Labo 1

Objectifs : LINUX, la ligne de commande, VI et GCC.

Organisation

Les laboratoires de système d'exploitation ont pour objectif de vous permettre de mettre en pratique les différentes notions vues en cours de théorie.

Vous allez tout d'abord vous familiariser au système LINUX au moyen d'un émulateur de terminal (voir ci-dessous). Ensuite diverses séries d'exercices serons mise à votre disposition.

Elles sont accompagnées d'un document vous permettant

- de découvrir la matière au moyen de quelques exemples et d'explications complémentaires, et
- de la mettre en pratique au travers d'exercices nécessitant de plus en plus d'autonomie dans la recherche de la solution.

Le langage de programmation utilisé est le langage C.

Introduction

L'objectif de ce premier labo est de découvrir la démarche pour se connecter au serveur et d'apprendre à utiliser Linux via la ligne de commande (sans interface graphique).

La première partie du labo est une description succincte de LINUX, de la connexion au serveur, de la ligne de commande et de quelques commandes utiles.

Ensuite, nous aborderons les outils nécessaires à la rédaction des fichiers sources ainsi qu'à leur compilation/exécution. LINUX met à disposition une application basique pour l'édition de fichiers, VI, l'équivalent de Notepad sous Windows. Il ne nécessite pas non plus d'environnement complexe pour compiler les fichiers sources écrits en C, un compilateur en ligne de commande est prévu.

Il est conseillé d'exécuter/tester chacun des exemples ci-dessous afin de vous habituer à l'utilisation de PuTTY et des commandes.

LINUX

En quelques mots…

*"**LINUX** est le nom couramment donné à tout système d'exploitation libre fonctionnant avec le noyau LINUX. C'est une implémentation libre du système UNIX respectant les spécifications POSIX. Ce système est né de la rencontre entre le mouvement du logiciel libre et le modèle de développement collaboratif et décentralisé via Internet. Son nom vient du créateur du noyau LINUX, Linus Torvalds."*[1]

Plus en détail…

Les principales caractéristiques de LINUX sont les suivantes :

- multitâches : exécute plusieurs programmes en même temps
- multi-utilisateurs : permet à plusieurs utilisateurs d'être actifs en même temps
- multi plates-formes : fonctionne avec différents types de processeurs (Intel, Sparc, Alpha, PowerPC...)

[1] Copié de http://fr.wikipedia.org/wiki/Linux

- supporte un grand nombre de systèmes de fichiers : Ext(2|3|4), XFS, FAT, VFAT, NFS, CIFS…

Les trois termes les plus souvent rencontrés dans la description d'un système d'exploitation tel que LINUX sont le *shell*, les *system calls* et le *kernel*.

Le *shell* (en français *coquille*)

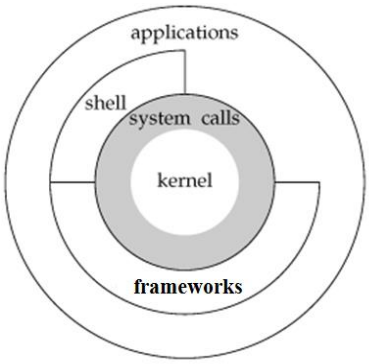

Figure 1 - Architecture de Linux

Il est chargé de faire l'intermédiaire entre le *kernel* et l'utilisateur. Appelé *interface système*, c'est une couche logicielle qui constitue l'interface utilisateur d'un système d'exploitation.

Le *shell* peut se présenter sous deux formes :

- une interface en ligne de commande (en anglais *Command Line Interface*, **CLI**)
- une interface graphique (en anglais *Graphical User Interface,* **GUI**)

Vous utilisez principalement des interfaces graphiques lorsque vous travaillez sur un ordinateur, nous supposerons donc que vous savez de quoi il s'agit.

Dans le cadre de ces séances d'exercices, vous aurez à faire à une interface en ligne de commande. Comme vous avez eu peu, voire aucun contact avec cette pratique "ancestrale", nous allons vous présenter cette deuxième manière d'interagir avec le système d'exploitation avec un peu plus de détails dans la suite de ce document.

Le *shell* est en résumé un exécutable chargé d'interpréter les commandes, de les transmettre au noyau et de retourner le résultat.

