[Math-L312] TP 0 : Présentation rapide de UNIX, Linux et de gcc

Adrien Semin
adrien.semin@math.u-psud.fr

1 UNIX et Linux

Linux est un système d'exploitation de type UNIX (ce qui signifie qu'il est à la fois **multi-tâches** et **multi-utilisateurs**) à l'origine écrit par Linus Torvalds qui permet de travailler sur des PC. Il existe d'autres systèmes UNIX, qui permettent de commander d'autres types de machines (par exemple des stations SUN), ce format garantit que la plupart des commandes qu'on va mentionner ici resteront valables sur ces autres versions. Ce système d'exploitation est gratuit, open-source et en développement constant, c'est à dire que n'importe qui peut avoir accès aux sources (en C ou C++ selon les versions) et les modifier ou les améliorer.

Vous pourrez avoir accès à un document en ligne présentant la version de Linux installée dans les salles d'informatiques de la fac d'Orsay à l'adresse suivante : http://www.math.u-psud.fr/%7Epascal/linux.html

1.1 Les terminaux et leurs raccourcis clavier courants

À la différence d'autres systèmes d'exploitation, il est très courant d'utiliser sous Linux des fenêtres de terminal (ou shell) pour taper directement des commandes en mode texte plutôt que d'utiliser une interface graphique. Voici une liste de quelques raccourcis utiles pour les utiliser efficacement :

- **Le presse-papier** : pour copier du texte, il suffit de le sélectionner avec le bouton gauche de la souris. Le texte est automatiquement copié lorsqu'on relâche le bouton. Pour le coller, il suffit de cliquer avec le bouton du milieu et le texte sera inséré à la position du curseur.
- **Avoir la liste des commandes ou des fichiers du répertoire courant** : si on commence à taper les premières lettres d'une commande existante, appuyer plusieurs fois sur la touche (Tab) permet de faire défiler la liste des commandes commençant par ces lettres. On pourra avoir aussi la liste des fichiers du répertoire courant de cette façon-là pour les arguments des commandes.
- **Utiliser l'historique des commandes :** avec les touches (Haut) et (Bas), on peut remonter ou descendre dans l'historique des commandes déjà tapées. Cela peut être utile pour éviter d'avoir à taper de nouveau des commandes identiques ou quasi-identiques à chaque fois.
- **Terminer une commande** : certaines commandes prennent du temps à s'exécuter. Par exemple, si vous avez écrit un programme qui entre dans une boucle infinie, il peut être utile de le forcer à s'arrêter. Il faut appuyer simultanément sur (Ctrl) et (C).
- **Mettre une tâche en pause** : appuyer simultanément sur (Ctrl) et (Z)[1]
- **Arrêter le défilement** : appuyer simultanément sur (Ctrl) et (S)
- **Reprendre le défilement** : appuyer simultanément sur (Ctrl) et (Q)

1.2 Les commandes de base

On va rappeler ici quelques commandes de base utiles[2] (mentionnées par ailleurs pour la plupart dans le document cité précédemment).

[1] Ce raccourci est utile pour les commandes qui lancent une interface graphique, comme `emacs` par exemple. Ainsi, on peut reprendre la main dans la fenêtre de terminal sans avoir à quitter le programme. La commande `bg` permet alors de reprendre la tâche mise en pause, mais en tâche de fond. On peut obtenir un comportement identique en faisant suivre la commande du symbole `&`, par exemple : `emacs &`

[2] **Attention** : Linux est «case-sensitive», c'est à dire qu'il fait la différence entre les majuscules et les minuscules. Par exemple la commande `Cd` n'existe pas.

TAB. 1: Les commandes les plus courantes

Catégorie	Commande	Signification
Obtenir de l'aide	`man <commande>` :	donne une fiche d'aide détaillée sur une commande. En général, pour une commande donnée, la commande suivie de l'option `--help` permet d'en avoir la version courte (et souvent plus immédiatement utile). Vous pourrez trouver des informations utiles et détaillées sur la plupart des fonctions et librairies standards du langage C en spécifiant à `man` d'afficher la section 3 du manuel, par exemple : `> man 3 printf` affichera la rubrique associée à la fonction `printf`.
	`xman` :	afficher l'aide détaillée dans une fenêtre externe.
Système de fichiers[3]	`cd` :	changer le répertoire courant. Certains symboles désignent des répertoires prédéfinis : – le répertoire de l'utilisateur[4] : `~` – le répertoire courant : `.` – le répertoire situé immédiatement au-dessous du répertoire courant : `..` Cette commande sans argument permet aussi de revenir au répertoire de l'utilisateur.
	`ls` :	afficher la liste des fichiers et répertoires du répertoire courant.
	`rm` :	détruire un fichier ***de façon irréversible***.
	`rd` :	détruire un répertoire (celui-ci doit être préalablement vide).
	`mkdir` :	créer un répertoire.
	`mv` :	déplacer un fichier ou répertoire (ou le renommer).
	`cp` :	copier un fichier.
	`find` :	chercher des fichiers dans une arborescence.
	`chmod` :	changer les droits d'accès à vos fichiers et répertoires.
	`tar` :	créer/extraire une archive contenant des fichiers et répertoires.
	`gzip` et `gunzip` :	compresser/décompresser un fichier.
Gestion du compte	`yppasswd` :	changer son mot de passe[5]. Lorsque vous tapez le nouveau mot de passe, **il ne faut pas utiliser les chiffres du pavé numérique** sinon vous risquez de ne plus pouvoir accéder à votre compte.
	`du` :	afficher l'espace disque occupé par le répertoire courant et son contenu récursif[6].
	`quota` :	permet de connaître la taille des quotas.
	`logout` :	permet de se déconnecter du terminal[7].
Recherche et comparaison	`diff` :	comparer les différences entre 2 fichiers.
	`grep` :	rechercher une suite de caractères dans un fichier
Processus et tâches	`ps` :	afficher la liste des process lancés par l'utilisateur courant.
	`jobs` :	afficher la liste des tâches.
	`kill` :	«tuer» un processus ou une tâche appartenant à l'utilisateur courant.
	`killall` :	«tuer» toutes les tâches d'un nom donné.
	`bg` :	réactiver une tâche mise en pause, mais en arrière-plan (la fenêtre de terminal reste alors active).
Édition de fichiers	`pico` :	un petit éditeur de fichier-texte très simple en mode terminal

