2 Écriture d'un programme

L'éditeur de texte est un outil qui vous permet d'éditer (écrire) dans un fichier texte. Tous vos programmes seront d'abord écrit dans un fichier texte avant d'être compilés. Par défaut, nous utiliserons l'éditeur `gedit` installé sur les machines de l'université. Il en existe un grand nombre et vous êtes libre d'utiliser l'éditeur de votre choix comme : `emacs`, `vi`, etc. Il est également possible d'utiliser des outils comme `eclipse`, `xcode` ou encore `visualBasic`, etc.

Le choix de l'éditeur de texte est libre et très personnel, l'important est de se sentir à l'aise avec celui qu'on utilise. Cependant, `emacs` et `vim`, bien que difficile de prise en main, sont des outils vraiment très efficaces. Pour ces deux logiciels, en plus de la documentation foisonnante sur internet, on peut trouver deux aides mémoires à l'adresse suivante : `https://ljll.math.upmc.fr/~chupin`.

1.2.1 Le programme `Hello World`

1. Créez un fichier texte `hello_world.cxx` à l'aide de la commande `emacs hello_world.cxx &`. L'utilisation du caractère & vous permet de garder la main sur votre terminal. On utilisera l'éditeur de son choix bien évidemment.
2. Écrivez un programme qui ce contente d'afficher `Hello world!` dans le terminal.

1.3 Compiler et exécuter un programme C++

Pour compiler un programme de la manière la plus basique on utilise la commande [1]

```
> g++ hello_world.cxx
```

Le compilateur `g++` peut émettre une liste d'erreurs, si tel est le cas, il faudra les corriger si l'on veut pouvoir exécuter son programme.

Il est important de prendre le temps de bien lire les messages du compilateur qui donnent souvent des informations précieuses pour comprendre la nature de l'erreur. La compilation se déroule en fait en deux étapes :

1. La création d'un module objet `hello\_world.o` qui est la traduction en langage machine du fichier texte.
2. L'édition de liens qui génère un fichier exécutable `a.out` qui correspond au programme écrit dans le fichier compilé. L'édition de liens peut permettre de rassembler plusieurs modules objets dans le même programme.

L'exécution du programme s'effectue en suite à l'aide de la commande

```
> ./a.out
```

Si vous voulez donner un autre nom que `a.out` à votre exécutable il faut utiliser la commande

```
> g++ hello_world.cxx -o nom_de_lexecutable
```

Voici comment effectuer de manière distincte les deux étapes de compilation de vos programmes C++ :

1. `g++ -o nomfichier.o -c nomfichier.cxx` : compilation, génération du fichier `nomfichier.o`.
2. `g++ nomfichier.o -o nom_de_lexécutable` : édition de liens, génération d'un exécutable `nom_de_lexécutable`.

Compiler et exécuter le programme `hello_world.cxx` avec cette nouvelle série de commande.

Remarques :

→ L'ajout de l'option de compilation `-Wall` à la compilation permet d'afficher un maximum d'alertes (*warnings*) révélant quelques maladresses de programmation à corriger impérativement. On utilise cette option de cette manière `g++ -o nomfichier.o -c nomfichier.cxx -Wall`

→ L'ajout de l'option `-g` active les options de *debuggage* très utile des erreurs de programmation plus pointues. Elle s'utilise comme l'option `-Wall`.

→ On peut utiliser plusieurs options en simultané.

1. Le caractère > indique que l'on se trouve dans le terminal.

2 Entrées/Sorties

2.1 Saisie et affichage dans le terminal

Utilisation des fonctions `cin` et `cout`. Pour utiliser les entrées/sorties, on doit inclure le fichier d'en tête : `<iostream>`. Pour cela, notre fichier doit ressembler à :

```
1 # include <iostream>
2
3 using namespace std;
4
5 int main(){
6    ...
7
8    return 0;
9 }
```

Par commodité on se place dans l'espace des noms standard avec l'instruction en ligne 3. En effet, la classe `iostream` introduit les fonctions utiles pour l'écriture et la lecture sur la *sortie standard* (*e.i.* le terminal) dans l'espace des noms `std`, c'est-à-dire que pour appeler la fonction d'écriture `cout`, on doit écrire `std::cout`. En utilisant l'espace des noms standard, on peut simplement appeler la dite fonction par `cout`.

Exercice 1

→ Écrire un programme permettant à l'utilisateur de saisir deux entiers et d'afficher leur somme.
→ Modifier votre programme pour qu'il fasse la somme de deux entiers positifs en indiquant une saisie erronée à l'utilisateur si lors de la saisie l'entier est négatif.

Exercice 2

Rappel : tableau en C++, `type nom_du_tableau[nombre_éléments];` .
→ Écrire un programme permettant à l'utilisateur de saisir autant de valeurs qu'il le souhaite et les stocker dans un tableau.
→ Calculer la moyenne de ces valeurs.
→ Afficher l'ensemble des valeurs saisies ainsi que la moyenne.

Exercice 3

→ À l'aide de la fonction `switch`, écrire un programme permettant à l'utilisateur de saisir un entier et affichant le jour de la semaine correspondant si cet entier est compris entre 1 et 7. Dans le cas contraire on affichera "cet entier ne correspond à aucun jour".

2.2 Saisie et affichage dans un fichier

Inclure l'en tête : `# include <fstream>` qui permet la gestion de lecture et d'écriture dans des fichiers.

```
1 #include <iostream>
2 #include <fstream>
3 using namespace std;
4
5 int main(){
6    ...
```

```
7
8       return 0;
9   }
```

Exercice 4

→ Créer un fichier `input.dat` contenant un nombre de valeurs à lire sur le première ligne et sur la ligne suivante toutes les valeurs.
→ À l'aide de la fonction `ifstream`, lire le contenu du fichier.
→ avec la fonction `ofstream`, réécrire les données dans le fichier *output.dat* en multipliant toutes les valeurs par 10.

Exercice 4 bis

→ Donner les noms de fichier à lire et à écrire en argument à l'exécution du programme.