Il existe plusieurs *shells* : **sh** (*Bourne shell*), **bash** (*Bourne-again shell*), **csh** (*C Shell*), **tcsh** (*Tenex C shell*), **ksh** (*Korn shell*), **zsh** (*Zero shell*), **rsh** (*Remote shell / Restricted shell*). Leur nom correspond généralement au nom de l'exécutable.

Le *kernel* (en français *noyau*)[2]

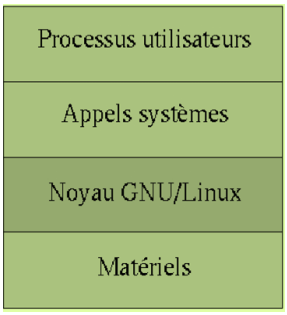

Figure 2 - Vue en couches

Pour permettre aux éléments matériels (processeur, mémoire, périphériques…) d'interagir entre eux (transférer des données, effectuer des opérations complexes…), il est nécessaire d'avoir un élément logiciel.

Comme vu dans le cours de Description des ordinateurs, lors du démarrage de l'ordinateur, le BIOS initialise chaque élément, vérifie ses capacités et règle les divers composants du système (priorité des bus, fréquence d'horloge des ports PCI, fréquence du processeur…). Après cette phase d'initialisation, le noyau prend le relais pour gérer les accès aux éléments matériels.

L'existence d'un noyau nécessite une séparation virtuelle de la mémoire vive physique en deux régions disjointes, l'une réservée au noyau (*espace noyau*) et l'autre dévolue aux autres applications (*espace utilisateur*). En effet, le noyau a tout pouvoir sur l'utilisation des ressources matérielles, en particulier de la mémoire. Dans l'espace utilisateur, un certain nombre de restrictions sont donc

[2] Inspiré de http://linux-france.mirrors.skynet.be/prj/inetdoc/cours/interco.noyau/interco.noyau.presentation.html et de http://notesdecours.drivehq.com/courspdf/Cours%20OS_JNinforge_V1.1.pdf

imposées (par exemple, la création d'un fichier ne peut se réaliser que si les droits sont suffisants), et le processus ne peut avoir accès qu'aux zones mémoires qui lui ont été allouées.

On lit parfois que LINUX a deux modes de fonctionnement : le mode noyau (superviseur), où toutes les instructions sont autorisées, et le mode utilisateur (pour les programmes des utilisateurs et les utilitaires), où certaines instructions ne sont pas permises.

De plus, cette architecture influe également sur le travail des développeurs : le développement de code dans l'espace noyau est a priori plus délicat que dans l'espace utilisateur car la mémoire n'y est pas protégée. Ceci implique que des erreurs de programmation, altérant éventuellement les instructions du noyau lui-même, sont potentiellement beaucoup plus difficiles à détecter que dans l'espace utilisateur où de telles altérations sont rendues impossibles par le mécanisme de protection.

Le noyau de LINUX est composé de cinq sous-systèmes principaux. Un sous-système peut être défini comme un ensemble de fonctions/procédures qui fournissent une fonctionnalité particulière.

Gestion des processus et communications inter processus

Ce sous-système est chargé de répartir équitablement les accès au processeur entre toutes les applications actives. Cela n'inclut pas seulement les processus utilisateurs, mais aussi les sous-systèmes du noyau lui-même. Cette fonction est réalisée par le *scheduler*.

Dans la mesure où un processus ne peut avoir accès qu'à la zone mémoire qui lui a été allouée, LINUX propose plusieurs mécanismes permettant à des applications de communiquer entre elles.

Gestion de la mémoire

Ce sous-système est chargé d'affecter à chaque programme une zone mémoire. Il a également un rôle de protection : la mémoire pour un processus est privée et celle-ci ne doit pas être lue ni modifiée par un autre.

Système de fichier virtuel

Ce sous-système garantit une gestion correcte des fichiers et un contrôle des droits d'accès. Pour limiter la complexité liée aux nombreux systèmes de fichiers existants, LINUX adopte le concept de *Virtual FileSystem* (VFS). Le principe du VFS est de proposer des appels systèmes identiques quel que soit le système de fichiers. Il est de la responsabilité du noyau de détourner les appels standards vers les appels spécifiques au système de fichiers.

Service réseau

Ce sous-système permet à LINUX de se connecter à d'autres systèmes à travers un réseau informatique. Il y a de nombreux périphériques matériels qui sont supportés et plusieurs protocoles réseaux peuvent être utilisés.