Suite page suivante...

[3] Il est évidemment recommandé de bien connaître ces commandes et leur utilisation, afin de maintenir un certain niveau d'organisation dans les fichiers de votre compte. Par exemple, Emacs crée des fichiers temporaires ou de *backup* lorsque vous sauvegardez quelque chose. Il est recommandé de détruire ces backups lorsqu'ils ne sont plus utiles.

[4] Sur les systèmes `UNIX` chaque utilisateur a son propre répertoire racine, c'est le répertoire courant qui est activé au démarrage.

[5] Il vous est demandé, pour des raisons de sécurité, de changer votre mot de passe dès la première utilisation de votre compte !

[6] `Linux` étant un système multi-utilisateur, tous les fichiers de tous les utilisateurs sont répartis sur les machines du réseau. Il vous est demandé de ne pas dépasser un certain quota d'espace-disque occupé, (auquel cas votre compte serait désactivé jusqu'à suppression du problème). La commande `du` vous permet de connaître l'espace-disque que vos fichiers occupent.

[7] **Attention** : après chaque session que vous ouvrirez sur une machine, il faudra penser à vous délogger de la machine, pour des raisons de sécurité évidentes !

Suite des commandes les plus courantes		
	`emacs` ou `xemacs` :	un éditeur plus perfectionné. Emacs permet un formatage visuel automatique de la plupart des types de fichiers (`*.c`, `*.tex`, `*.html` ...) de façon à les rendre plus lisibles. Il effectue aussi une indentation automatique pour le C par exemple.
	`kate` :	un environnement d'édition de code-source multi-document très pratique, recommandé pour taper du C en TP par exemple.
Lecture de fichiers	`acroread` :	pour lire les fichiers Acrobat Reader (`*.pdf`), et en particulier les polycopiés du cours. Par exemple, vous pouvez afficher ce document en tapant `> acroread /home/doc/semin/TP0/TP0.pdf &` Il faudra bien sûr remplacer le '`0`' par le numéro du TP en cours pour les prochaines séances.
	`xdvi` :	pour lire les fichiers `*.dvi`.
	`ghostview` :	pour lire les fichiers PostScript (`*.ps`).
	`cat` ou `more` :	pour afficher le contenu brut d'un ou plusieurs fichiers dans la sortie standard.
Navigation web	`lynx` :	un navigateur en mode texte très rapide. Évidemment, vous ne verrez pas les images s'afficher...
	`mozilla` :	un navigateur avec interface graphique.
	`netscape` :	un autre navigateur avec interface graphique.
Communication	`pine` :	un client pour lire et écrire ses mails en mode texte. Ne demande a priori aucune configuration particulière pour lire vos mails sur votre compte.
	`talk` :	un programme pour converser à distance avec un autre utilisateur.
Compilation	`gcc` :	le compilateur C qu'on va utiliser.
	`make` :	un programme pour réaliser automatiquement des compilations interdépendantes.
Divers	`mount` :	consulter la liste des accès aux fichiers du server. Cette commande permet aussi d'utiliser des lecteurs de disquette, CD (`mountcd`) et autres.
	`umount` :	pour démonter un accès au système de fichiers (par exemple un lecteur CD, voir `umountcd`).
	`who` :	pour afficher la liste des autres personnes connectées sur les autres machines du réseau local.
	`whoami` :	afficher son nom d'utilisateur.
	`date` :	afficher la date.
	`echo` :	afficher du texte.

2 gcc

`gcc` signifie « GNU C Compiler ». Cela indique que `gcc` est un programme sous license `GNU`[8]. Ce programme va transformer vos fichiers de code-source en langage C, en des fichiers exécutables (i.e. dans un langage que la machine peut comprendre).