Le noyau donne accès aux ressources qu'il gère au travers des appels système.

Les *system calls* (en français *appels système*, abrégé *syscall*)

Les appels système LINUX constituent l'interface de base entre les applications (ou les commandes) et le noyau. À chaque appel correspond une opération ou une fonctionnalité de base du noyau.

Sur la majorité des systèmes d'exploitation, les appels système peuvent être utilisés **comme** de simples fonctions écrites en C.

Il est cependant important de noter qu'un appel système n'est pas identique à un appel de fonction classique. Une procédure spécifique est nécessaire pour transférer le contrôle au noyau. En effet, lorsqu'un programme fait un appel système ou lors de l'appel d'une commande du *shell*, le noyau transmet ou interprète les informations du matériel via des interruptions. (Voir labo 2)

Quelques appels systèmes classiques sont *open*, *read*, *write* et *close* qui permettent les manipulations sur les systèmes de fichiers.

Distributions

On appelle distribution l'ensemble des logiciels autour d'un noyau LINUX qui fournit un système "clé en main".

Le noyau d'une distribution peut être mis à jour afin de permettre la prise en compte de matériels récents, toutefois cette manipulation consistant à recompiler le noyau est délicate car elle nécessite un certain niveau de connaissance du système et du matériel.

La plupart des distributions proposent également une installation graphique qui leur est propre ainsi qu'un système de gestion de paquetages permettant d'installer automatiquement des logiciels en gérant les dépendances.

Chaque distribution possède ses avantages et ses inconvénients. Les distributions les plus connues sont à découvrir sur le site www.linux.com.

Connexion au serveur LINUX

Terminal Virtuel[3]

PuTTY est un *émulateur de terminal* doublé d'un client pour les protocoles SSH, Telnet... SSH est un protocole permettant d'ouvrir une console sur un ordinateur distant (alternative plus sécurisée à Telnet). Il permet également d'établir des connexions directes par liaison série RS-232.

PuTTY est écrit et maintenu principalement par *Simon Tatham*.

C'est un logiciel libre distribué selon les termes de la licence MIT.

Un **émulateur de terminal**, aussi appelé **console virtuelle** ou **terminal virtuel**, est un logiciel qui émule le fonctionnement d'un terminal informatique.

*Un **terminal informatique** est un ordinateur complet ou un écran/clavier muni d'un module de communication. Un terminal possède peu de ressources propres, il ne fait qu'envoyer les requêtes vers un serveur via une connexion réseau ou série qui lui renvoie le résultat. Les terminaux travaillent donc typiquement en mode client/serveur.*[4]

En informatique, l'**émulation** consiste à substituer un élément de matériel informatique – tel un terminal informatique, un ordinateur ou une console de jeux – par un logiciel. L'émulation est généralement faite dans une fenêtre, ce qui permet d'émuler et utiliser plusieurs terminaux sur un seul moniteur d'ordinateur.

Afin de pouvoir exécuter les différents exercices proposés aux laboratoires, nous vous demandons d'utiliser PuTTY téléchargeable sur le site www.putty.org .

LinuxIG

Dans le cadre des laboratoires de système d'exploitation, un serveur est mis à votre disposition afin de pouvoir tester les programmes réalisés en C via le compilateur **gcc** – GNU C Compiler – que nous présenterons dans le Labo 2.

[3] Inspiré de Wikipédia
[4] Copié de http://www.materiel-informatique.be/terminal.php

Lorsque PuTTY est téléchargé, exécutez-le. La fenêtre suivante apparaît :

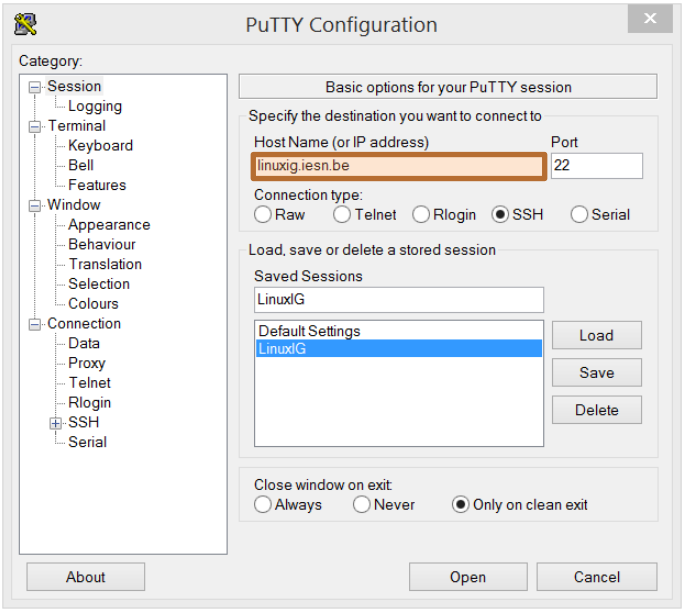

Figure 3 - Configuration de PuTTY

Inscrivez l'adresse du serveur **linuxig.iesn.be** en lieu et place de l'Host Name. Vous pouvez sauver cette adresse en inscrivant un nom sous le libellé **Saved Sessions** (ici : LinuxIG) et en cliquant sur **Save** afin de le recharger la prochaine fois. D'autres options sont disponibles dont certaines précisées lors des laboratoires.

En cliquant sur le bouton **Load** ou en double-cliquant sur le nom sauvegardé dans la liste des sessions (ici : LinuxIG), vous lancez une fenêtre permettant la communication avec le serveur en "ligne de commande".

Figure 4 - Invite de connexion de PuTTY

Lors du premier laboratoire, un nom d'utilisateur et un mot de passe vous sera fourni afin de vous connecter sur votre compte.

Avant de lancer votre session PuTTY, vous pouvez modifier certains paramètres tels que la taille de la fenêtre, le nombre de lignes "mémorisées", les couleurs…

Pour le transfert de fichiers, une connexion sFTP est possible via le programme « WinSCP » (disponible sur https://winscp.net/eng/download.php dans la rubrique « Portable executables »).

En exécutant WinSCP.exe, vous obtenez ceci :

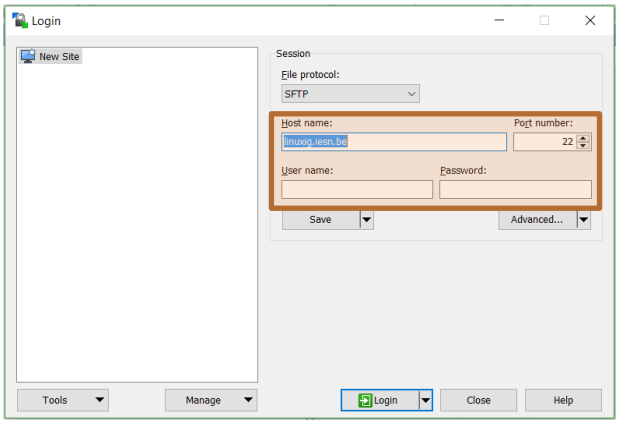

Figure 5 - FTP vers le serveur dans l'Explorer de Windows

Complétez les champs `User name` et `Password` et accédez à la session. Vous pouvez maintenant transférer des fichiers dans les deux sens.

En pratique…

Interface en ligne de commande

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, en respectant des règles syntaxiques très précises ;
- l'ordinateur affiche du texte correspondant au résultat de l'exécution de la commande.

Lorsque l'interface est prête à recevoir une commande, elle invite l'utilisateur à entrer sa commande au moyen de quelques caractères (le nom de compte de l'utilisateur, et/ou l'unité logique par défaut, et/ou le chemin par défaut, et/ou date…) se terminant par un caractère particulier (`]`, **#**, $ ou **>**). Ces caractères forment ce qu'on appelle l'**invite de commande**. Dans notre cas, ell™e ressemblera à ceci :

```
[username@svr24 username] $
```

Manuel[5]

Parmi les commandes du *shell*, la plus pratique est celle donnant accès au manuel d'utilisation des commandes : **`man`**.

Elle doit être utilisée sous la forme suivante :

```
man [-s<section>] <nom_de_commande>
```

Par exemple, pour voir ce que prévoit le manuel pour la commande `ls`, il faut taper…

```
$ man ls
```

Pour naviguer dans une page, il faut généralement utiliser les flèches vers le haut et vers le bas ou les touches *page down* et *page up*. On sort généralement d'une page avec la touche Q.