2.1 Le langage C

Un excellent polycopié sur le langage C se trouve ici : `http://www.dil.univ-mrs.fr/~garreta/Polys/PolyC.pdf`.

2.2 Premiers pas avec gcc

On va expliquer pas à pas la création d'un programme simple qui écrit «Hello world !» dans la sortie standard. Voici le code-source de ce programme en langage C, téléchargeable à l'adresse `/home/doc/semin/TP/c/hello.c` :

[8] Ce type de license correspond à des programmes opensource et gratuits, c'est à dire que vous pouvez les utiliser gratuitement et accéder à leurs sources, voire les modifier. Il est d'ailleurs possible que la version installée de `gcc` sur les machines que vous utilisez ait elle-même été compilée avec `gcc`...

```c
#include <stdio.h>

int main ()
{
  printf("Hello u-psud !\n");
  return 0
}
```

Pour compiler ce fichier, il faut suivre les étapes suivantes :

1. Il faut d'abord taper ce code-source (sans fautes de frappe !) dans un éditeur de fichier-texte (par exemple `kate` ou aussi `pico` en mode terminal), et on l'enregistre sous le nom `hello.c`.
2. Ensuite, on le compile avec `gcc` :
   `> gcc hello.c -o hellocompile`
   L'option `-o hellocompile` permet de spécifier à `gcc` d'écrire les données compilées de l'exécutable dans le fichier `hellocompile` (par défaut, ce sera dans `a.out` si on omet cette option). Si le compilateur n'imprime pas de messages d'erreur, c'est que tout s'est passé correctement.
3. On peut ensuite exécuter le fichier `hellocompile`, en utilisant la commande :
   `> ./hellocompile`

`gcc` admet d'autres options, telles que `-I` pour inclure un répertoire pour rechercher des fichiers d'`include`. Si le programme qu'on veut compiler utilise des fichiers situés dans le répertoire courant par exemple (avec la directive `#include <file>`), il faudra utiliser l'option : '`-I.`' (on rappelle que '.' est le nom abrégé pour le répertoire courant).

`gcc` affichera 2 types de messages lors de la compilation :

- les erreurs (error). Dans ce cas, `gcc` ne générera en général pas de fichier exécutable.
- les avertissements (warnings). Ceux-ci servent à indiquer que l'on compile du code suspect (peu fiable), ou qui ne respecte pas les conventions du langage C.

S'il n'y a pas de message affiché, c'est que tout s'est passé correctement, mais évidemment *ceci ne constitue en aucun cas une preuve que votre programme va s'exécuter sans erreur* !

3 Travail à faire

- `listing.txt` ***À Rendre!*** Créez la structure de répertoires suivant :
  `~/a`
  `~/a/a1`
  `~/a/a2`
  `~/a/a2/a21`
  `~/a/a3`
  `~/a/a3/a31`
  `~/a/a3/a32`
  `~/a/a3/a33`
  Placez-vous dans le répertoire /a/a3/a31, et tapez la commande
  `echo Bonjour, je suis 'whoami' > toto.txt`
  Affichez le contenu du fichier `toto.txt`, puis copiez-le dans le répertoire `~/a/a3/a33`, en changeant son nom de destination en `tutu.txt`
  Déplacez le fichier `toto.txt` dans le répertoire `~/a` Listez récursivement le contenu du répertoire `~/a` dans le fichier `~/listing.txt`, puis effacez le répertoire `~/a`
- Dans votre répertoire utilisateur, créez un répertoire `M312`, puis dans ce répertoire `M312`, créez un répertoire `TP0`, puis placez-vous dans ce répertoire.
- `prog1.c` et `prog1c.c` ***À Rendre!*** dans le répertoire `~/M312/TP0`, tapez le code suivant :

```c
// Contenu du fichier prog1.c
#include <stdio.h>

int main(void)
{
  float a = 2.5;
  int b = 2.5;
  float c = a + b;
  printf("a + b = %f\n",c);
  return 0;
```

```
}
```

Compilez ce programme.
Que donne ce programme ? Pourquoi ?
Copiez le fichier ci-dessus dans le fichier `prog1c.c`, et corrigez le fichier `prog1.c`

– `prog2.c` et `prog2c.c` *** À *Rendre!* *** dans le répertoire `~/M312/TP0`, tapez le code suivant :

```
// Contenu du fichier prog2.c
#include <stdio.h>

int main(void)
{
  int a=5,
    b=3;
  c = a / b;
  printf("a / b = %d\n",c)
  return 0;
}
```

Compilez ce programme. Pourquoi le programme ne compile pas ?
Copiez le fichier ci-dessus dans le fichier `prog2c.c`, et corrigez le fichier `prog2c.c` de manière à pouvoir compiler le programme.
Que donne ce programme ? Pourquoi ?

Lorsque, dans les feuilles de TP, vous voyez *** À *Rendre!* *** , vous devez envoyer le ou les programmes demandés à l'aide de la fonction `/home/doc/semin/uploadwork`. Pour envoyer tous les fichiers `.c` qui se trouvent dans le répertoire courant, il suffit de taper `/home/doc/semin/uploadwork *.c`