[5] Inspiré de Wikipédia

Les pages du manuel LINUX sont divisées en plusieurs sections associés chacune à un numéro à mettre en lieu et place de `<section>` :

1. Commandes utilisateur
2. Appels système
3. Fonctions de bibliothèque
4. Fichiers spéciaux
5. Formats de fichier
6. Jeux
7. Divers
8. Administration système
9. Interface du noyau LINUX

Chaque section possède une page d'introduction qui présente la section :

```
$ man <section> intro
```

Pour plus d'information sur man, il est possible de regarder la page man de man, avec la commande

```
$ man man
```

Variable d'environnement

" Les variables d'environnement constituent un moyen d'influencer le comportement des logiciels sur votre système. Par exemple, la variable d'environnement « LANG » détermine la langue que les logiciels utilisent pour communiquer avec l'utilisateur.

Les variables sont constituées de noms auxquels on assigne des valeurs. Ainsi, le système d'un utilisateur français devrait avoir la valeur « fr_FR.UTF-8 » assignée à la variable « LANG ». " [6]

Pour interpréter une commande, le *shell* doit connaître l'emplacement de celle-ci. La variable d'environnement `PATH` contient les répertoires dans lesquels le *shell* doit aller chercher la commande.

Pour connaître le contenu d'une variable d'environnement, en LINUX, on utilise la commande `echo` dont le résultat est affiché à la Figure 6.

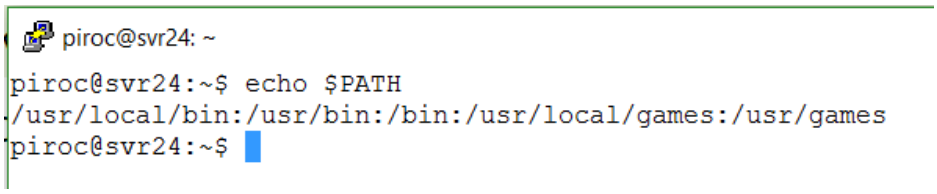

Figure 6 - Résultat de la commande "echo $PATH"

Quelques commandes utiles sous LINUX

À savoir avant de commencer !

Casse

LINUX est sensible à la casse (*case sensitive* en anglais), c'est à dire qu'il distingue les majuscules des minuscules. Ainsi, pour créer un répertoire, la commande est `mkdir`, ce n'est pas la peine d'essayer `MKDIR` ou `mKdiR`, cela ne fonctionnera pas. De même, les noms de fichiers et de répertoires sont également sensibles à la casse.

[6] Copié de http://doc.ubuntu-fr.org/variables_d_environnement

Séparateurs

Sous LINUX, le séparateur dans les chemins d'accès est un *slash*. On écrira donc `/etc/init/xfs` mais jamais `etc\init\xfs`.

Jokers

Les caractères '?' et '*' dans les noms de fichiers et de répertoires permettent de représenter des caractères quelconques. Le '?' représente un seul caractère, tandis que '*' en représente un nombre quelconque.

Par exemple, "*.jpg" représente tous les fichiers se terminant par jpg ; "*toto*" tous les fichiers contenant "toto".

Quelques commandes

Voici quelques commandes de base… De plus amples explications sont disponibles, par exemple sur https://doc.ubuntu-fr.org/tutoriel/console_commandes_de_base.

Directory Operations

`pwd`	Show current directory
`cd dir`	Change to directory *dir*
`mkdir dir`	Create a new directory *dir*
`rmdir dir`	Delete directory *dir*
`ls dir`	List contents directory *dir*

Special Directories

	Current directory
`..`	Up a directory
`.`	Current directory
`~`	Home directory
`/`	Root directory
`-`	Previous directory

ls Options

`-a`	all inc. hidden
`-l`	long format
`-t`	sort by time
`-S`	sort by size
`-r`	reverse order
`-R`	recursive

File Operations

`touch file`	Create file *file*
`cp file1 file2`	Copy *file1* to *file2*
`mv file1 file2`	Move *file1* to *file2*
`rm file`	Delete *file*
`cat file`	Display contents of *file*
`cat file1 file2`	Concatenate files
`less file`	Display *file* (paginated), q to quit
`head file`	Show first 10 lines
`tail file`	Show last 10 lines `-n N` *N* lines `-f` Continuos update

Help

`man cmd`	Manual page for *cmd*
`man -k word`	Search for manual page with *word*
`-h`	Commands show help when used

File Searching

`grep pattern file`	Search for lines with *pattern* in file
`grep -v`	Inverted search
`grep -r`	Recursive search
`grep -e patt -e patt`	Multiple patterns
`locate file`	Quick search for *file*
`which cmd`	Find location of binary
`find dir -name pattern`	Find file with *pattern* in *dir*

Standard IO Streams

stdin	Input typed on the command line
stdout	Output on the screen
stderr	Errors output on the screen
`echo string`	Write *string* to stdout

Redirection

`cmd > file`	Output of *cmd* to *file*
`cmd < file`	*file* used as input to *cmd*
`cmd >> file`	Append output to *file*
`cmd 2> file`	Write errors to *file*
`cmd &> file`	Errors and stdout to *file*

Pipes and Multiple Commands

`cmd1 | cmd2`	Stdout of *cmd1* is used as input to *cmd2*
`cmd1 |&cmd2`	Stderr of *cmd1* is used as input to *cmd2*
`cmdpart1 \ cmdpart2`	Continue command on next line
`cmd1; cmd2`	Execute *cmd1* then *cmd2*

À garder dans un coin de votre mémoire…

Processes *Pour le labo 2*

`ps`	Show processes of user
`ps -e`	Show all processes
`ps -fA`	Show all processes in detail
`top`	Show processes in real-time
`cmd &`	Run command in background
`Ctrl-c`	Stop (kill) currently active process
`Ctrl-z`	Suspend currently active process
`bg`	Place suspended process in background
`fg`	Bring background process to foreground
`kill pid`	Kill process with process id *pid*
`kill -9 pid`	Kill process *pid* (ungraceful)

Bash Shortcuts

`Ctrl-k`	Cut line of text
`Ctrl-y`	Paste line of text
`Ctrl-e`	Go to end of line
`Ctrl-a`	Go to start of line
`TAB`	Autocomplete command/file
`TAB-TAB`	Show list of possible autocompletes
`up arrow`	Scroll previous commands
`down arrow`	Scroll previous commands
`history`	List recent commands
`!!`	Repeat last command
`!N`	Execute command *N* from history
`!abc:p`	Print last command starting with *abc*
`!abc`	Execute last command starting with *abc*

Testons nos connaissances…

Afin de vous familiariser un peu avec les commandes de bases, trouvez les commandes permettant de répondre aux besoins suivants :

1. Afficher l'aide sur l'utilisation du manuel

2. Connaître l'utilisateur courant de la session sur laquelle vous êtes (deux solutions)

3. Créer un répertoire nommé Exercices

4. Descendre dans l'arborescence afin d'être dans ce répertoire
   En quoi l'utilisation de [TAB] peut vous faire gagner du temps ?

5. Créer un fichier vide

6. Lister les éléments (dossiers et fichiers) du répertoire Exercices

7. Se positionner dans le répertoire racine

8. À partir du répertoire racine, créer un répertoire Textes dans votre répertoire (home)

9. Se positionner dans le répertoire Textes

10. Copier le fichier premier.txt d'origine dans le répertoire Textes

11. Nettoyez votre répertoire de travail en effaçant tous les fichiers et les répertoires créés pour cet exercice ! À vous d'en trouver la/les commande(s)…

VI

VI (prononcez vie-ae ou [vi:ai]) est l'éditeur de texte de base sous LINUX. Vous risquez d'avoir à vous en servir au plus mauvais moment, c'est-à-dire lorsque plus rien d'autre ne fonctionne. Les autres éditeurs les plus connus sont VIM (une amélioration de VI), GNU Emacs, pico, Gedit…

Lancer VI

La commande qui permet de lancer VI est la suivante :

```
$ vi
```

La page d'accueil de VI est alors affiche telle que sur la figure 1.

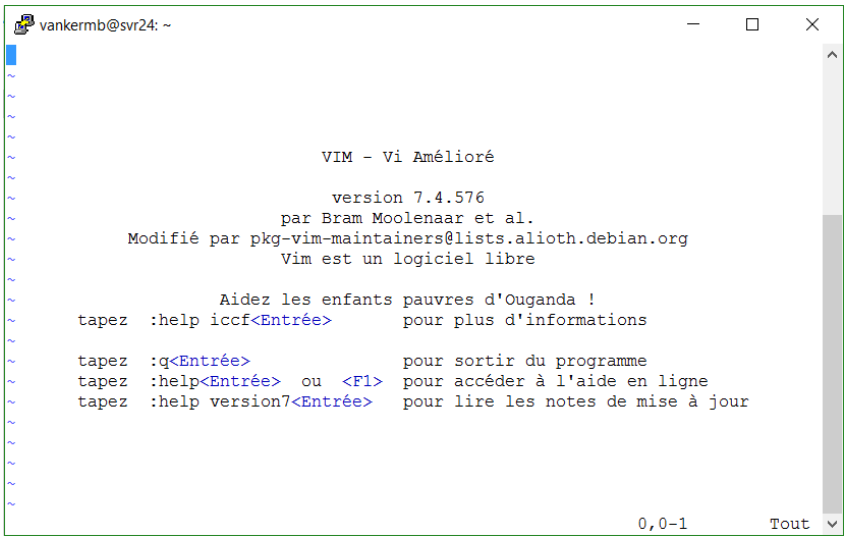

Figure 7 - Fenêtre d'accueil de VI

Vous pouvez ouvrir un fichier existant au moyen de la commande suivante :

```
$ vi [nom_du_fichier]
```

Cette commande permet également de créer un fichier lors de son ouverture.

Les modes

L'éditeur VI a deux modes de fonctionnement :

- Mode commande
  Dans ce mode, l'utilisateur ne saisit que des commandes (souvent représentées par un caractère). C'est le mode par défaut.
- Mode saisie ou insertion
  Dans ce mode, l'utilisateur se limite à saisir du texte.
  Attention, évitez d'utiliser le clavier numérique lorsque vous travaillez sous VI!

Lorsque VI s'ouvre, il est en mode commande. Pour passer en mode saisie,

- tapez [i] ou [Insert] pour insérer du texte l'endroit où se trouve le curseur
- tapez [a] pour ajouter du texte la fin d'une ligne

Pour quitter ce mode, tapez [Esc].

Un "cheat sheet" est mise à votre disposition sur le "*CommonProfsEtudiants*"…

Premier programme en C

Dans votre répertoire de travail, ouvrez un nouveau fichier `hello.c` dans le répertoire `Labo2` via les commandes suivantes :

```
$ cd ~
$ mkdir Labo2
$ cd L[TAB]
$ vi hello.c
```

Lorsque vous êtes dans VI, tapez sur 'i' afin de passer en mode insertion.

Écrivez les lignes de code lisibles sur la figure 2.

Figure 8 - Fichier hello.c dans VI

N'oubliez pas de terminer par un passage à la ligne !

Pour sauver ce fichier et quitter VI, il faut repasser en mode commande via [ESC], puis entrer la commande suivante :

```
:wq
```

Pour *write and quit*.

S'il refuse de sortir, essayez ceci :

```
:wq!
```

Passer de Notepad++ à Linux

Afin d'éviter VI et toute manipulation via FTP, vous pouvez travailler sous Windows avec n'importe quel éditeur de code. Recopiez le code de l'exemple précédent et faites une copie de ce code avec [CTRL+c].

Entrez la commande suivante dans PuTTY, en choisissant un nom de fichier approprié :

```
$ cat > nom_du_fichier.c
```

La commande `cat` permet de lister un fichier… Si aucun fichier ne lui est fourni, c'est le `stdin` qui est lu… Faites un *clic droit* pour coller le code/texte précédemment copié et appuyez sur [CTRL+d] pour envoyer ce code/texte au programme qui est en train de lire, en l'occurrence la commande `cat` !

GCC

GCC – GNU Compiler Collection – est une collection de logiciels libres intégrés capables de compiler divers langages de programmation, dont C, C++, Objective-C, Java, Ada et Fortran.

La commande pour compiler le fichier hello.c créé avec VI est

```
$ gcc hello.c -o hello
```

L'option `-o` permet de spécifier le nom du fichier résultant de la compilation. Bien d'autres options sont disponibles et sont listées dans le manuel.

Pour exécuter le programme `hello`, la commande est la suivante :

```
$ ./hello
```

La variable d'environnement PATH contient la liste des répertoires dans lesquels le système va chercher les exécutables (idem Windows). Elle ne contient cependant pas le répertoire courant par défaut. Il faut donc préciser un *pathname* pour l'exécution. En ajoutant "`./`", on impose le répertoire courant comme *pathname*.

Le résultat de ces commandes est présenté à la figure 3.

```
[piroc@svr24 piroc]$ gcc hello.c -o hello
[piroc@svr24 piroc]$ ./hello
Hello world !
[piroc@svr24 piroc]$
```

Figure 9 - Compiler et exécuter